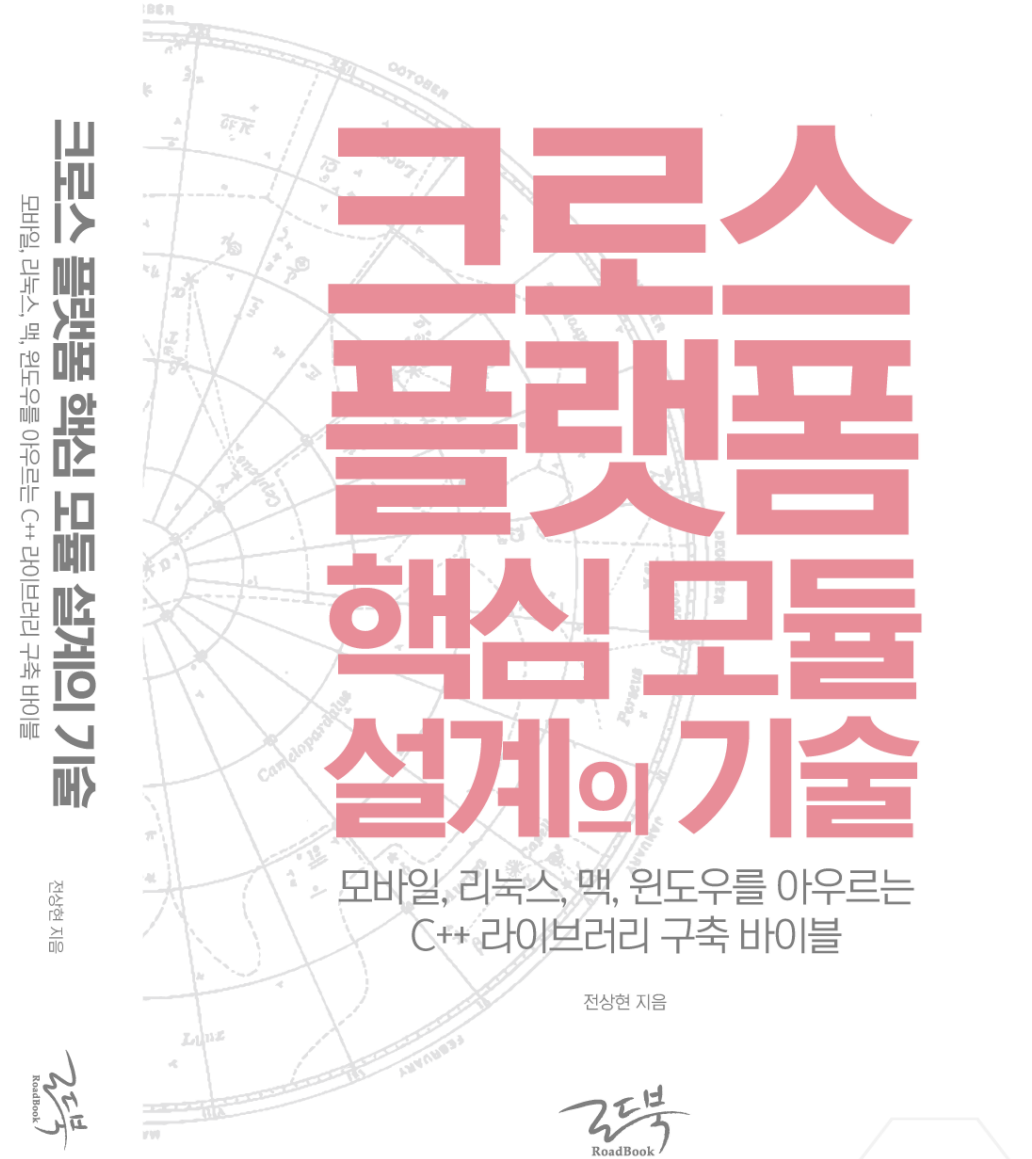
시스템 프로그래밍4

S-개발자 4기 2026-03-11(수)

서울 송파구 동남로 130, 2층 제 4강의실

악성코드검거단 MALWARE ARREST TEAM 대표이사 전상현

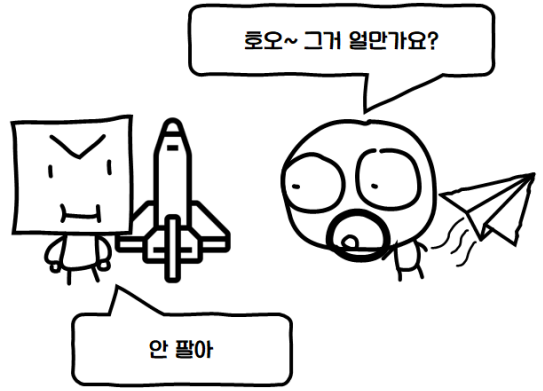


들어가기 전에 수업 개요

과정명	시스템 프로그래밍	강사명	전상현
강의시간	24시간 (4일 x 6시간)		
강의목표	컴퓨터 시스템을 이해한다. 나아가 다양한 시스템을 제어하는 C++ 프로그래밍 기술을 익힌다.		
평가방식	시험 50%, 실습 50%		

강의내용

시간(H)	제목	내용
1H	오리엔테이션	강사 소개 후 간단한 퀴즈와 함께 실행파일의 구조를 이해한다.
2H	변수형과 유니코드	시스템에 존재하는 모든 종류의 변수형을 알아본다. 다국어를 표현하는 문자체계인 유니코드를 이해한다.
3H	개발환경 구축하기	윈도우/리눅스(wsl)에 빌드할 수 있는 환경을 구성한다. 기본적인 개발툴 사용법을 익힌다.
2H	문자열 정복하기	C++과 친해지기 위한 코드를 작성해본다. VisualStudio 디버깅 기술을 배운다.
2H	C++ 컴파일러/링커/디버거 정복하기	유니코드 상호 변환하는 함수를 이해한다. 문자열을 자유자재로 다루는 방법을 배운다.
2H	파일시스템 정복하기	플랫폼별 파일/디렉토리 접근 함수를 이해한다. 파일 시스템을 자유자재로 다루는 방법을 배운다.
2H	데이터 정복하기	STL 자료구조를 이해하고, 실무 응용 기술을 배운다. XML/JSON/INI 등을 자유자재로 다루는 방법을 배운다.
2H	메모리 정복하기	C++의 꽃, 스택 메모리의 장단점을 알아본다. 4가지 영역의 메모리를 자유자재로 활용한다.
2H	소켓 정복하기	UDP/TCP 통신, DATAGRAM/STREAM의 차이를 이해한다. 소켓 통신 시스템을 자유자재로 다루는 방법을 배운다.
6H	**최종실습**	**앞에서 배운 지식과 기술을 활용하여 미니 프로젝트를 진행한다.**



코딩 실력이 곧 우주선이다.

우주같이 광활한 컴퓨터 세계로 여행을 떠나자.

<참고서적>

크로스플랫폼 핵심모듈 설계의 기술(2018) – 전상현, 로드북

아무도 알려주지 않은 C++ 코딩의 기술 (2023) – 전상현, 로드북

들어가기 전에 여러분은 얼마나 긴 코드를 작성할 수 있나요?

개발력(?)은 작성할 수 있는 코드의 길이에 비례합니다.

들어가기 전에 C++ 과 친하지 않다면 왜 그런지 이유를 들어볼까요?

아마도 대부분 C언어는 배워봤을 겁니다.

그런데 그 이후에 C++이 아닌 파이썬이나 자바를 선택하는 경우가 많습니다.

왜 C++을 포기할(선택하지 않을)까요?

알고 보면 C++은 매우 훌륭한 언어입니다.

다른 언어들은 C++보다 쓰기 쉽게 만들었지만 결코 그 장점을 모두 살릴 수가 없습니다.

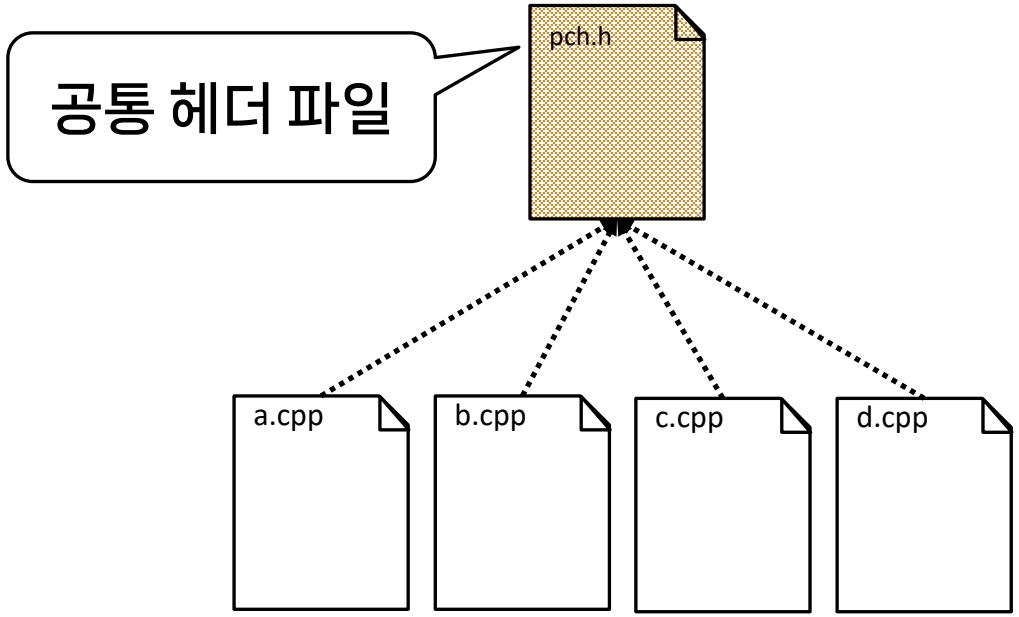
들어가기에 앞서 다음 퀴즈를 풀어봅시다.

1. 헤더 파일(.h)은 왜 필요할까?
2. 컴파일러와 링커의 역할은 무엇인가?
3. 문법만 이해하면 코딩할 수 있을까?
4. 디버깅은 무엇일까?

헤더 관리

도대체 C/C++에는 왜 헤더 파일(*.h)이 있을까?

헤더와 정의는 무엇인가?



미리 컴파일된 헤더는 자주 쓰이는 공통 구문들을 모아놓은 파일입니다.

모든 **cpp** 파일이 참조하는 헤더이므로 첫 줄에는 다음과 같이 쓰여야 합니다. **(반면 헤더파일은 절대로 pch.h 파일을 포함하면 안됨)**

#include “pch.h”

cpp 파일 첫 줄이 위와 같이 시작되지 않으면 “불안하고 찝찝한 느낌이 들 정도”로 강박적이어도 됩니다. 그만큼 중요합니다.

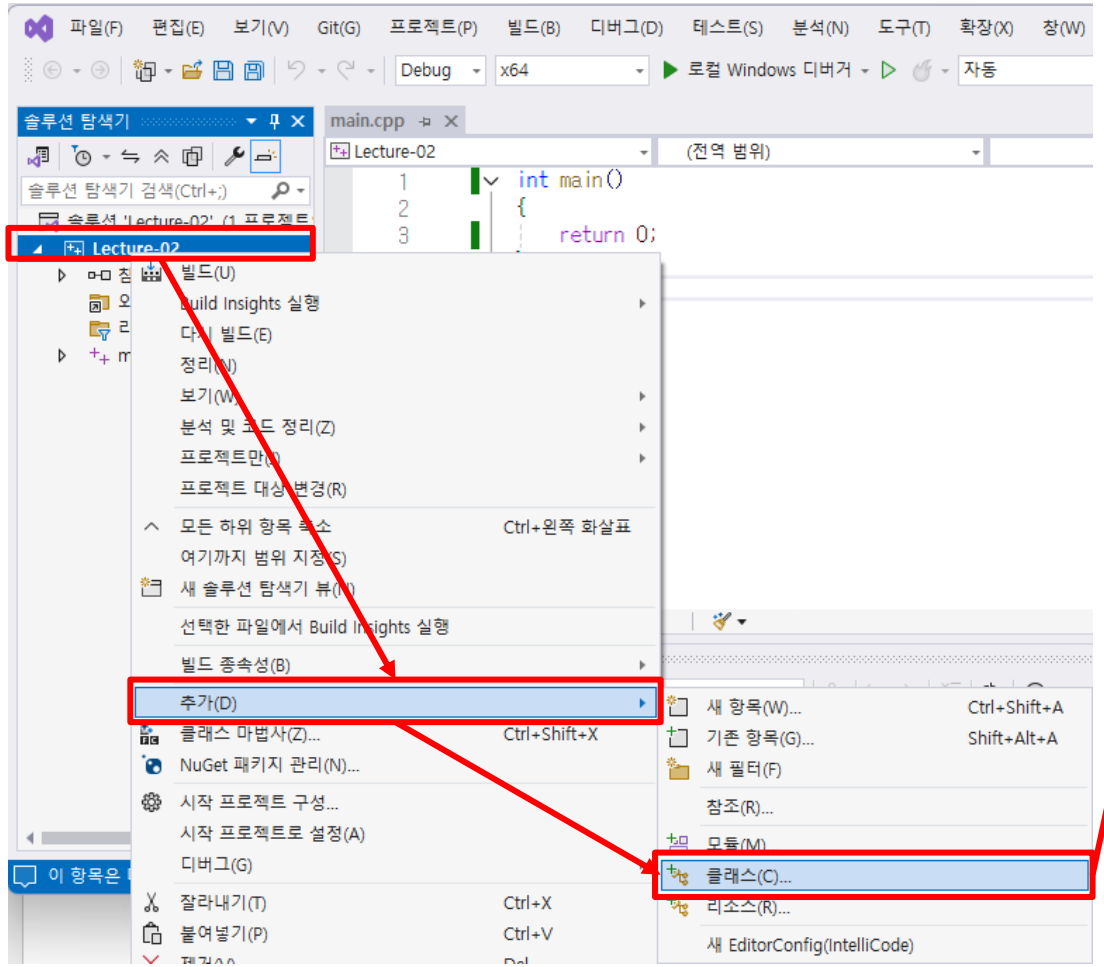
pch.h 파일의 예시는 다음과 같습니다. 대개 우리가 자주 쓰는 include 구문을 모아둡니다.

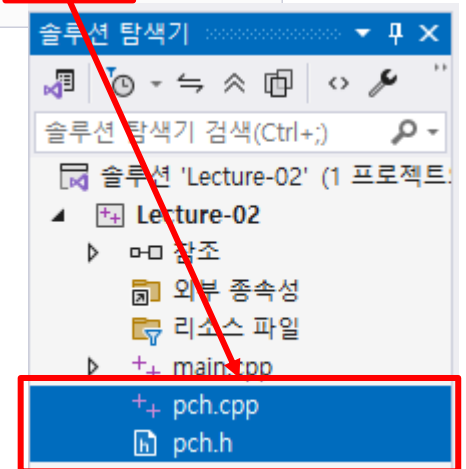
```
#include <stdio.h>

#include <iostream>
```

공통 헤더파일 만들기

클래스 추가는 헤더(h)와 소스(cpp) 파일을 동시에 만들어주는 기능입니다.

이 기능으로 pch.h와 pch.cpp 파일을 프로젝트에 생성합니다.





공통 헤더파일이란 **모든 cpp 파일이 포함하는 파일**을 말합니다. 비주얼 스튜디오에 미리 컴파일된 헤더를 설정하면 새로 만든 cpp 파일 첫머리에 stdafx.h나 pch.h 헤더가 포함되는 것을 알 수 있습니다.

단, 절대로 h파일이 포함하면 안됩니다. Cpp 파일만 참조하는 파일이라는 것을 명심하세요.

<pch.h>	<pch.cpp>
#pragma once #include <…> …	#include “pch.h”

<A.cpp>	<B.cpp>
#include “pch.h” …	#include “pch.h” …

앞서 만든 main.cpp도 예외는 아닙니다.

각 파일은 아래와 같은 모습이 나오게 될 것입니다.

<pch.h>

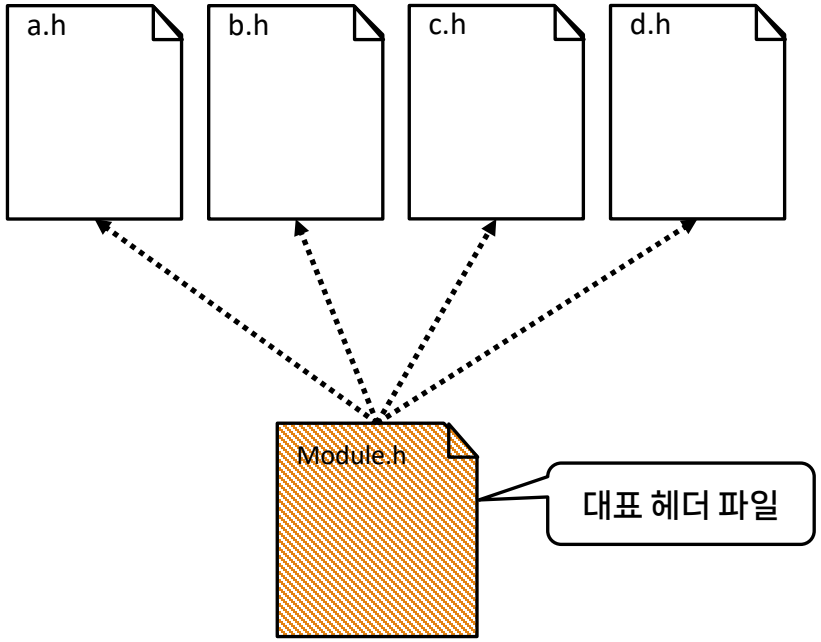
```
#pragma once

#include <string>

#include <list>

#include <vector>

#include <stdio.h>

#include <stdlib.h>

...
```

<main.cpp>

```
#include "pch.h"


int main()

{

    return 0;

}
```

대표 헤더 파일이란,

내가 만든 소스코드를 다른 사람에게 전달하기 위한 것입니다.



여러 개의 파일로 이루어졌다면 가져다 쓰는 입장에서 어떤 파일을 include 해야 할 지 난감합니다.
-> 전부 다 include 하기에는 상당히 번거롭겠지요.

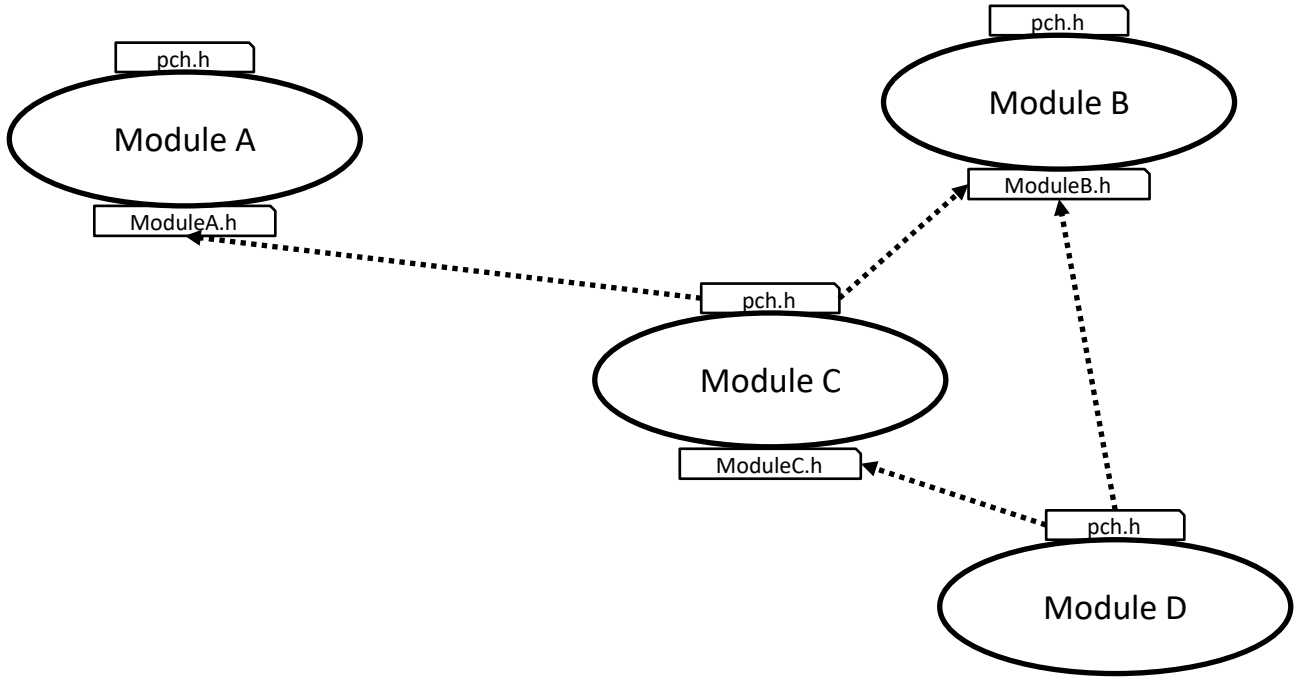
가장 좋은 방법은 대표 헤더 파일을 만들고 예제 코드도 하나 제공해주는 것입니다.

대표 헤더 파일은 다음과 같이 생겼을 것입니다. 즉, 내가 제공할 헤더 파일들을 대신 포함시켜주는 것이죠.

```
#pragma once
#include “a.h”
#include “b.h”
#include “c.h”
#include “d.h”
```

사용자는 “대표 헤더 파일”만 “공통 헤더 파일”에 include 해두면 모든 cpp 들이 사용할 수 있게 되는 것입니다.

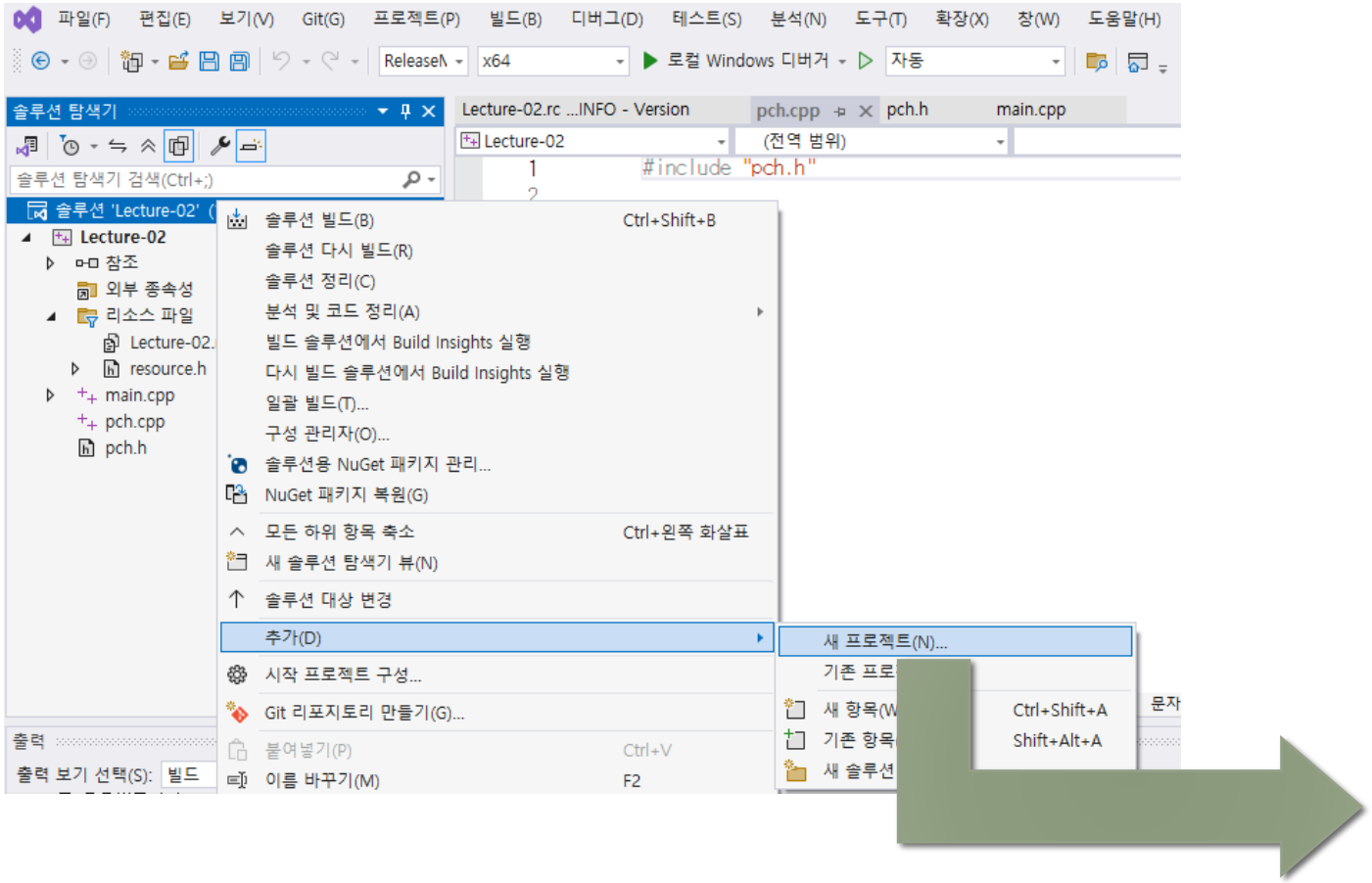
두 종류의 헤더파일을 모두 구성하면 여러 모듈(라이브러리)을 사용하는 프로그램은 다음과 같이 연결됩니다.



정적 라이브러리 만들기

정적 라이브러리 추가

기존에 만들어 둔 솔루션에 새로운 프로젝트를 추가합니다.

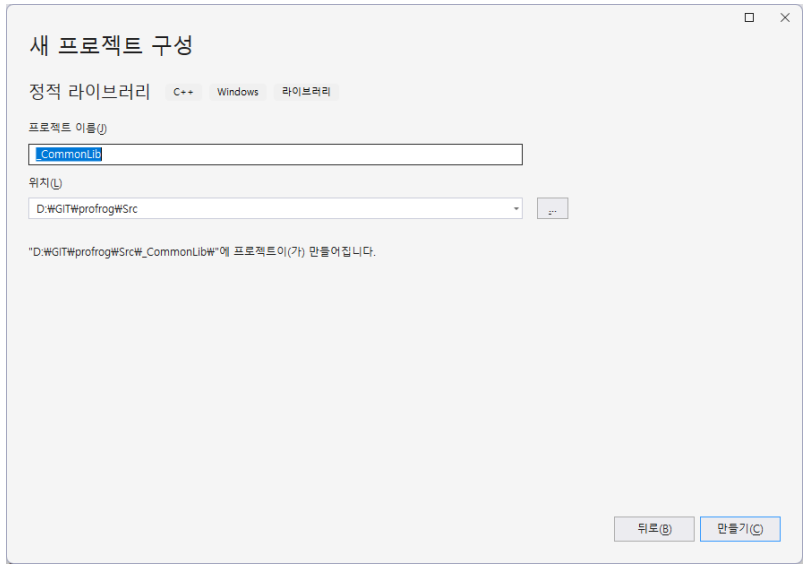


[C++ | Windows | 라이브러리] 항목에서

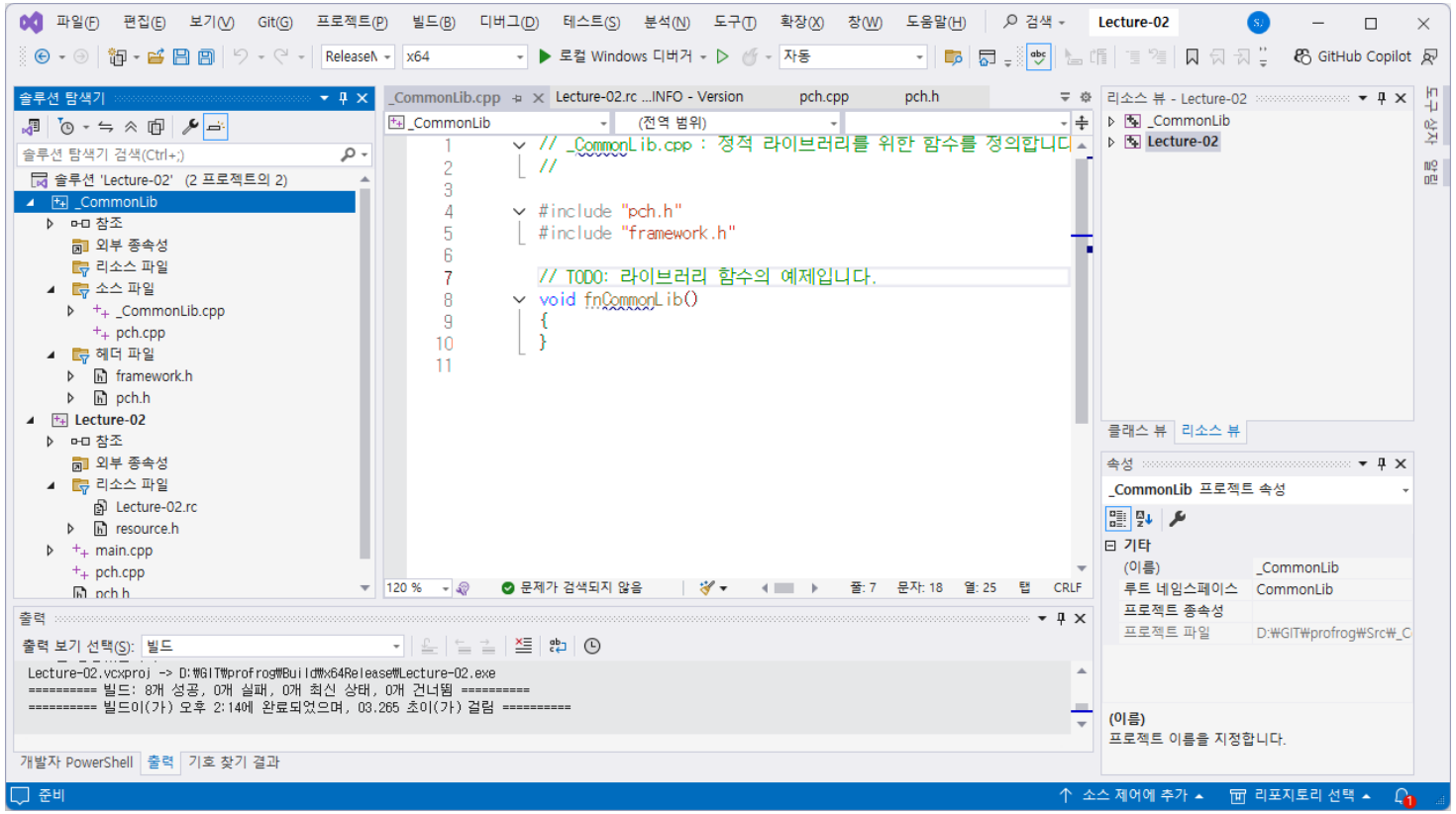
"정적 라이브러리 " 를 선택해봅니다.

정적 Lib. 기존에 만들어 둔 솔루션에 새로운 프로젝트를 추가해보겠습니다.

이름은 **_CommonLib** 이라고 지어보겠습니다.



그럼 왼쪽 솔루션 탐색기에 정적 라이브러리 프로젝트가 추가됩니다.



정적 Lib. 정적 라이브러리 기본 코드들 정리

마찬가지로 미리 작성된 코드들이 있습니다.

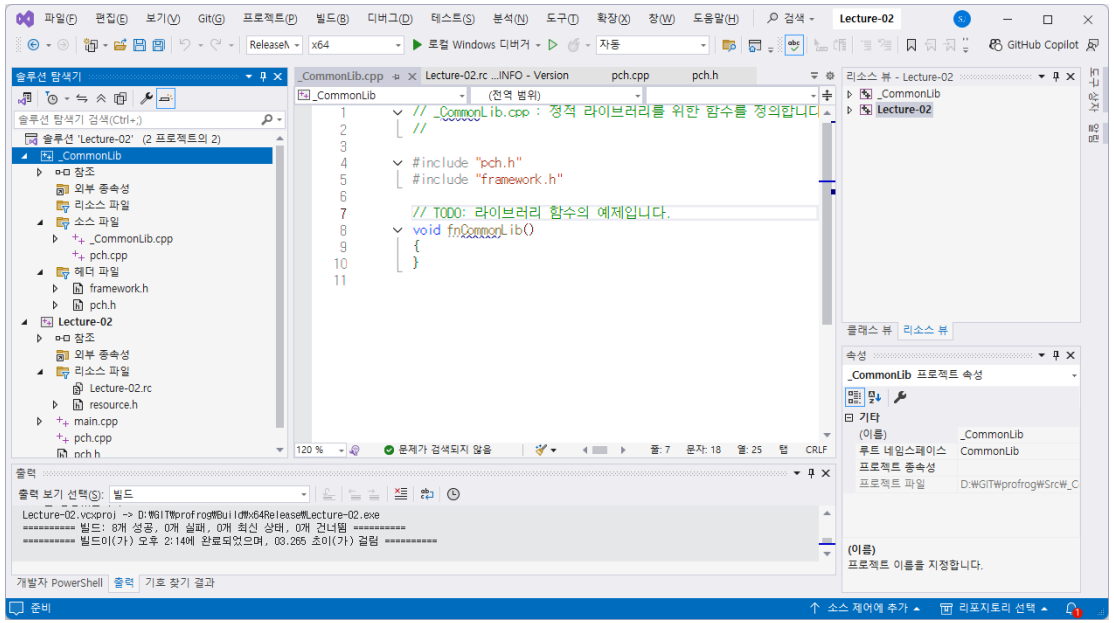
다음과 같은 수순으로 깨끗하게 정리합니다.

1. pch.h 및 pch.cpp를 제외한 파일들 삭제
 - _CommonLib.cpp은 hello world 와 같은 것으로 내가 쓴 코드가 아니므로 과감히 삭제
 - framework.h 에 선언된 다음과 같은 부분은

   "#define WIN32_LEAN_AND_MEAN"

   향후에 정확한 의미를 파악해서 쓰길 바랍니다.

   지금은 없어도 무방



2. pch.h의 내용을 전부 삭제하고 #pragma once만 남김

!!중요!! 앞으로 모든 헤더 파일은 항상 #pragma once로 시작합니다.

3. pch.cpp는 #include "pch.h" 외에 아무것도 남기지 않음

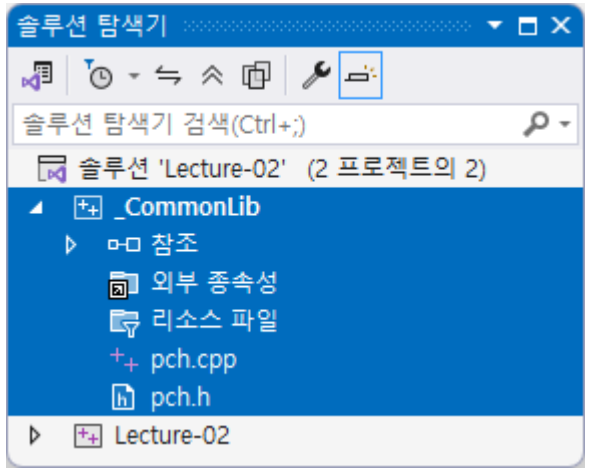
!!중요!! 앞으로 모든 pch.cpp는 위와 같이 아무 내용없이 include 한줄만 둡니다.

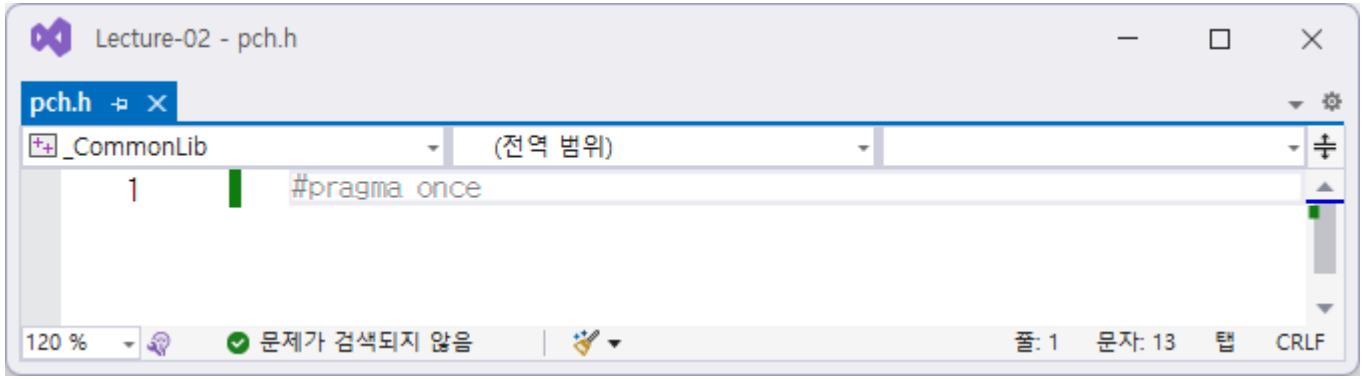
정적 라이브러리 기본 코드들 정리 cont`

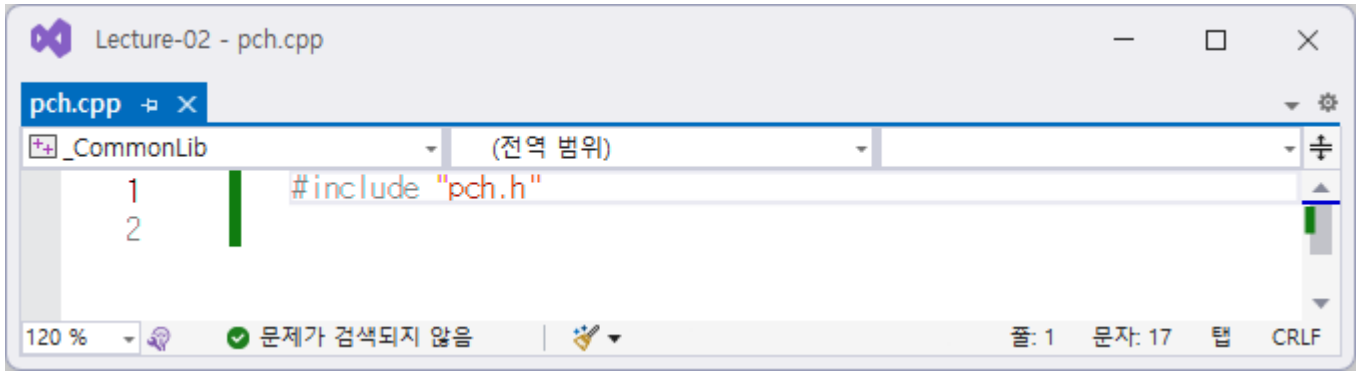
정리하자면 새로 만든 정적 라이브러리인 _CommonLib은 오른쪽과 같은 구조가 됩니다.

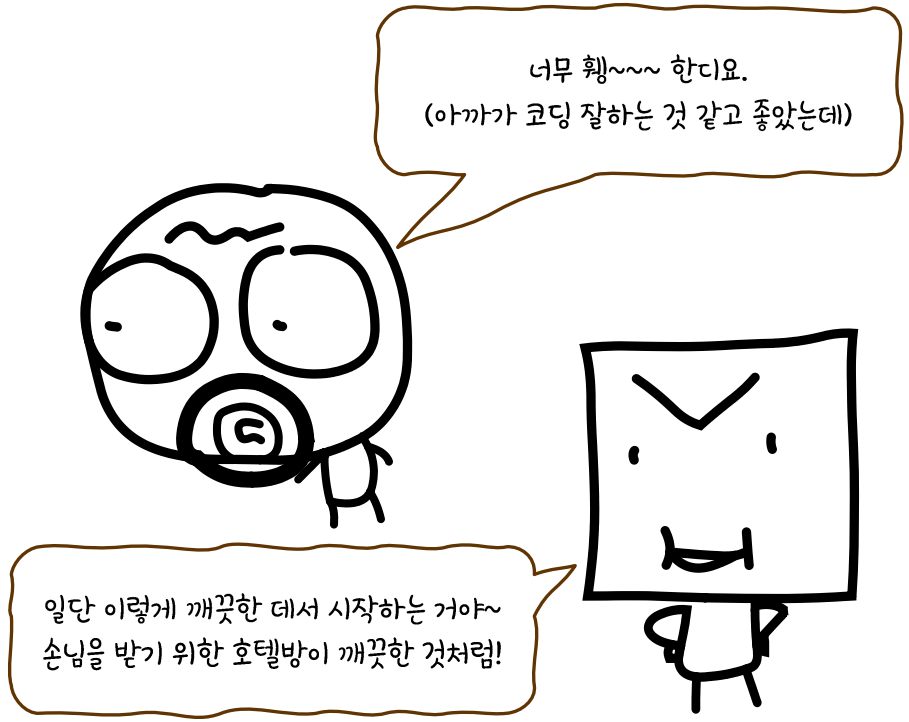
다 없애고 pch.h와 pch.cpp만 남긴 것이죠.

각 파일의 내용은 다음과 같습니다.

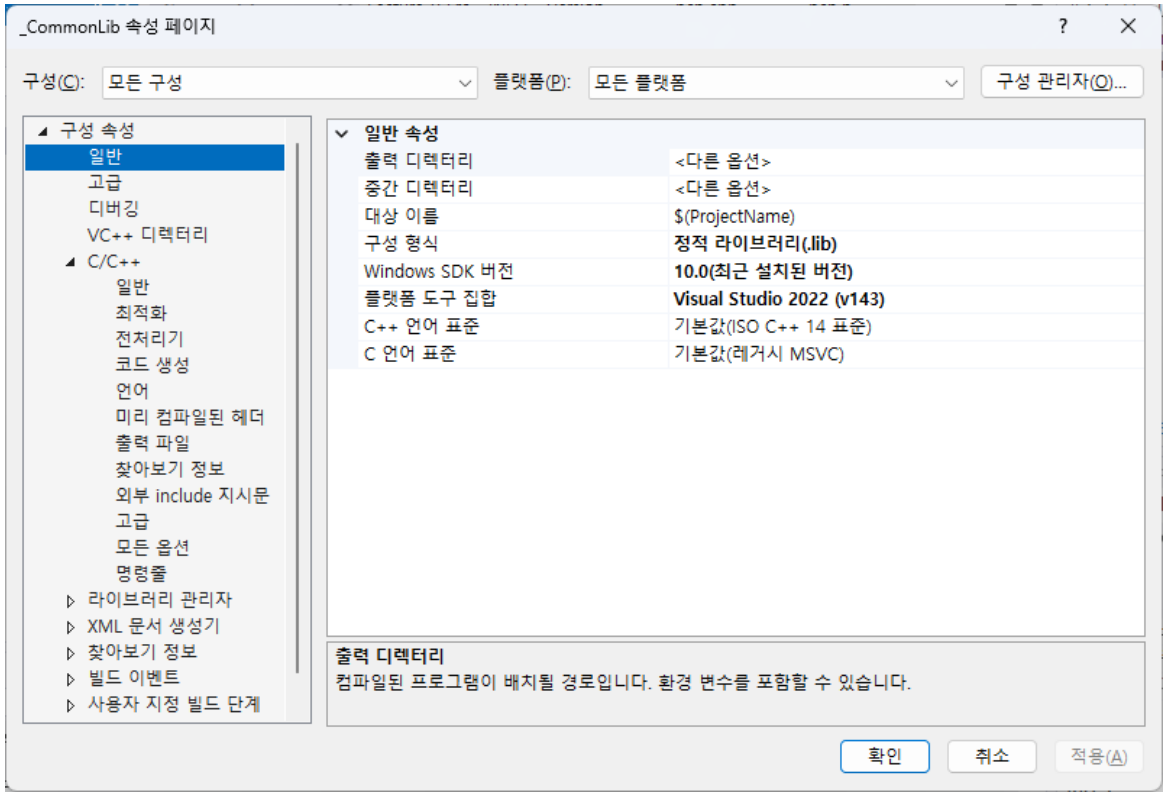








정적 라이브러리 빌드 경로 조정하기



[속성] -> [일반] 탭에 가보면 새로 만든 프로젝트라 출력 디렉터리와 중간 디렉터리가 또 초기화된 상태로 있습니다.

마찬가지로 ***[모든 구성]***, ***[모든 플랫폼]***을 선택해서 각 항목을 다음과 같이 변경합니다.

출력 디렉터리	$(ProjectDir)\..\..\Build\$(Platform)$(Configuration)\
중간 디렉터리	$(ProjectDir)\..\..\Output\$(Platform)$(Configuration)\$(ProjectName)\

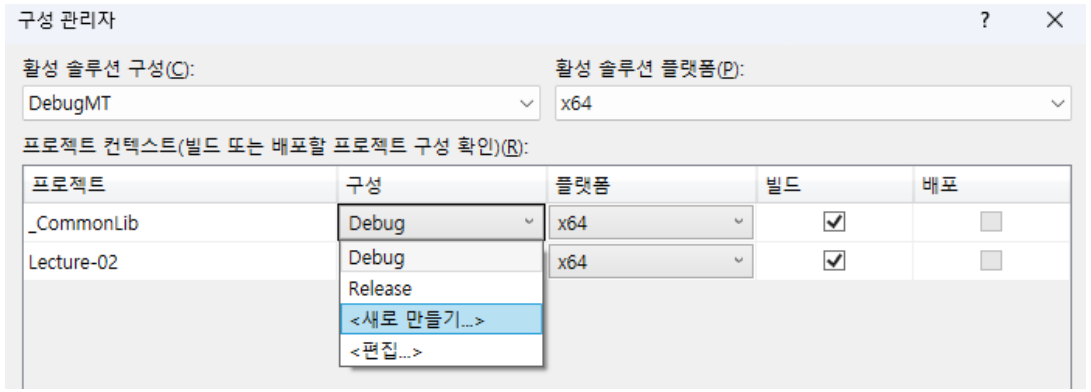
정적 라이브러리 MT 구성 추가하기

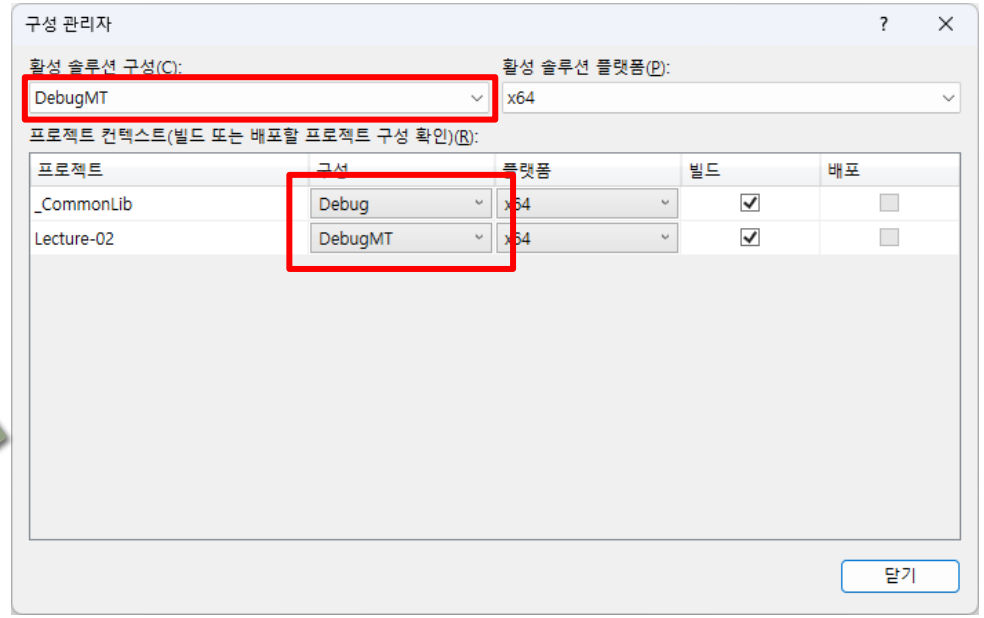
그리고 MT 구성을 추가하는 것도 잊으면 안되겠죠~~

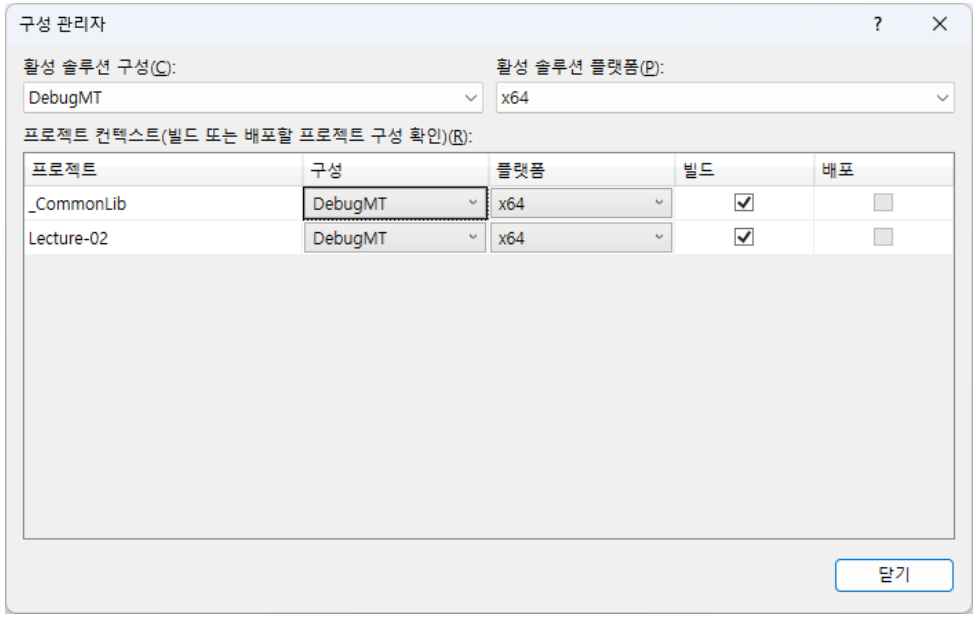
이번엔 [빌드] -> [구성 관리자] 메뉴를 눌러봅니다.

4가지의 활성 솔루션 구성에 맞게 각 프로젝트도 동일하게 구성돼야 하는데, 새로 만든 프로젝트인 정적 라이브러리는 존재하지 않으므로 이렇게 어긋나게 구성되어 있습니다. 콤보박스를 눌러서 <새로 만들기>로 DebugMT와 ReleaseMT를 만들어 동일한 구성으로 맞춰줍니다.

8가지 경우의 수에 대해 모두 동일한 구성이 되도록 맞춰주세요.

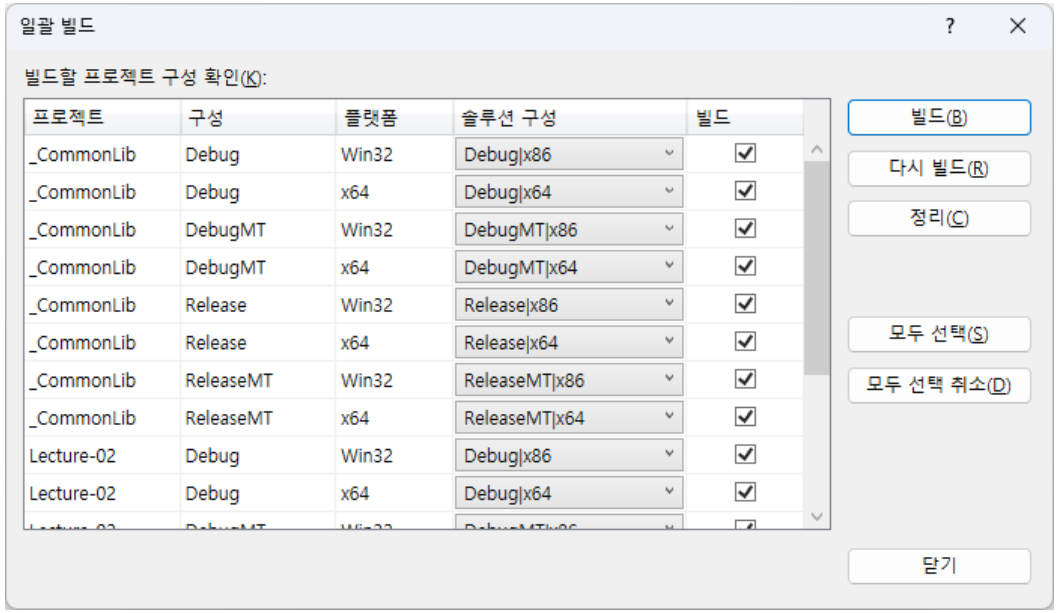


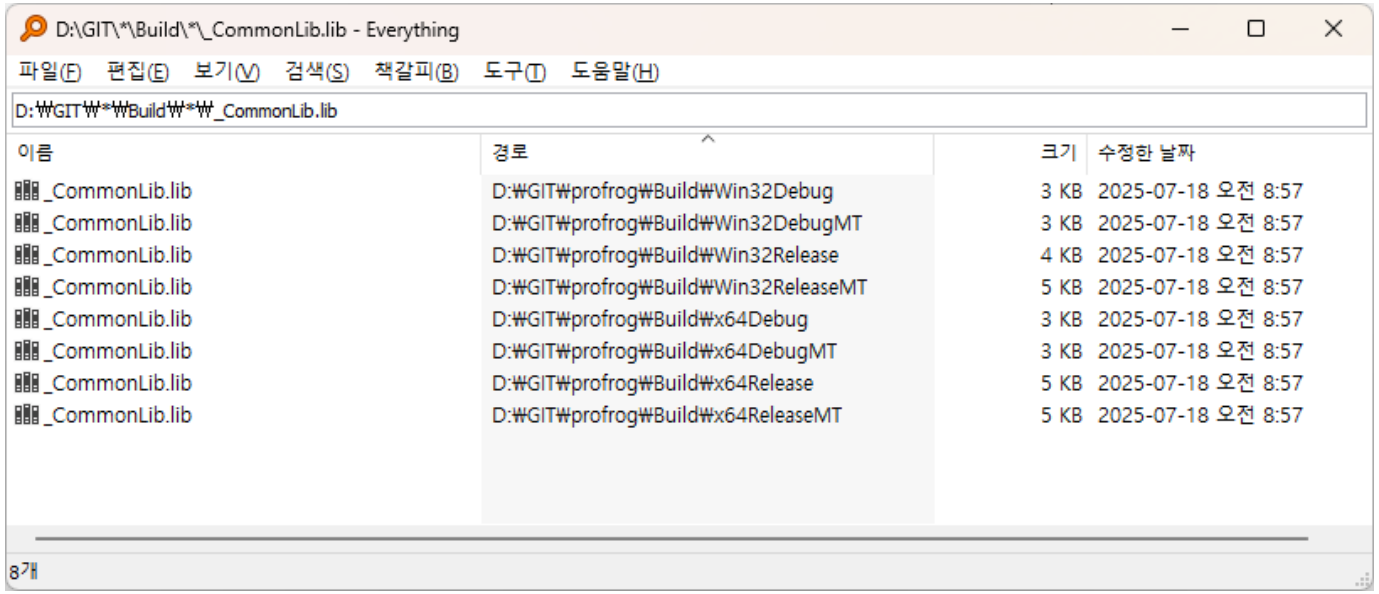




정적 Lib. 정적 라이브러리 일괄 빌드 후 생성된 파일들 확인하기

기억하시죠? [빌드] -> [일괄 빌드] 메뉴로 늘어가서 모든 구성을 선택해서 빌드해 봅니다. 잘 설정이 되었다면 8색조 lib 파일이 생성됩니다.

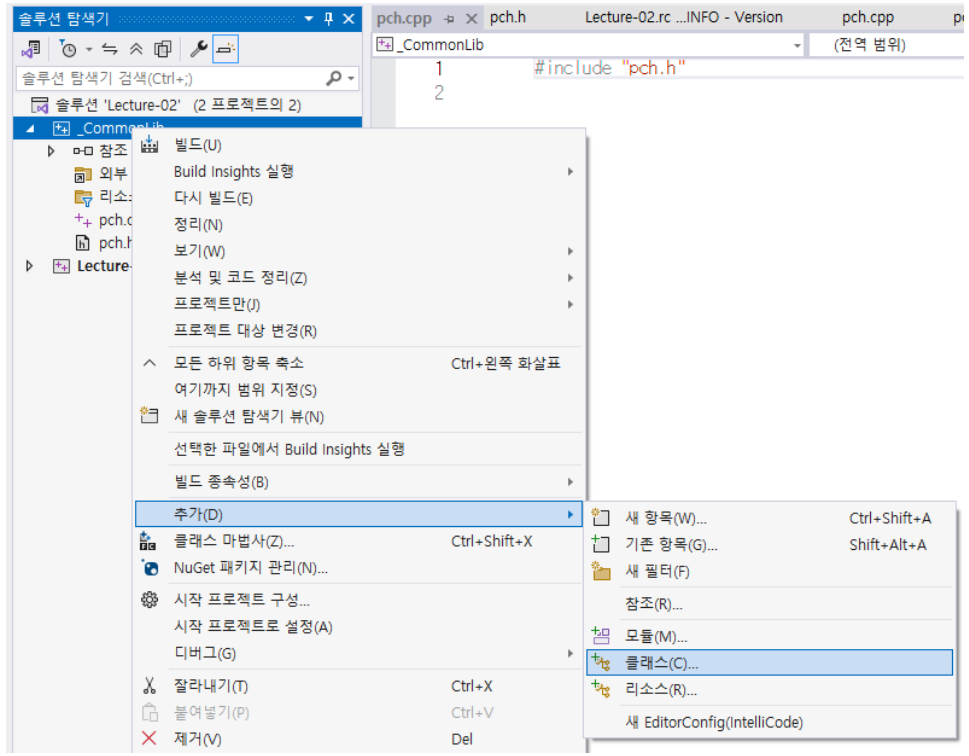




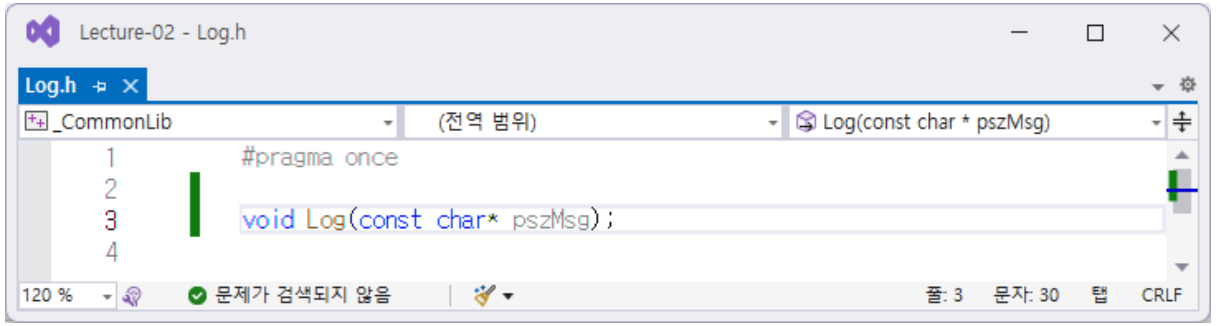
정적 라이브러리에 함수 구현해보기

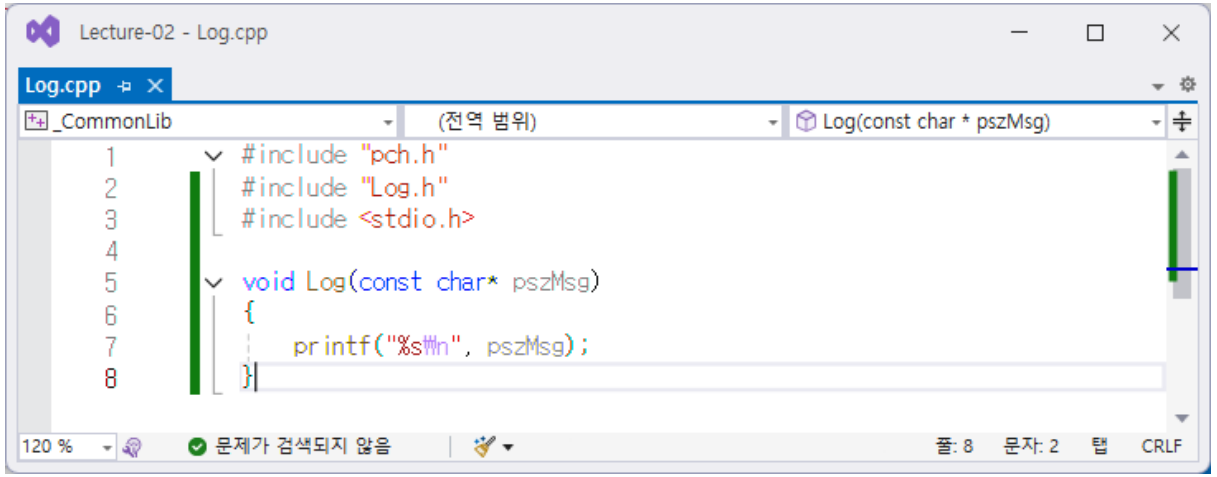
이제 정적 라이브러리에 간단한 함수를 구현해보겠습니다.

[_CommonLib 우클릭] -> [추가] -> [클래스] 로 Log.h와 Log.cpp 파일을 추가해봅니다.



그리고 다음과 같이 간단한 Log 함수를 구현해 봅니다.



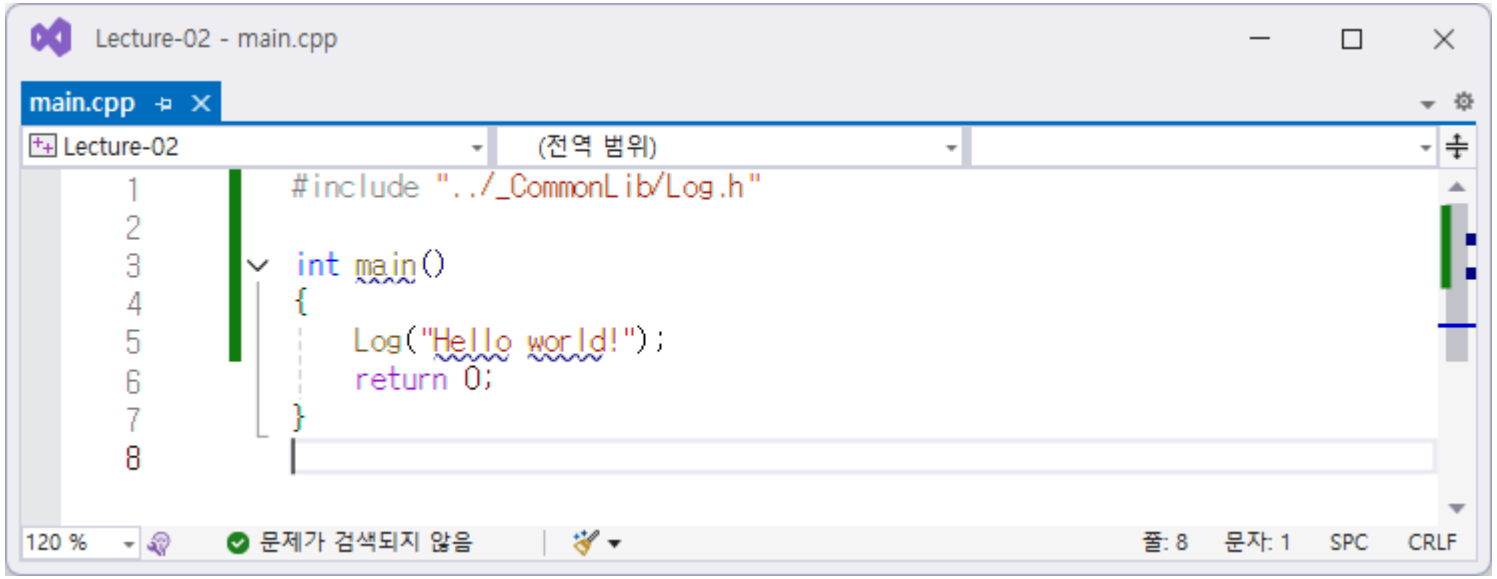


정적 라이브러리 사용하기

앞의 Log 함수를 Lecture-02 프로젝트에서 사용해봅니다.

처음에 작성했던 Lecture-02 프로젝트의 main.cpp에 다음과 같이 Log 함수를 이용해 Hello world를 출력해봅니다.

물론 include 부터 작성해야겠죠.



그 다음 빌드(CTRL + SHIFT + B)하면...

오류가 발생합니다.

당연하죠. 아직 내가 만든 정적 라이브러리를 링크하지 않았거든요! 이어지는 내용을 참고하세요.

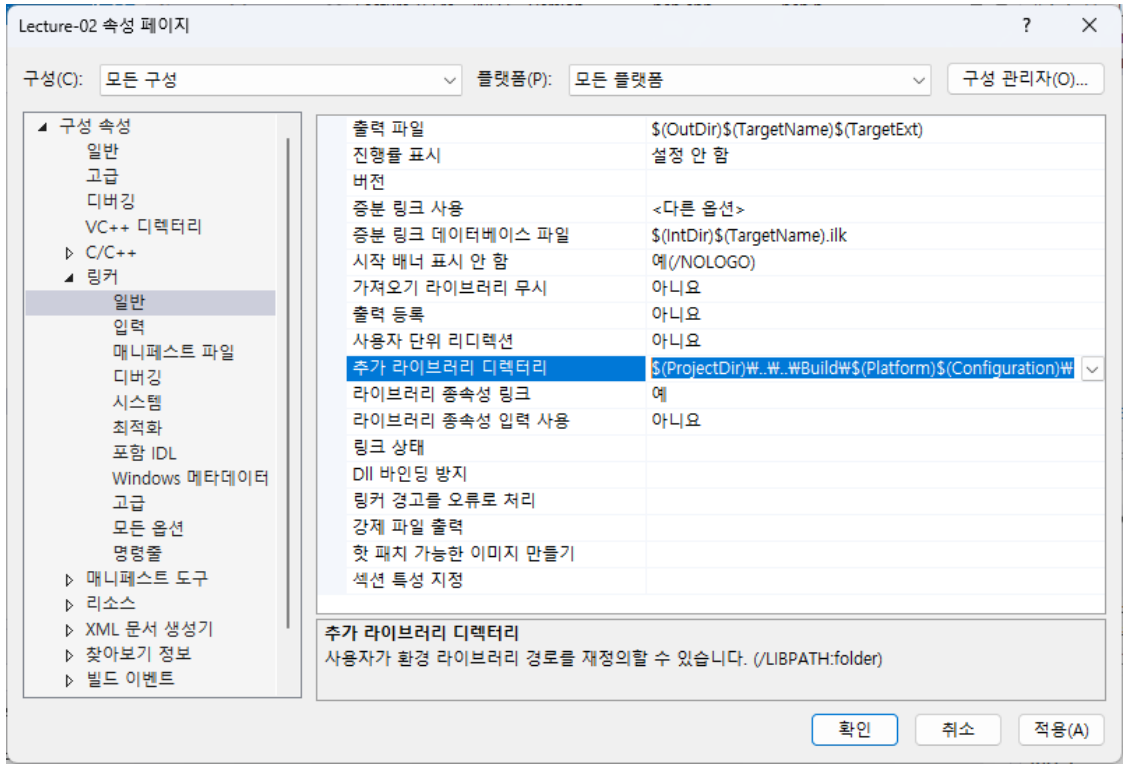
```
오전 9:06에 빌드를 시작함...
1>------ 빌드 시작: 프로젝트: Lecture-02, 구성: DebugMT x64 ------
1>main.obj : error LNK2019: "void __cdecl Log(char const *)" (?Log@@YAXPEBD@Z)main 함수에서 참조되는 확인할 수 없는 외부 기호
1>D:₩GIT₩profrog₩Src₩Lecture-02₩₩..₩..₩Build₩x64DebugMT₩Lecture-02.exe : fatal error LNK1120: 1개의 확인할 수 없는 외부 참조입니다.
1>"Lecture-02.vcxproj" 프로젝트를 빌드했습니다. - 실패
========== 빌드: 0개 성공, 1개 실패, 1개 최신 상태, 0개 건너뜀 ==========
========== 빌드이(가) 오전 9:06에 완료되었으며, 00.447 초이(가) 걸림 ==========
```

정적 라이브러리 링크하는 3가지 방법

링크하려는 실행 프로젝트의 [속성] -> [링커] -> [일반] -> [추가 라이브러리 디렉터리]에 빌드된 라이브러리 경로를 알려줍니다.

우리의 경우에는 실행파일과 동일한 디렉토리에 각각 lib 파일을 생성했으니까 다음과 같은 경로가 되겠네요.

$(ProjectDir)\..\..\Build\$(Platform)$(Configuration)\



그 이후, 정적 라이브러리 파일을 지정하는 3가지 방법이 있습니다.

1) 소스코드로 작성
2) 라이브러리 종속성에 추가
3) 참조 프로젝트로 추가

정적 라이브러리 링크하는 3가지 방법 – 1) 소스코드로 작성

<pch.cpp> 파일 하단에 다음과 같이 코드를 작성합니다.

```
#pragma comment(lib, "_CommonLib.lib")
```

기본적으로 C++ 코드에서 #으로 시작하는 명령은 컴파일러가 읽는 명령이라고 보면 됩니다.

이 경우에도 컴파일러에게 _CommonLib.lib 파일을 링크 시점에 가져다 쓰라고 알려주는 것입니다.

파일경로 없이 파일명만 알려줘도 되느냐고요?

➔ 네, 앞서 프로젝트 속성에서 라이브러리 디렉토리 경로를 이미 알려줬으니까요.

➔ 컴파일러는 앞서 알려준 디렉토리에서 해당 파일명을 추적합니다.

이 방식은 VisualStudio 컴파일러만 인식한다는 점입니다.

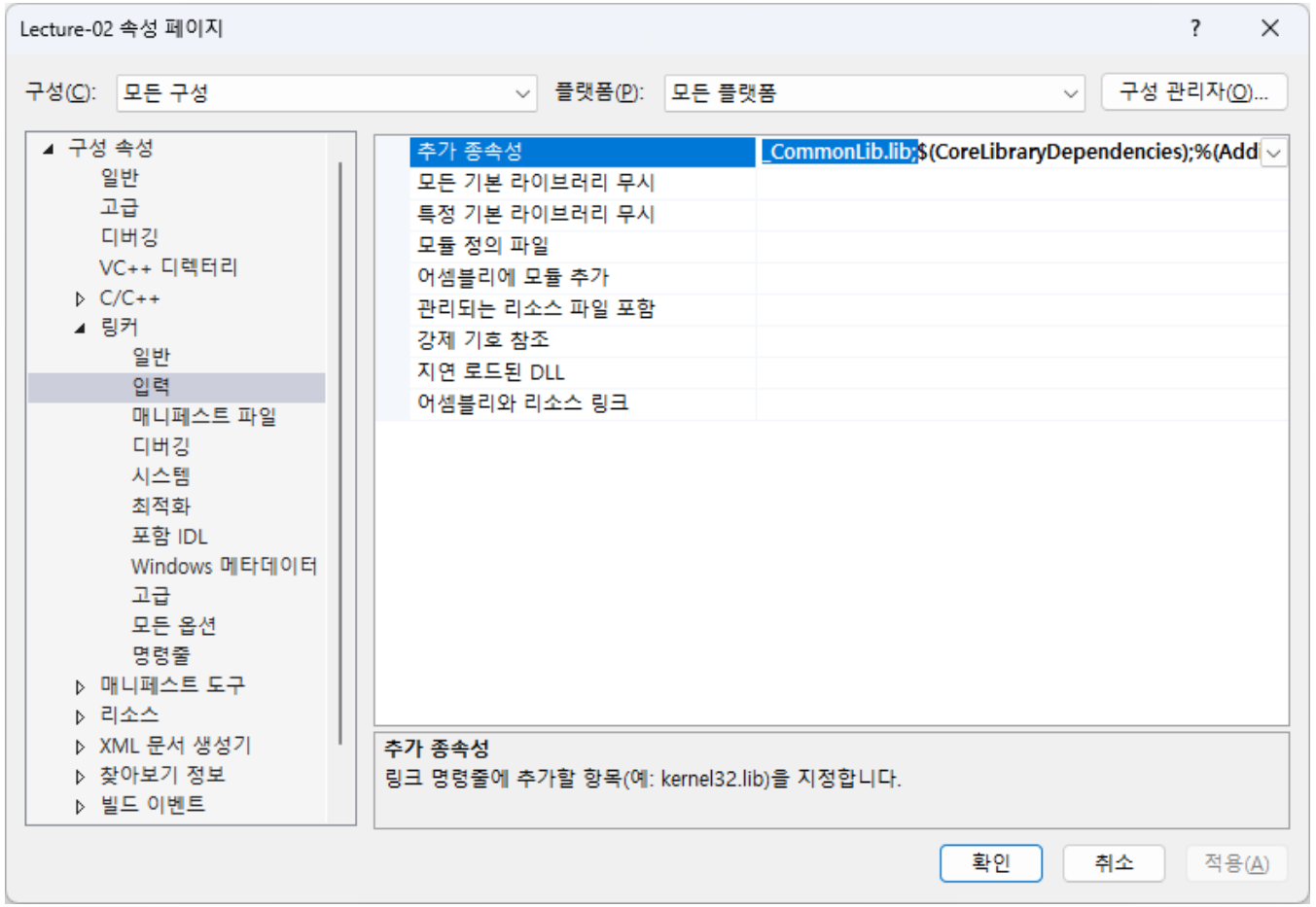
같은 코드를 linux에서 빌드하면 동작하지 않습니다.(오류는 나지 않지만 컴파일러가 무시합니다.)

정적 라이브러리 링크하는 3가지 방법 – 2) 라이브러리 종속성 추가

이 방식은 추가 라이브러리 경로를 지정한 것처럼 VisualStudio 프로젝트 파일에 기입하는 방법입니다.

프로젝트의 [속성] -> [링커] -> [입력] -> [추가 종속성] 항목에 _CommonLib.lib; 을 추가합니다.

여러 라이브러리를 지정할 수 있게 각 이름은 세미콜론으로 구분해야 합니다.



정적 라이브러리 링크하는 3가지 방법 – 3) 참조 프로젝트로 추가

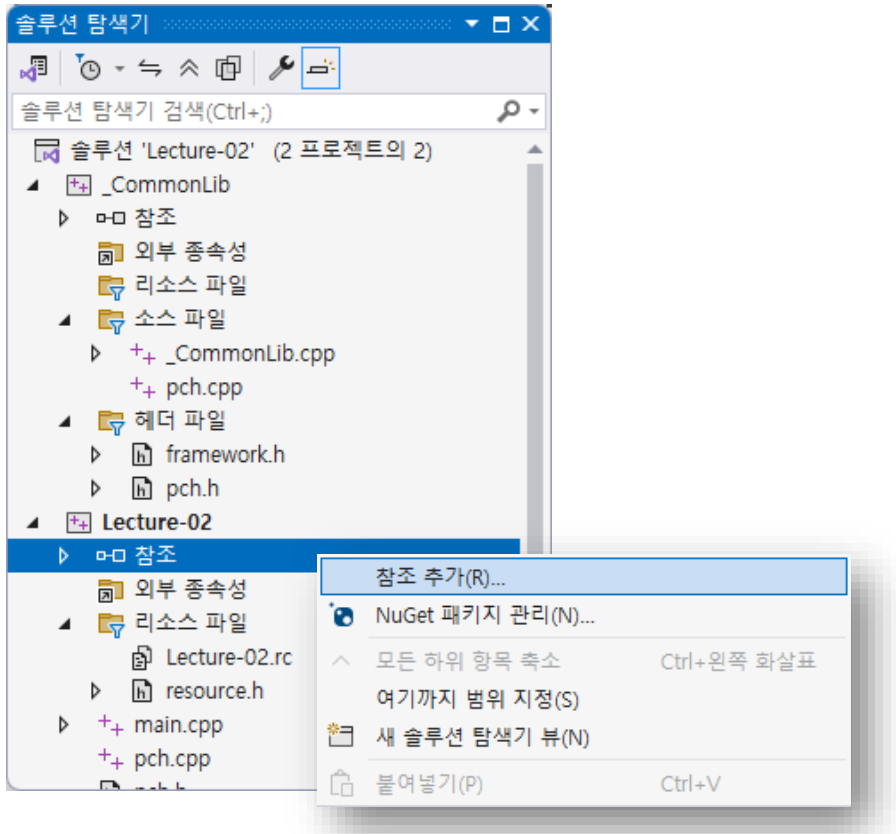
가장 편리한 방법입니다.

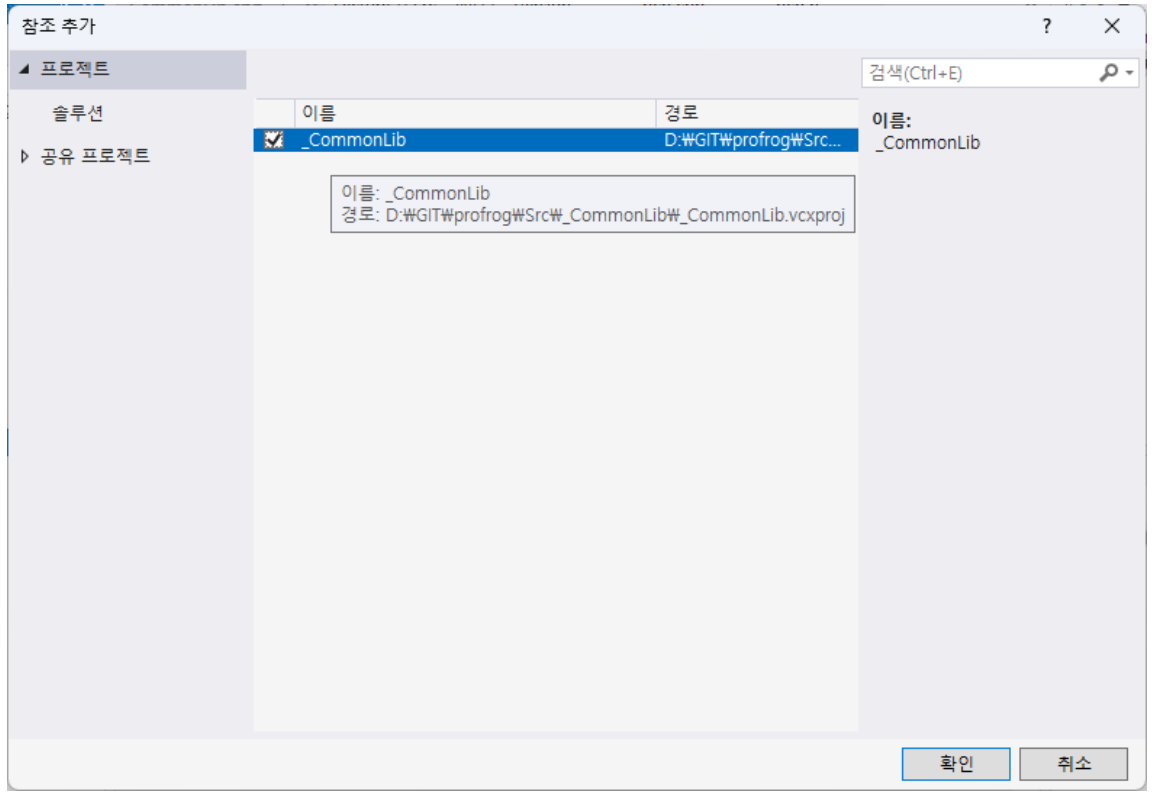
다만, 같은 솔루션에 존재하는 프로젝트에 한해 사용할 수 있습니다.

솔루션 탐색기에서 링크하려는 프로젝트를 보면 [참조]라는 항목이 있습니다.

우클릭 후 [참조 추가] 메뉴를 선택하면 우측 화면이 나타납니다.

여기서 링크하고 싶은 정적 라이브러리 프로젝트를 선택하면 이후 빌드시 자동으로 링크됩니다.

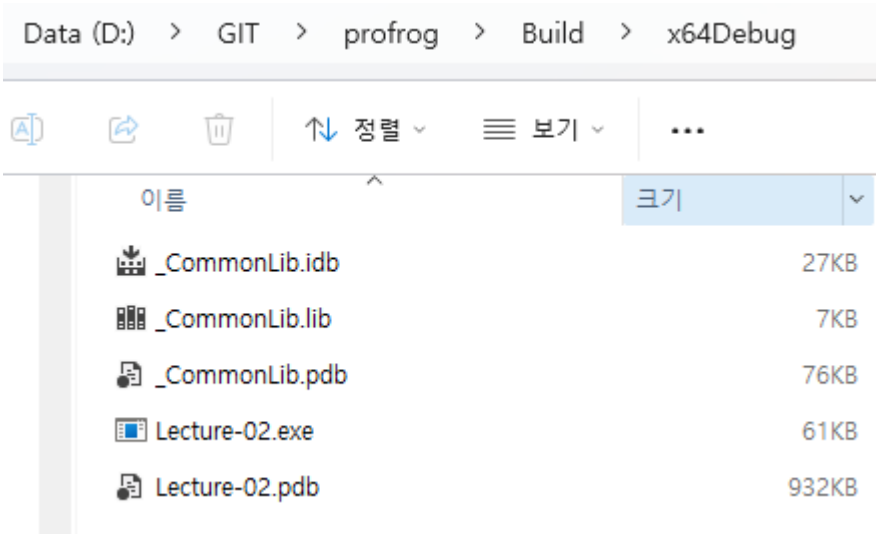




이제, CTRL + SHIFT + B 로 빌드해봅니다.

특별한 것이 없다면 빌드에 성공할 것이고,

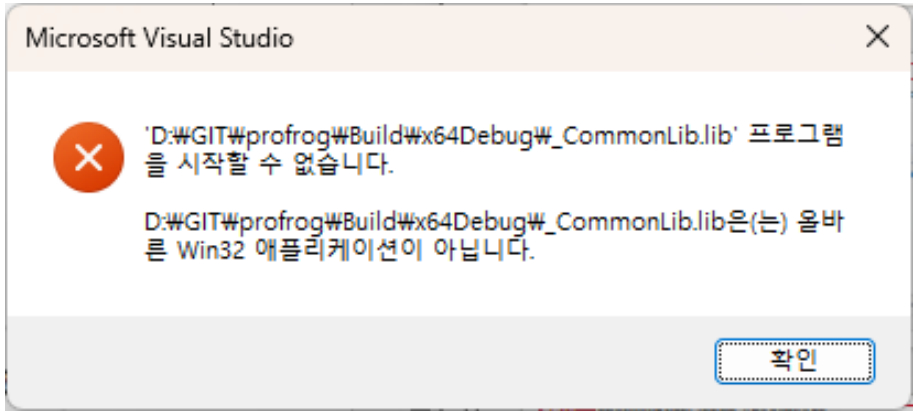
Build 경로에는 다음과 같이 lib과 exe 파일이 생성되었을 것입니다.



다중 프로젝트 관리

디버깅을 위해서 실행하기(F5 혹은 CTRL+F5) 버튼을 눌렀습니다.

아래 에러 메시지는 왜 발생했을까요?

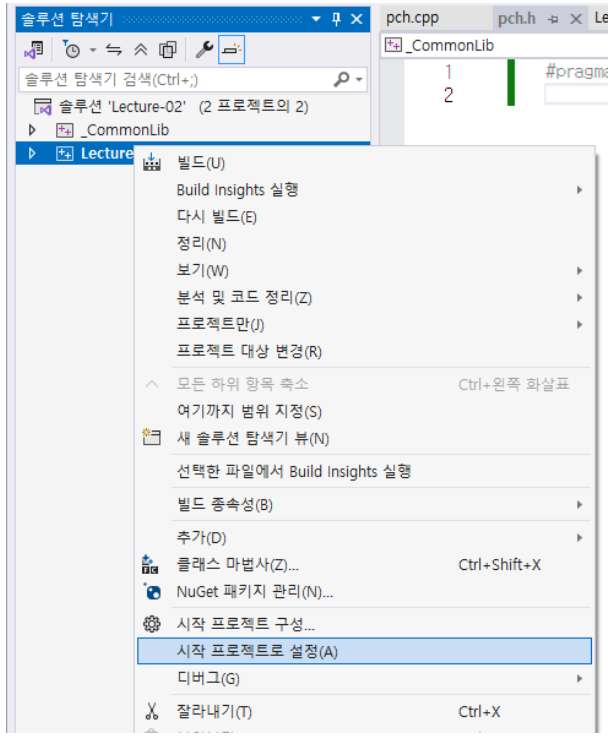


정적 라이브러리가 시작프로젝트로 지정되었기 때문입니다

실행 파일을 빌드하는 프로젝트를 시작 프로젝트에 지정했어야 하는데 말입니다.

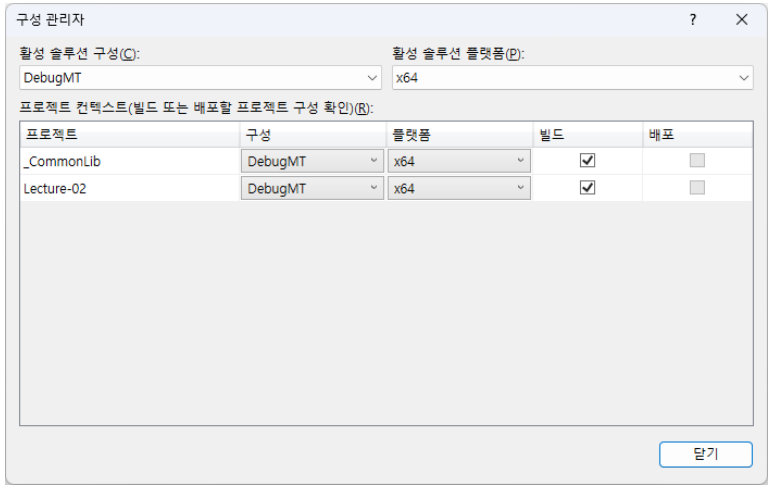
시작 프로젝트는 굵은 글씨로 표기됩니다.

언젠가 겪을 수 있는 문제이니 꼭 알아두길 바랍니다.

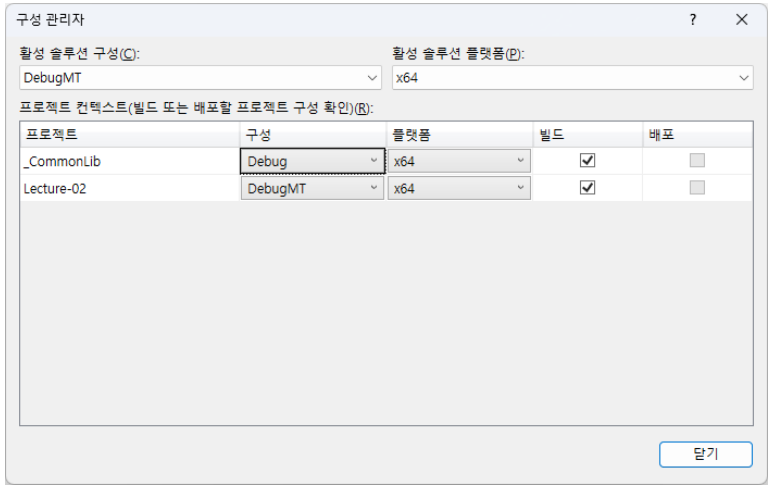


하나의 솔루션은 여러 개의 프로젝트로 이루어졌습니다. 그 중에는 정적 라이브러리처럼 실행 불가한 경우도 있지만 Exe처럼 실행 가능한 프로젝트가 두 개 이상 구성될 수도 있습니다.

[메뉴] -> [빌드] -> [구성 관리자]를 선택하면 띄울 수 있습니다.



(O) 옳은 구성



(X) 틀린구성

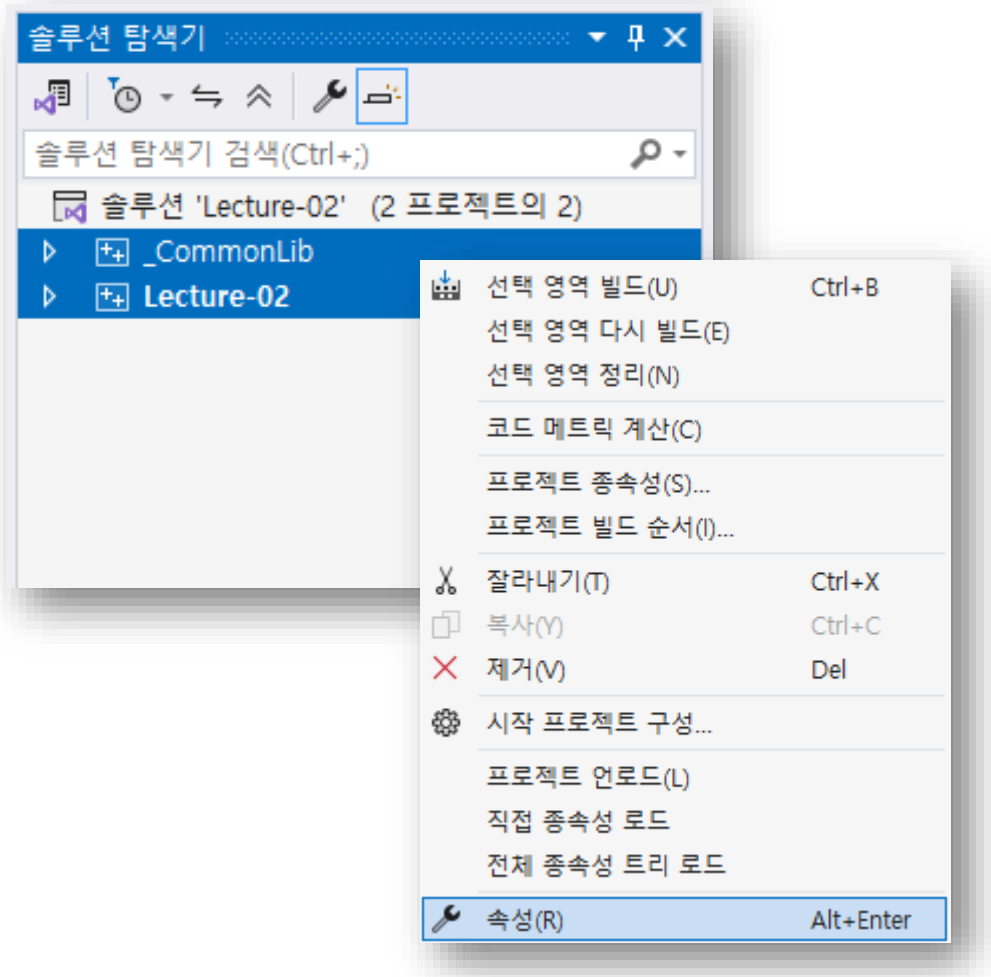
상단의 [활성 솔루션 구성]과 [활성 솔루션 플랫폼]이 있고, 그 아래로 프로젝트마다 구성과 플랫폼을 다시 선택할 수 있는 콤보 박스가 있습니다. **구성과 플랫폼은 항상 통일되도록 솔루션을 구성해 두어야 합니다.**

모든 설정에 서로 다른 구성이 섞이기 않게 관리하세요. 빌드에 실패하게 될 겁니다.

여러 프로젝트의 속성을 한꺼번에 수정

둘 이상의 프로젝트가 존재하면, 일일이 속성창을 열어 수정해야 하는 번거로움이 있습니다.

이 때, 여러 프로젝트를 동시에 선택해서 속성을 변경하는 것도 가능합니다. 아래와 같이 둘 이상의 프로젝트를 모두 선택한 후 속성 메뉴를 눌러보시면, 선택된 모든 프로젝트의 설정을 바꿀 수 있는 창이 나타납니다.



빌드 도구

컴파일러/링커 그들은 원수인가, 친구인가?

아래와 같이 네 개의 파일로 구성된 코드는 컴파일이 될까요?

<ClassA.h>

```
#pragma once
#include “ClassB.h”

class ClassA
{
        void Func(ClassB* pB);
};
```

<ClassB.h>

```
#pragma once
#include “ClassA.h”

class ClassB
{
        void Func(ClassA* pA);
};
```

<ClassA.cpp>

```
#include “ClassA.h”
void ClassA::Func(ClassB* pB)
{
}
```

<ClassB.cpp>

```
#include “ClassB.h”
void ClassB::Func(ClassA* pA)
{
}
```

에러가 납니다. 왜 이런 메시지가 나올까요?

```
빌드 시작...
1>------ 빌드 시작: 프로젝트: VeryImportantProject, 구성: Debug x64 ------
1>ClassA.cpp
1>D:₩GIT₩VeryImportantProject₩Src₩VeryImportantProject₩ClassB.h(7,12): error C2061: 구문 오류: 식별자 'ClassA'
1>ClassB.cpp
1>D:₩GIT₩VeryImportantProject₩Src₩VeryImportantProject₩ClassA.h(7,12): error C2061: 구문 오류: 식별자 'ClassB'
1>코드를 생성하고 있습니다...
1>"VeryImportantProject.vcxproj" 프로젝트를 빌드했습니다. - 실패
========== 빌드: 0개 성공, 1개 실패, 1개 최신 상태, 0개 건너뜀 ==========
========== 빌드이(가) 10:38 PM에 시작되었고 00.313 초이(가) 소요됨 ==========
```

앞의 문제를 해결하는 방법은 세 가지 입니다.

1. 두 클래스를 하나로 합친다.
2. 클래스를 세 개로 나눈다.
3. 컴파일 순서를 조정한다.

컴파일러가 처리하는 최소 단위는 소스파일(*.c / *.cpp)입니다.

#include 구문은 가리키는 파일을 말그대로 소스 파일에 포함하는 것입니다.

그렇다면 컴파일러가 이해할 수 있도록 #include를 정리하면 되는 것입니다.

ClassB.h	대략적인 ClassA 선언
	ClassB 선언
ClassA.h	ClassA 선언
ClassA.cpp	ClassA 구현

ClassB.h	대략적인 ClassA 선언
	ClassB 선언
ClassB.cpp	ClassB 구현

앞선 코드의 문제는 이렇게 해결할 수 있게 됩니다.

<ClassA.h>

```
#pragma once
#include “ClassB.h”

class ClassA
{
        void Func(ClassB* pB);
};
```

<ClassB.h>

```
#pragma once

class ClassA;
class ClassB
{
        void Func(ClassA* pA);
};
```

<ClassA.cpp>

```
#include “ClassA.h”
void ClassA::Func(ClassB* pB)
{
}
```

<ClassB.cpp>

```
#include “ClassB.h”
#include “ClassA.h”
void ClassB::Func(ClassA* pA)
{
}
```

유독 C++에서 함수와 정의가 분리된 것은

이렇게 헤더와 cpp의 우선순위를 조정하기 것입니다.

또한, *정의는 함수만 할 수 있는 게 아니라 클래스나 변수도 가능하다는 것을 잊지 마세요.*

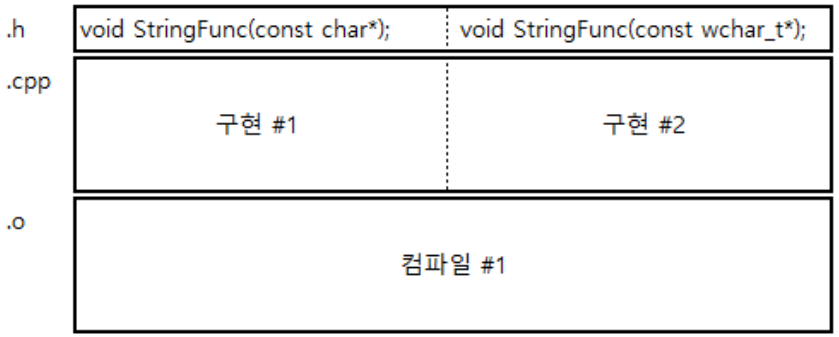
모든 문자열 함수는 char형과 wchar_t 형으로 각각 만들어야 한다.

따라서 StringFunc라는 함수를 하나 만들려고 하면,

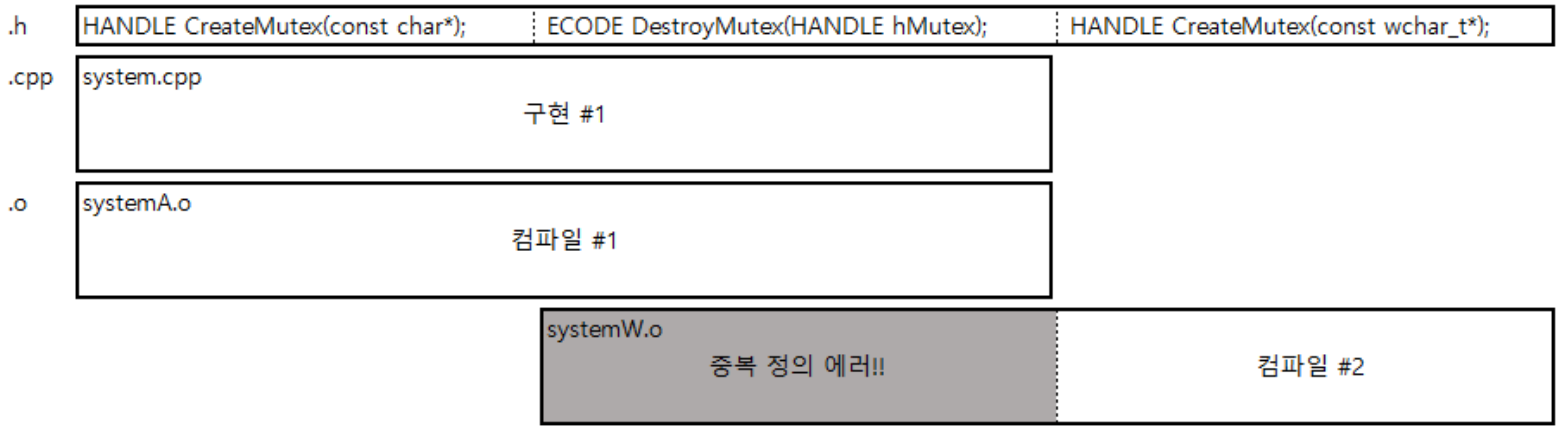
비슷한 코드를 두 번씩 짜야하는 수고로움이 발생한다.

이 것을 컴파일러와 링커에게 일을 분배해본다.

어떻게 가능할까?

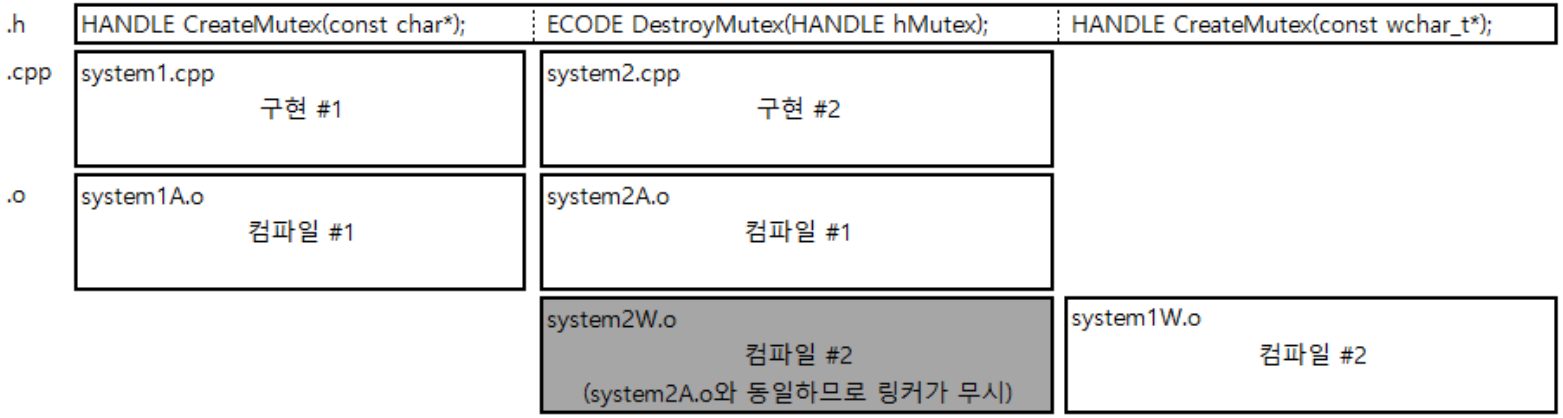


아래와 같이 두 함수를 하나의 cpp 파일의 정의해 듀얼 컴파일하여 링크하면 오류가 발생한다.



이것을 두 파일에 나누어서 작성하면 문제가 해결된다.

이렇게 에러가 나지 않도록 파일을 나누는 기준은 무엇일까?



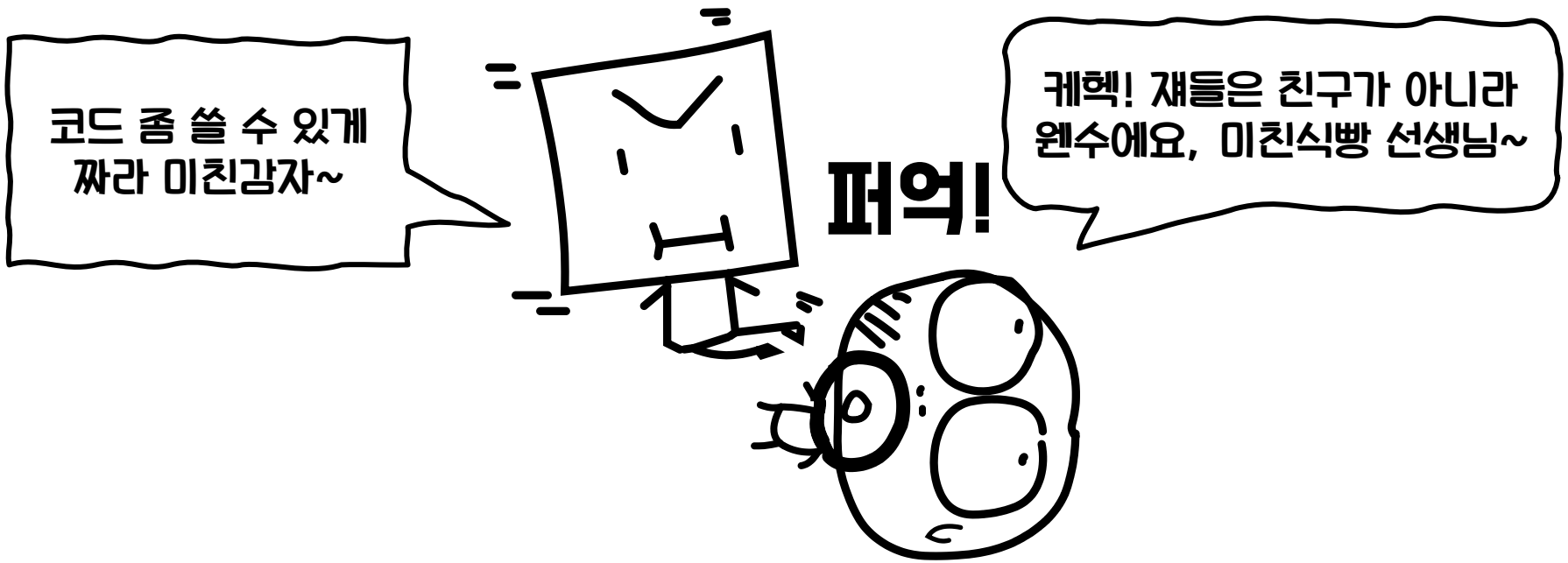
이렇듯이 *컴파일러*와 *링커*는 우리의 친구입니다.

이제 좀 친해진 것 같나요?

이제 소스코드 관리기술에 대해 알아볼 차례입니다.

여러분은 지금까지 얼마나 긴 코드를 작성해 보셨나요?

실무에서는 소스 코드가 어떻게 관리되는지 살짝 보여드릴게요.



코딩하기

기본적인 코딩과 빌드 맛보기

코딩은 위 세 가지를 모두 다룰 수 있어야 가능한 겁니다.

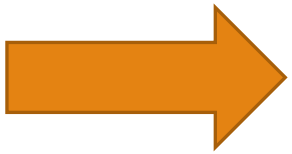
```
#include "pch.h"

int main()
{
    HANDLE hFile = CreateFile(TEXT("d:/report.txt"), GENERIC_READ_, OPEN_EXISTING_, 0);
    if (NULL == hFile)
        return -1;

    char szBuffer[10000+1];
    DWORD dwReadSize = 0;
    ReadFile(hFile, szBuffer, 10000, &dwReadSize);

    szBuffer[dwReadSize] = 0;
    printf("%s₩n", szBuffer);

    CloseFile(hFile);
    return 0;
}
```

C++은 네 가지 언어의 집합체입니다.
적어도 C언어의 4배는 더 노력해야 한다는 거죠.
관련 서적은 많으니 찾아서 공부하세요~

시스템 API의 네이밍 규칙을 익히고 MSDN에서 상세한 사용법 확인하며 배워 나갑니다.

결국 많은 API를 이해하고 쓸 줄 아는 사람이 고급 개발자입니다.

<table>
<tr><td colspan="2">윈도우</td><td>윈도우 전용 API</td></tr>
<tr><td rowspan="2">유닉스</td><td>리눅스</td><td>리눅스 전용 API</td></tr>
<tr><td>맥OSX</td><td>맥 전용 API
(리눅스와 유사함)</td></tr>
</table>



cppcore 전용 API
(윈도우와 유사함)

명령	커널객체	유니코드 사용여부
Create/Destroy/Delete Open/Close Enter/Leave Suspend/Resume/Terminate Initialize/Finalize Find/Read/Write Get/Set/Flush/Enum/Wait/Exit	File/Directory/Process/Pipe FileName/ModuleFileName CriticalSection/Mutex/Semaphore/Event LocalTime/SystemTime SystemDirectory/TempPath	A/W (인자에 문자열이 있는 경우에 한함)
Ex) CreateFileA, GetLocalTime, GetCurrentDirectory, ClosePipe…		

오픈소스(패키지)를 가져다 쓰는 기술도 코딩 기술에 포함됩니다.

1. 얼마나 많은 오픈소스를 알고 있고

2. 내 프로젝트에 접목할 수 있으며

3. 사용할 줄 아는가

위의 3가지가 패키지 사용 기술이라고 생각하면 됩니다.

우리는 cppcore를 사용해볼 예정입니다. 다음 순서를 따라해봅시다.

1. 다음 저장소를 git clone명령으로 내려 받음

https://github.com/profrog-jeon/cppcore.git

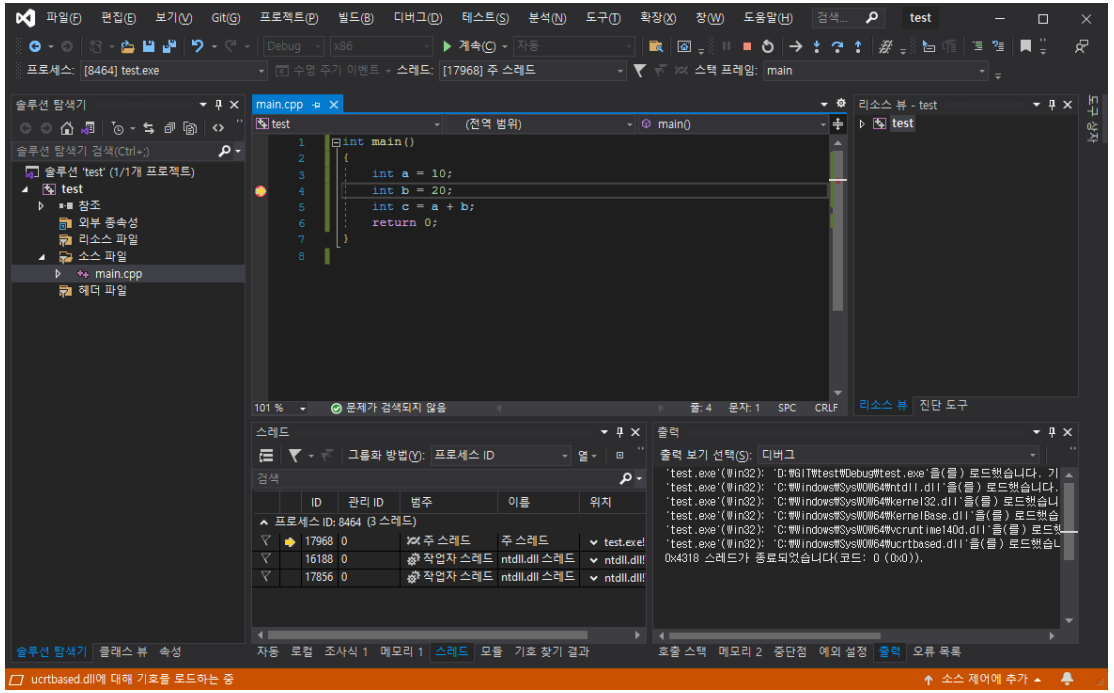
2. 빌드하여 윈도우용 cppcore.lib 및 리눅스용 libcppcore.a을 생성

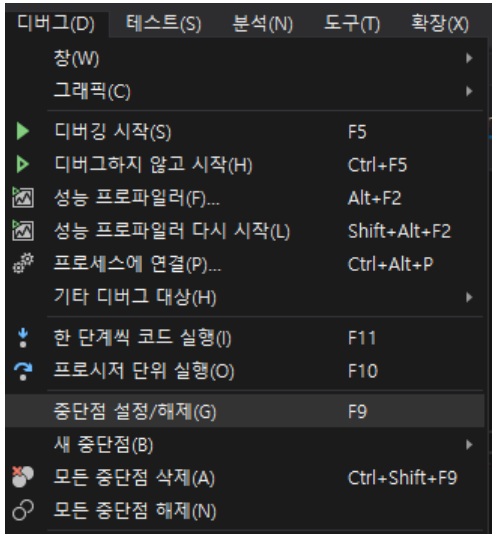
3. 내 프로젝트에 가져와서 빌드에 성공시킴

디버깅

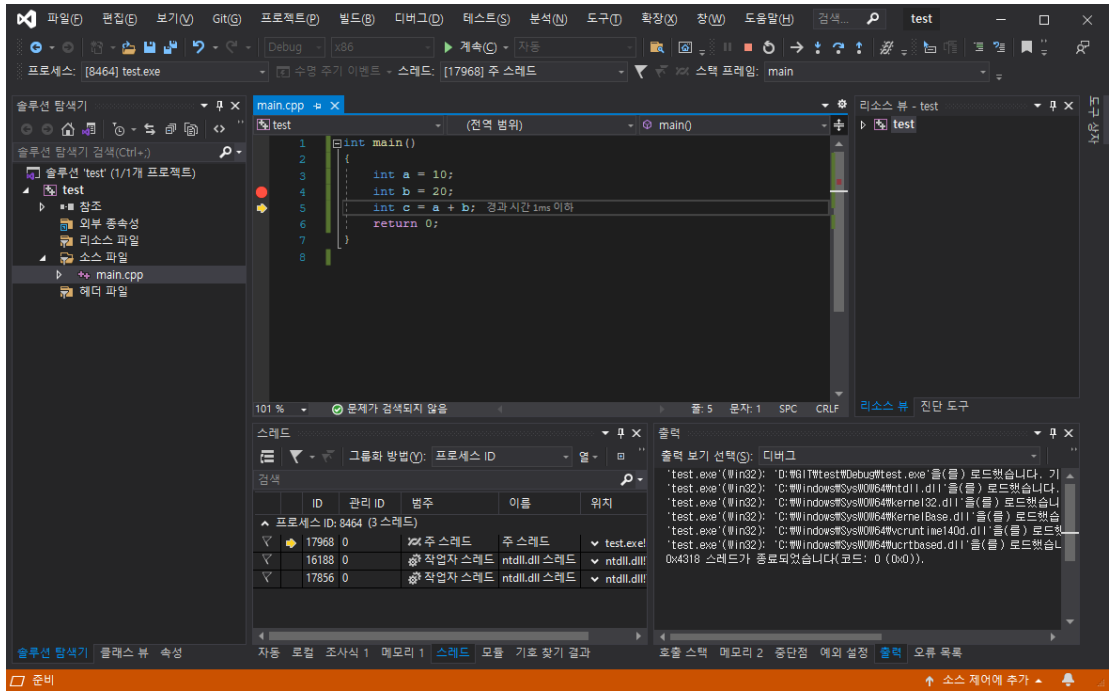
중단점(브레이크 포인트), 한 줄 실행(싱글 스텝),
변수 보기, 호출 스택(콜스택) 보기, 조사식,
메모리 보기, 스레드 이동,

특정 시점에 실행을 멈춰 두고 변수나 메모리 상태를 확인하고 싶다면,
명령 라인에 커서를 두고 **F9**를 눌러봅시다.
그럼 코드 왼쪽편에 빨간색 동그라미가 생성될 겁니다.
그 이후 F5 버튼으로 실행하면 빨간색 동그라미가 새겨진 라인에서 실행이 멈춥니다.
이 시점에 메모리나 변수, 콜스택 상태 등등을 확인할 수 있습니다.
F10을 누르면 한 줄씩 실행하고, F11 누르면 함수 안으로 진입,
F5를 누르면 다음 중단점까지 실행합니다.

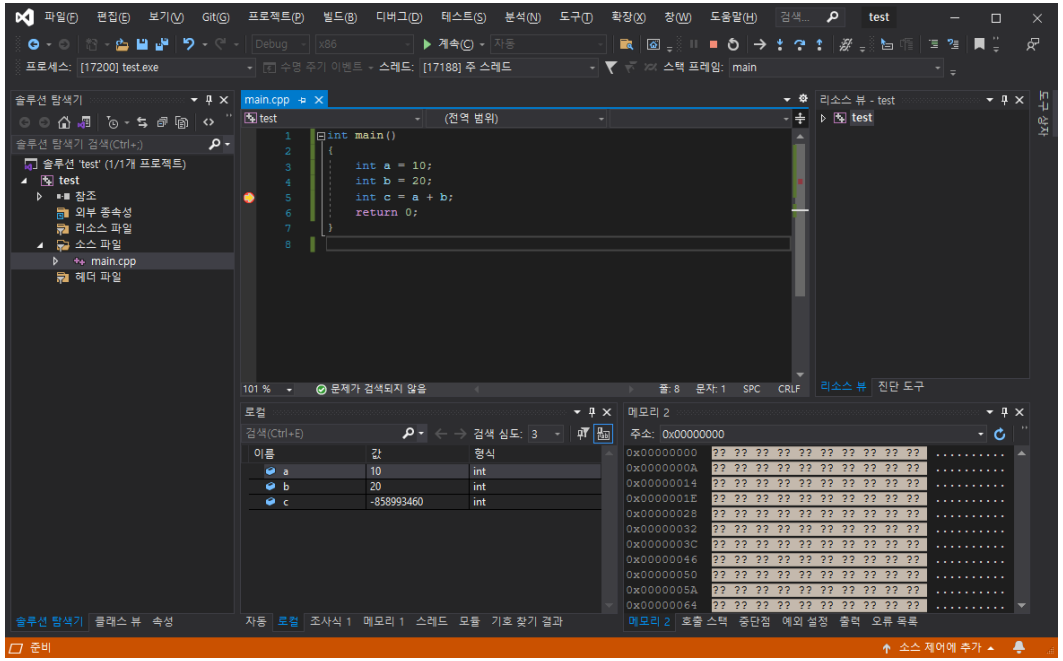


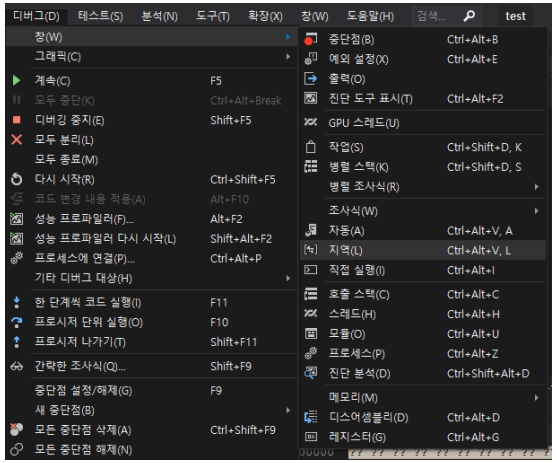


멈춰선 지점에서 한 명령만 실행하게 만들고 싶으면,
F10을 눌러 “프로시저 단위 실행”으로 한 줄 넘어갈 수 있어요.
만일 함수 구현부까지 따라가 보고 싶으면,
F11인 “한 단계씩 코드 실행”도 있습니다.

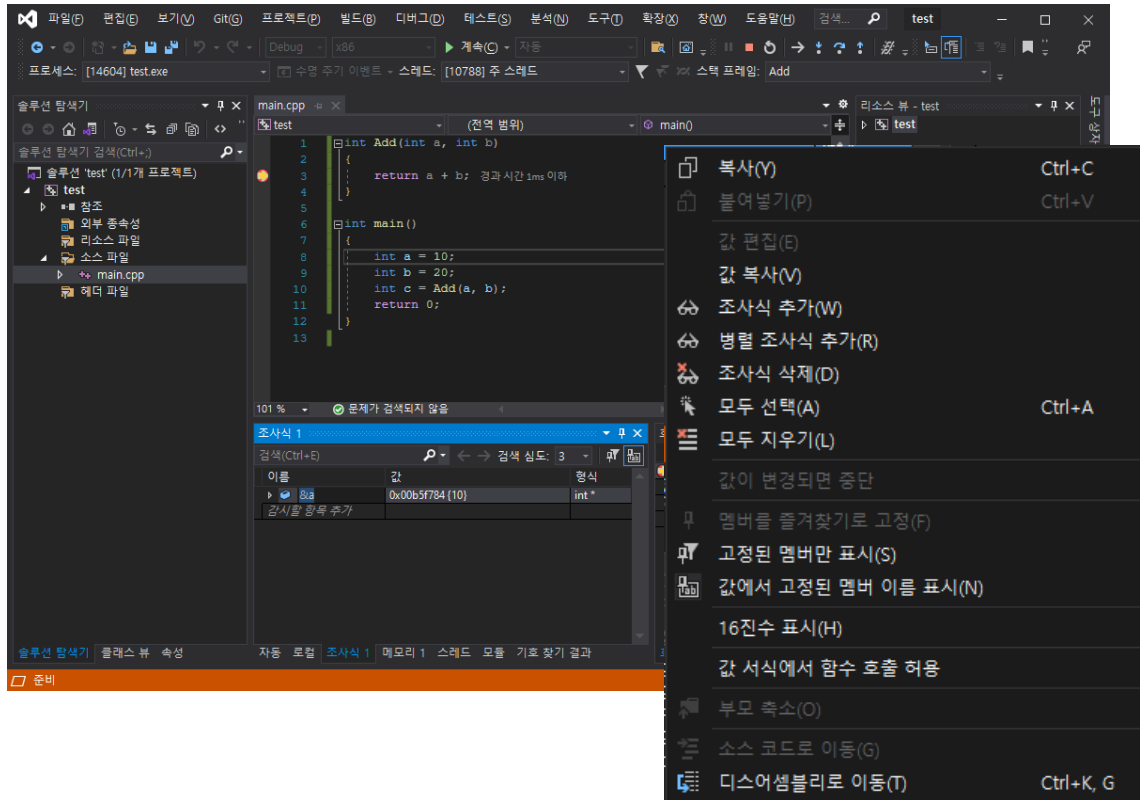


지역 변수들의 값을 확인하고 싶다면 멈춰선 상태에서,
[메뉴 -> 디버그 -> 창 -> 지역]을 선택해 로컬변수 창을 띄웁니다.
아래쪽 탭에 나타나요.





그 외에 내가 조사하고 싶은 변수나 수식은
“조사식”창에 직접 입력해서 값을 확인할 수 있어요.

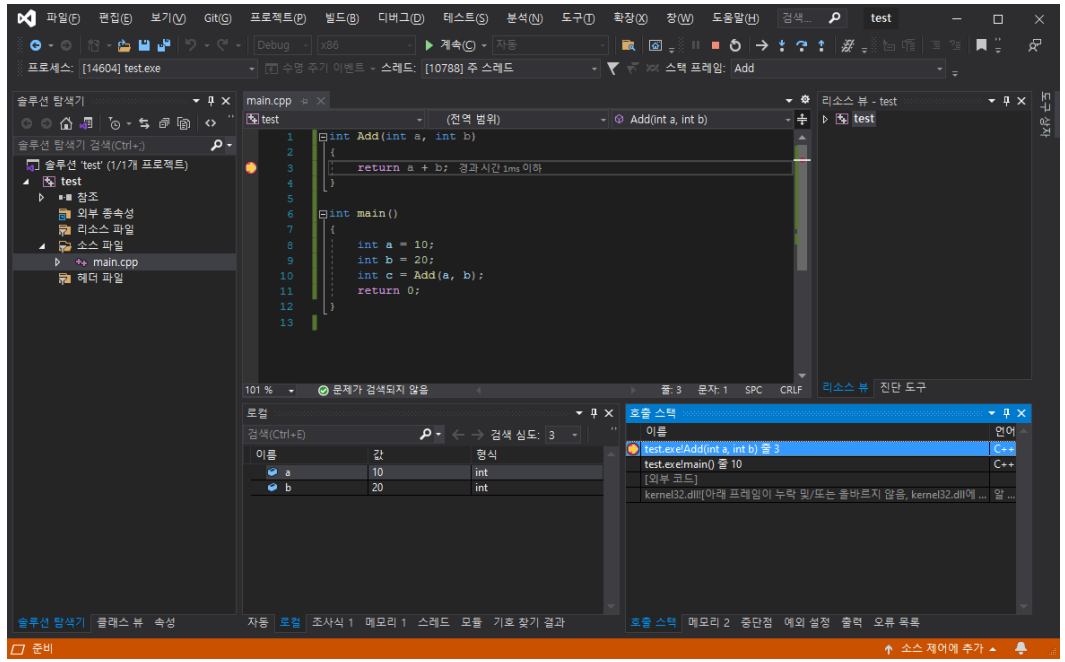


조사식에는 직접 변수나 포인터, 함수 등 C++ 문법을 사용할 수 있음

또한 값에 마우스 커서를 두고 오른쪽 클릭하면 메뉴 항목이 나타남

이 예제에서는 값복사를 하면 로컬변수 a의 메모리 주소를 얻어올 수 있음

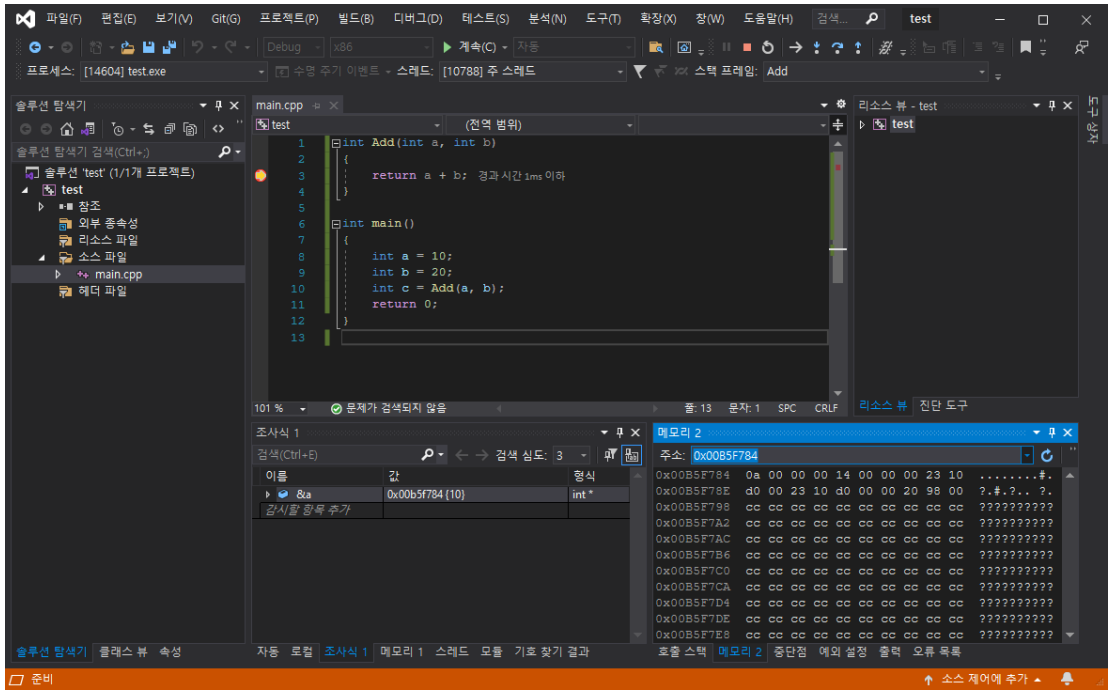
[호출 스택] 창은 지금 이 지점으로 들어올 때까지 어떤 함수가 개입되었는지를 보여줍니다.
알다시피 모든 함수는 main에서 시작하므로 가장 아래에 있지만 그 밑에는 OS가 main함수를 부르기 위한 시스템 함수들도 존재하죠.



임의로 함수를 만들어 호출한 후 호출 스택 상태를 보면 아래에서부터 위로 함수가 불리어진 상황을 볼 수 있음

각 항목을 더블클릭하면 디버깅 정보가 있는 경우에 한해 해당 시점의 변수 등을 조사 가능

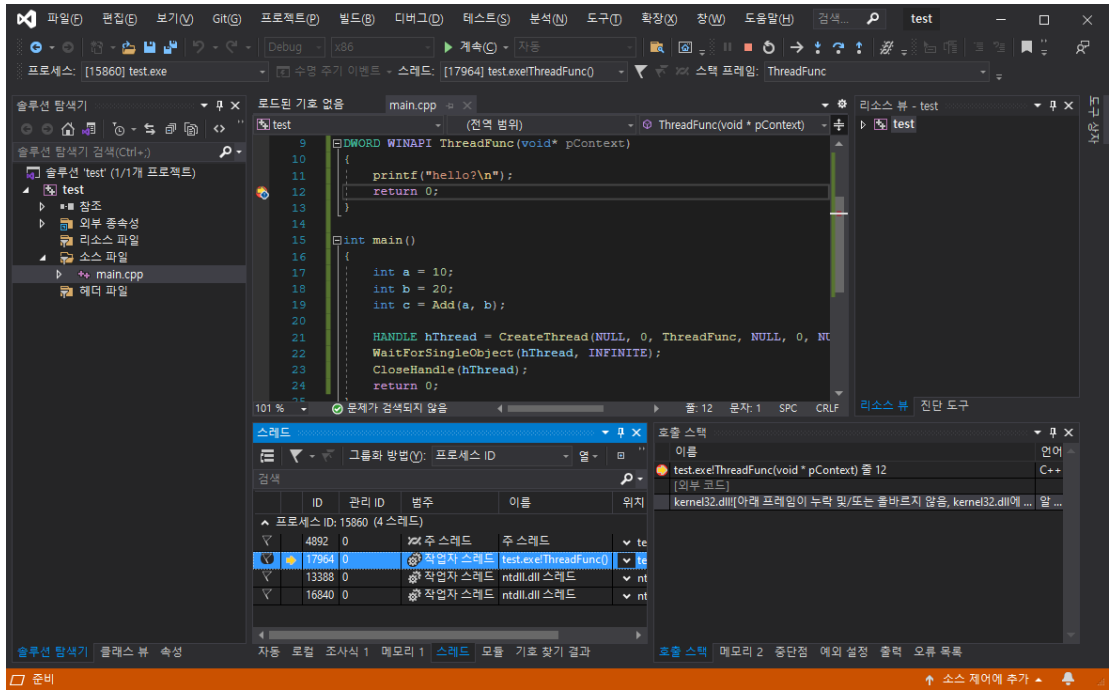
메모리 상태도 조회할 수 있어요.
보고싶은 메모리 주소를 16진수로 입력하면 해당 위치의 값을 헥스값과 아스키 형태로 나타내 줍니다.
값을 수정하고 싶다면 마우스로 클릭해 바꿀 수도 있어요.



앞에서 얻어온 로컬변수 a의 주소를 디버깅 메모리 창에 입력하면 해당 지점의 메모리 상태를 조사할 수 있음

4 바이트 int형에 LittleEndian 이므로 0a 00 00 00 이 기록되어 있음을 알 수 있음

마지막으로 현재 구동 중인 스레드들도 보여줍니다.
스레드를 선택하면 해당 스레드의 호출 스택이나 변수상태를 바로 보여줍니다. 멀티 스레드 프로그래밍시 요긴하게 쓰이겠죠.
특히 데드락을 해결하기 위해 이 기능은 필수적입니다.
멀티스레드 디버깅이 리눅스 디버거가 잘 못해주는 부분이에요.
MS에 감사해야죠.



임의로 스레드를 생성해서 중단점을 걸어보면 현재 생성된 스레드 상태를 조회 가능

주스레드는 main 함수를 호출한 것이고 그외에는 프로그램 실행중에 생성된 것들임

시스템에서 생성한 것들은 디버깅 정보가 없으므로 볼 수 없지만 코드로 생성한 스레드는 더블클릭하면 현재 컨텍스트를 불러와 볼 수 있음

고생 많으셨습니다!