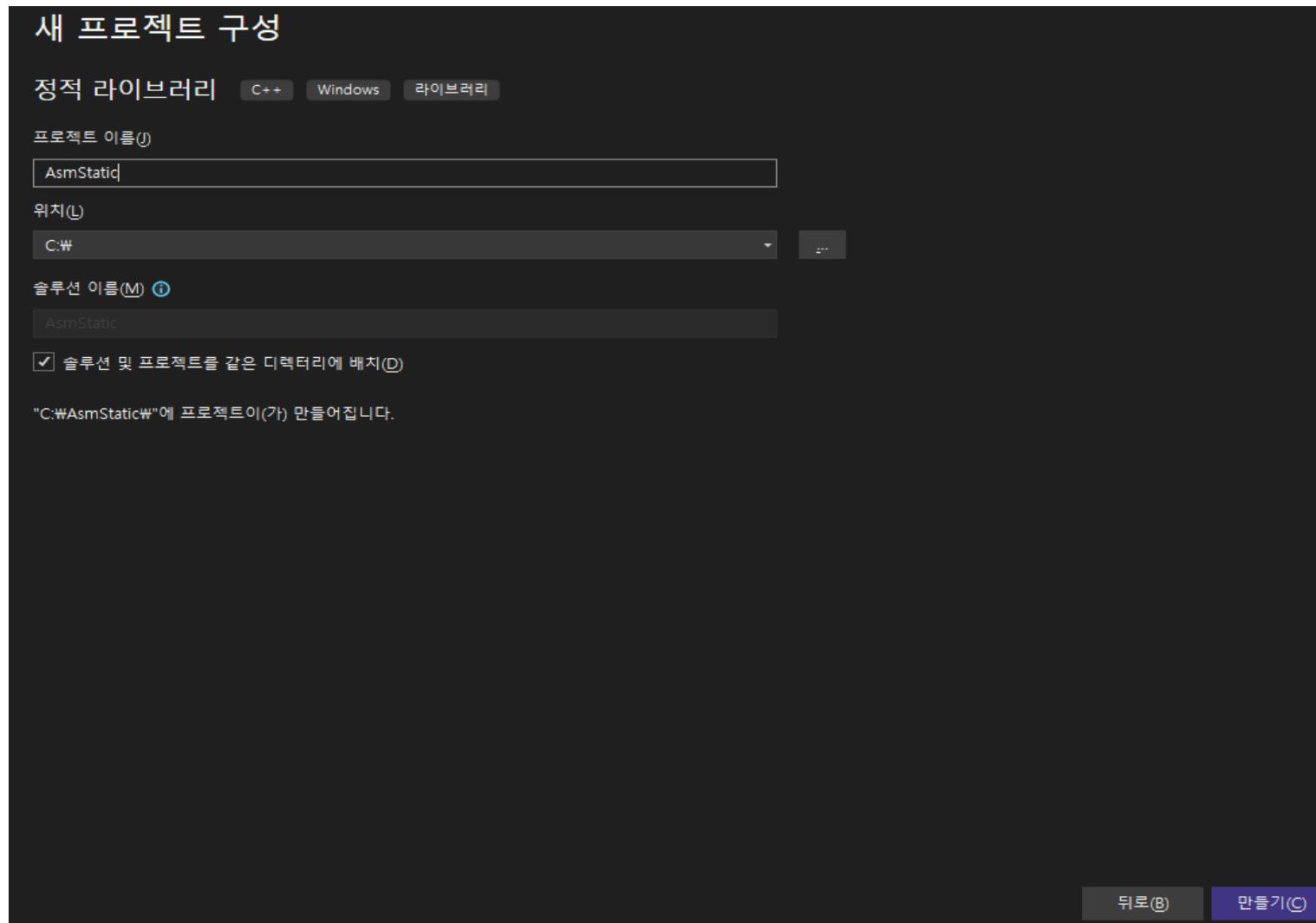


어셈블리(Asm + C -> 라이브러리 -> DLL)

먼저 정적 라이브러리를 만들어 줍니다. (C:\AsmStatic)



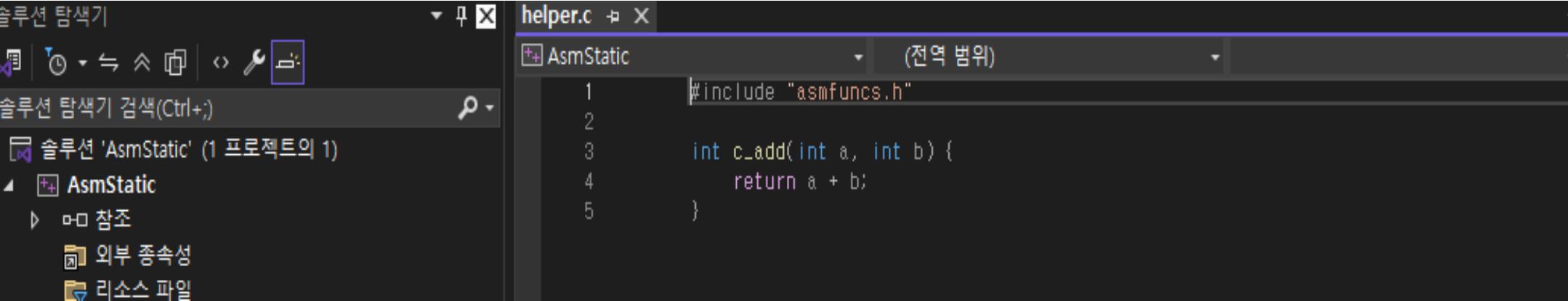
AsmStatic의 프로젝트에 .asm과 .c 그리고 .h를 추가해줍니다

add_impl.asm(Asm파일)

The screenshot shows the Microsoft Visual Studio IDE interface. On the left, the Solution Explorer displays the project structure for 'AsmStatic'. It includes files like AsmStatic.cpp, helper.c, pch.cpp, asmfuns.h, framework.h, and pch.h under various categories like References, External Dependencies, and Headers. The 'Sources' category contains the 'add_impl.asm' file, which is currently selected and open in the center editor pane. The right editor pane shows the assembly code for 'add_impl.asm'.

```
1 .386 ; 32비트 모드
2 .model flat, c ; C 호출 규약 사용
3 option casemap:none ; 대소문자 구분
4
5 PUBLIC add_impl ; 이 함수 export
6
7 EXTERN c_add:PROC ; C 함수 참조
8
9 .code ; 코드 섹션 시작
10
11 add_impl PROC uses eax ecx edx a:DWORD, b:DWORD
12 ;
13 ; C 함수 c_add(a, b)를 호출하는 ASM 함수
14 ;
15 push b
16 push a
17 call c_add
18 add esp, 8 ; 인자 정리 (cdecl)
19 ret
20 add_impl ENDP
21
22 END
```

helper.c(C파일)

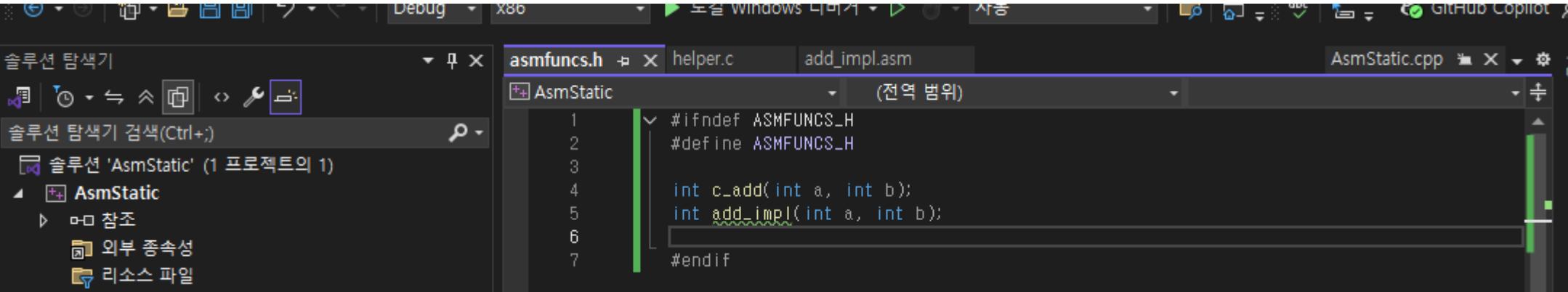


The screenshot shows the Visual Studio IDE interface with the 'helper.c' file open in the main editor window. The code contains a single function definition:

```
#include "asmfuncs.h"
int c_add(int a, int b) {
    return a + b;
}
```

The solution explorer on the left shows the project structure with 'AsmStatic' selected.

asmfuncs.h(헤더파일)



The screenshot shows the Visual Studio IDE interface with the 'asmfuncs.h' file open in the main editor window. The code defines a macro and two functions:

```
#ifndef ASMFUNCS_H
#define ASMFUNCS_H

int c_add(int a, int b);
int add_impl(int a, int b);

#endif
```

The solution explorer on the left shows the project structure with 'AsmStatic' selected. Other files like 'helper.c' and 'add_impl.asm' are also visible in the tabs.

.asm을 실행시키기 위해서 체크박스에 masm(target, props)를 체크 해줍니다

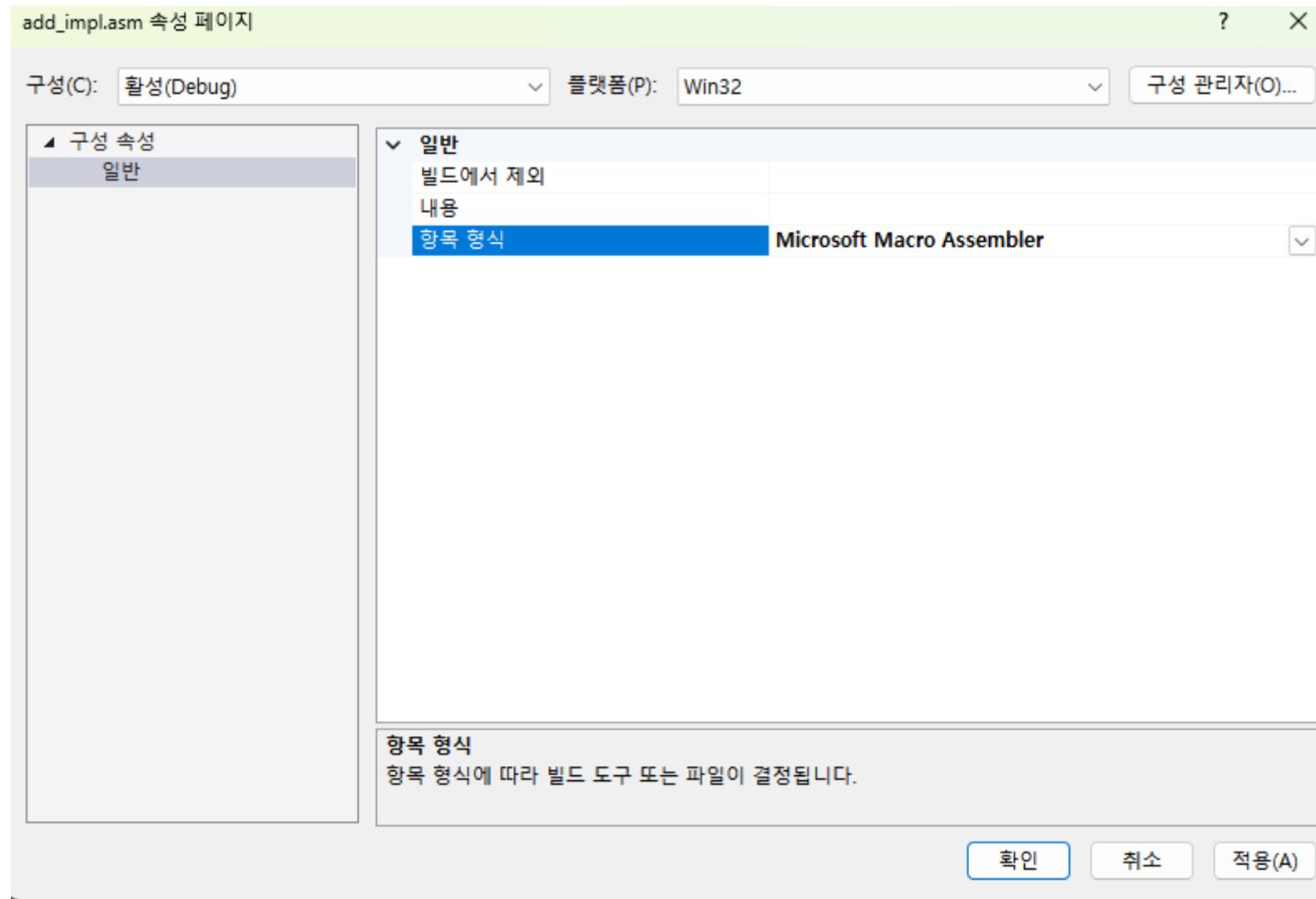
Visual C++ Build Customization Files

사용 가능한 빌드 사용자 지정 파일:

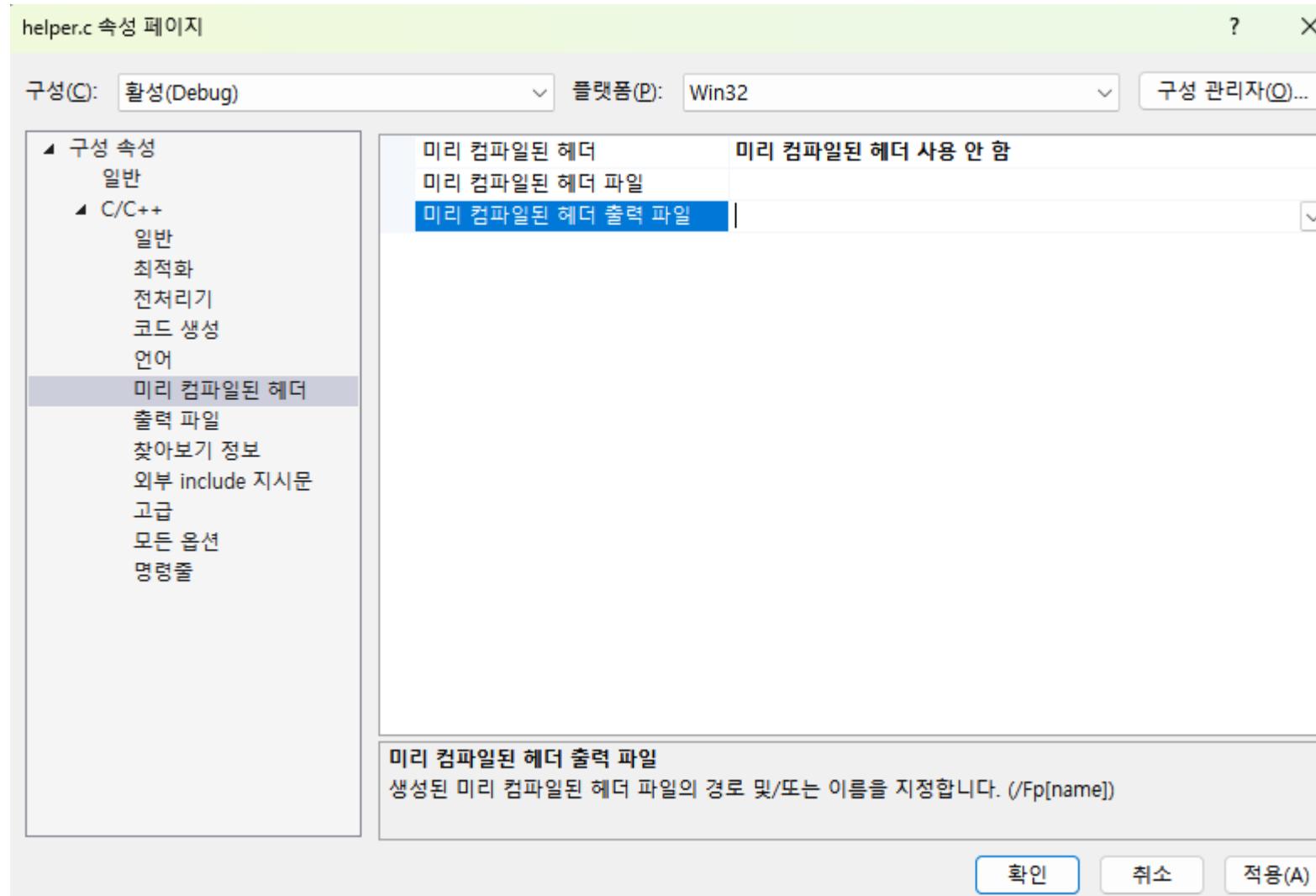
Name	Path
<input type="checkbox"/> ImageContentTask(.targets, .props)	\$(VCTargetsPath)\BuildCustomizations\ImageContentTask.targets
<input type="checkbox"/> lc(.targets, .props)	\$(VCTargetsPath)\BuildCustomizations\lc.targets
<input checked="" type="checkbox"/> marmasm(.targets, .props)	\$(VCTargetsPath)\BuildCustomizations\marmasm.targets
<input checked="" type="checkbox"/> masm(.targets, .props)	\$(VCTargetsPath)\BuildCustomizations\masm.targets
<input type="checkbox"/> MeshContentTask(.targets, .props)	\$(VCTargetsPath)\BuildCustomizations\MeshContentTask.targets
<input type="checkbox"/> ShaderGraphContentTask(.targets, .props)	\$(VCTargetsPath)\BuildCustomizations\ShaderGraphContentTask.targets

기존 파일 찾기... 목록 새로 고침 확인 취소

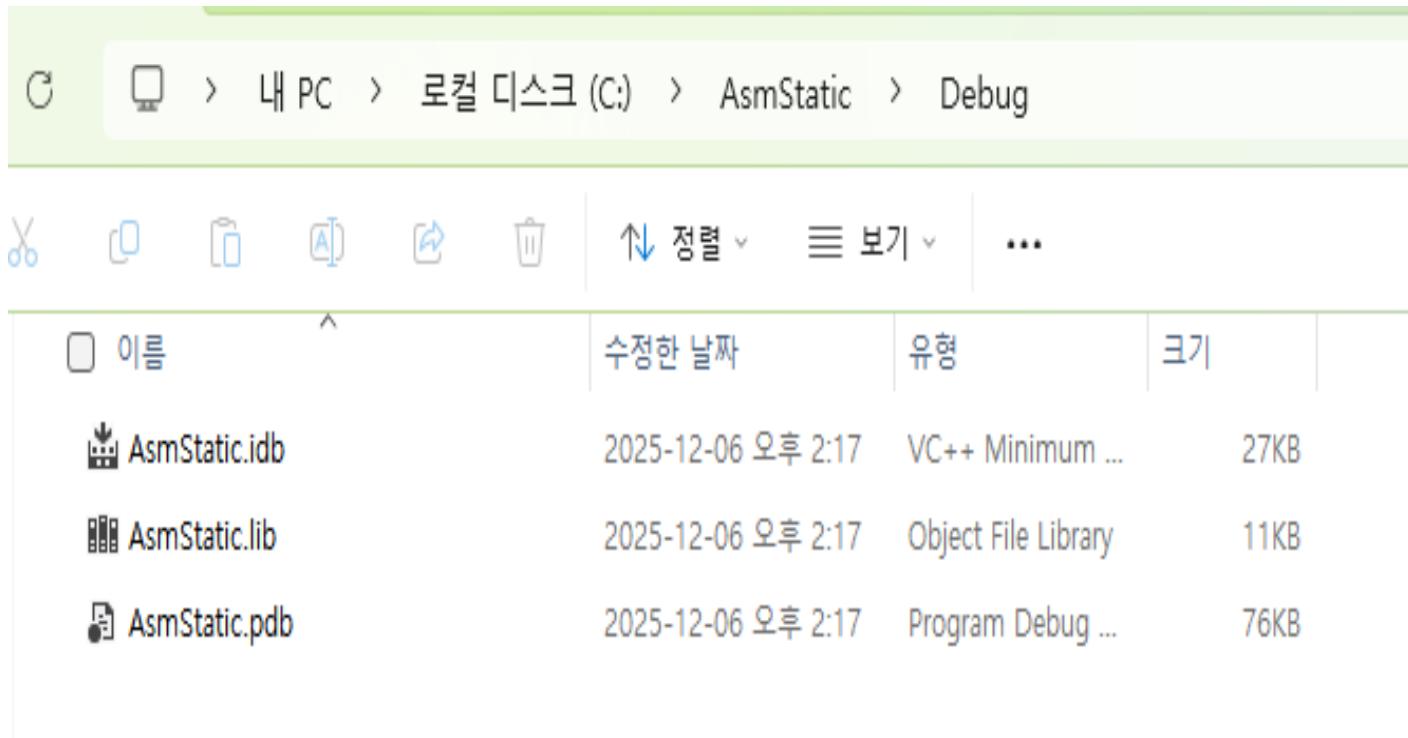
체크 후 add_impl.asm 우클릭 -> 속성 을 하여 활성화된 Microsoft Assembler를 선택



빌드를 실행 시키기 위해 미리 컴파일된 헤더를 “미리 컴파일된 헤더 사용 안함” 선택



솔루션 빌드 후 AsmStatic 폴더 안의 Debug 폴더에 .lib 파일(AsmStatic.lib) 생성



.lib파일 생성이 끝났으니 동적 라이브러리를 만들어 줍니다. (C\MyDll)

새 프로젝트 구성

DLL(동적 연결 라이브러리) C++ Windows 라이브러리

프로젝트 이름()

MyDll

위치(L)

C:\

솔루션 이름(M) ①

MyDll

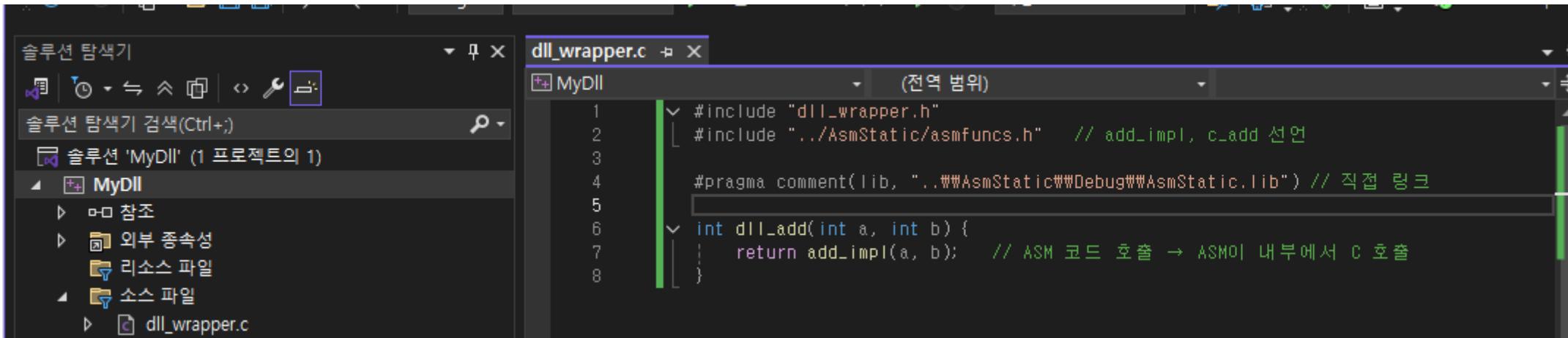
솔루션 및 프로젝트를 같은 디렉터리에 배치(D)

"C:\MyDll\"에 프로젝트이(가) 만들어집니다.

뒤로(B)

만들기(C)

dll_wrapper.c(C파일)



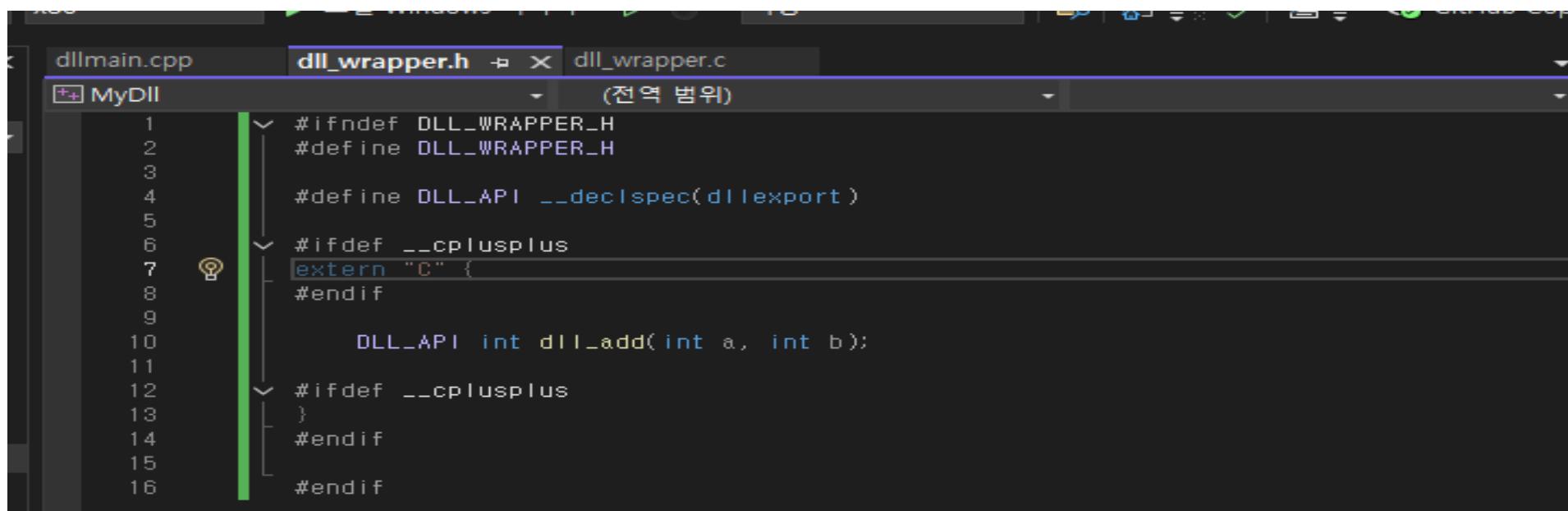
The screenshot shows the Visual Studio IDE interface with the 'MyDll' project selected in the Solution Explorer. The 'dll_wrapper.c' file is open in the main code editor window. The code contains the following content:

```
#include "dll_wrapper.h"
#include "../AsmStatic/asmfuncs.h" // add_impl, c_add 선언

#pragma comment(lib, ".\AsmStatic\Debug\AsmStatic.lib") // 직접 링크

int DLL_add(int a, int b) {
    return add_impl(a, b); // ASM 코드 호출 → ASMO 내부에서 C 호출
}
```

dll_wrapper.h(헤더파일)



The screenshot shows the Visual Studio IDE interface with the 'MyDll' project selected in the Solution Explorer. The 'dll_wrapper.h' header file is open in the main code editor window. The code contains the following content:

```
#ifndef DLL_WRAPPER_H
#define DLL_WRAPPER_H

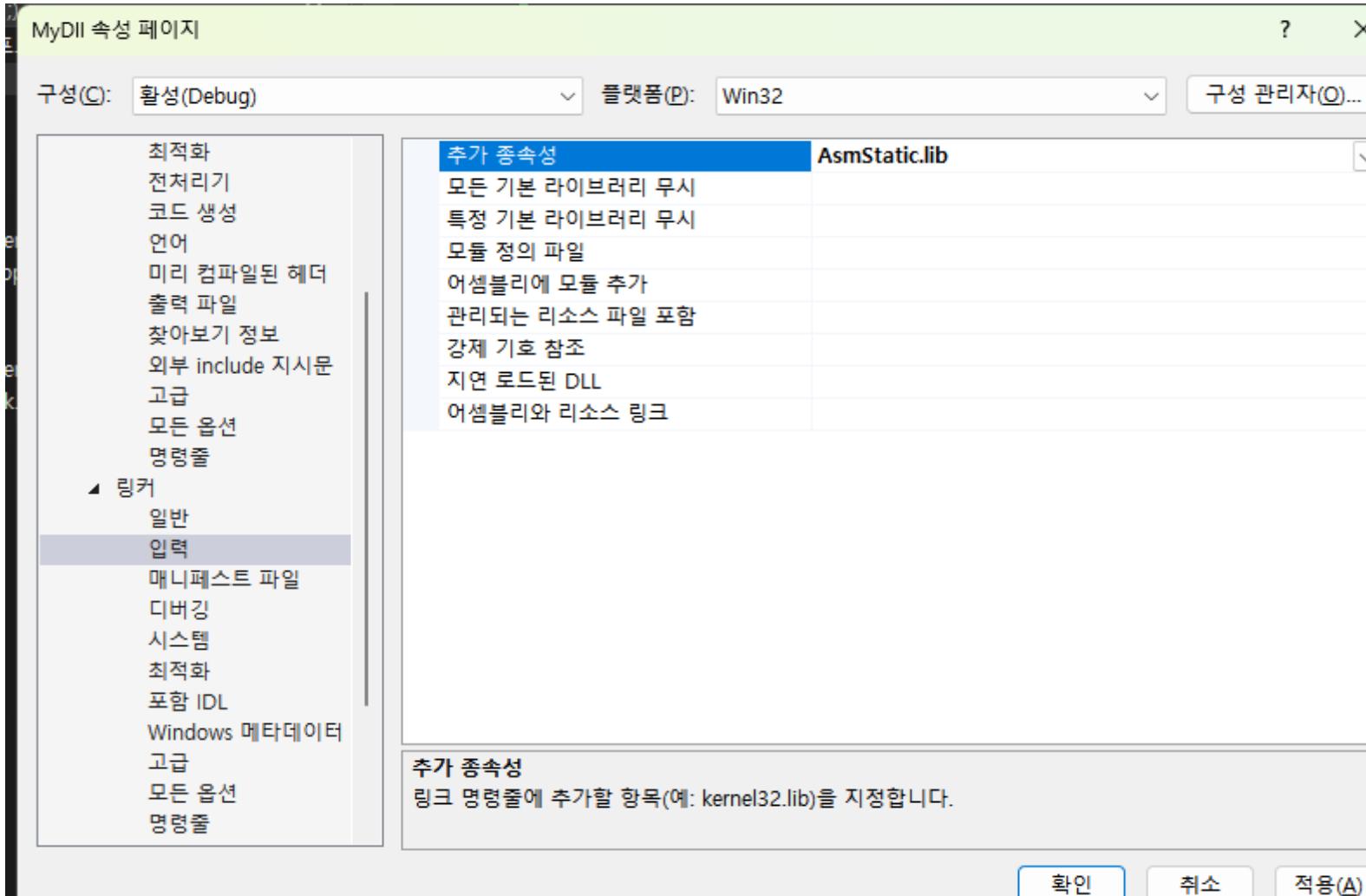
#define DLL_API __declspec(dllexport)

#ifndef __cplusplus
extern "C" {
#endif

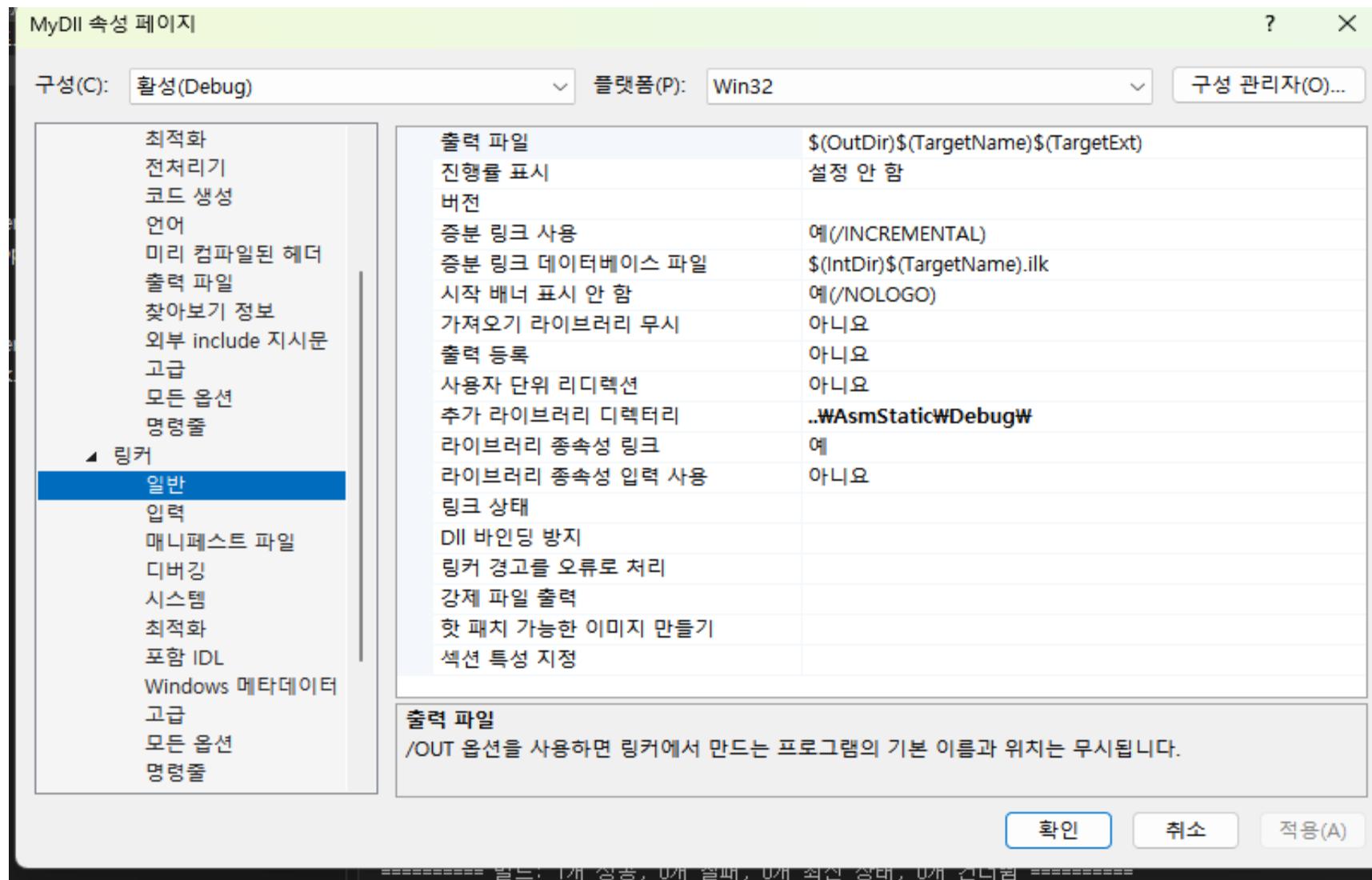
    DLL_API int DLL_add(int a, int b);

#ifndef __cplusplus
}
#endif
#endif
```

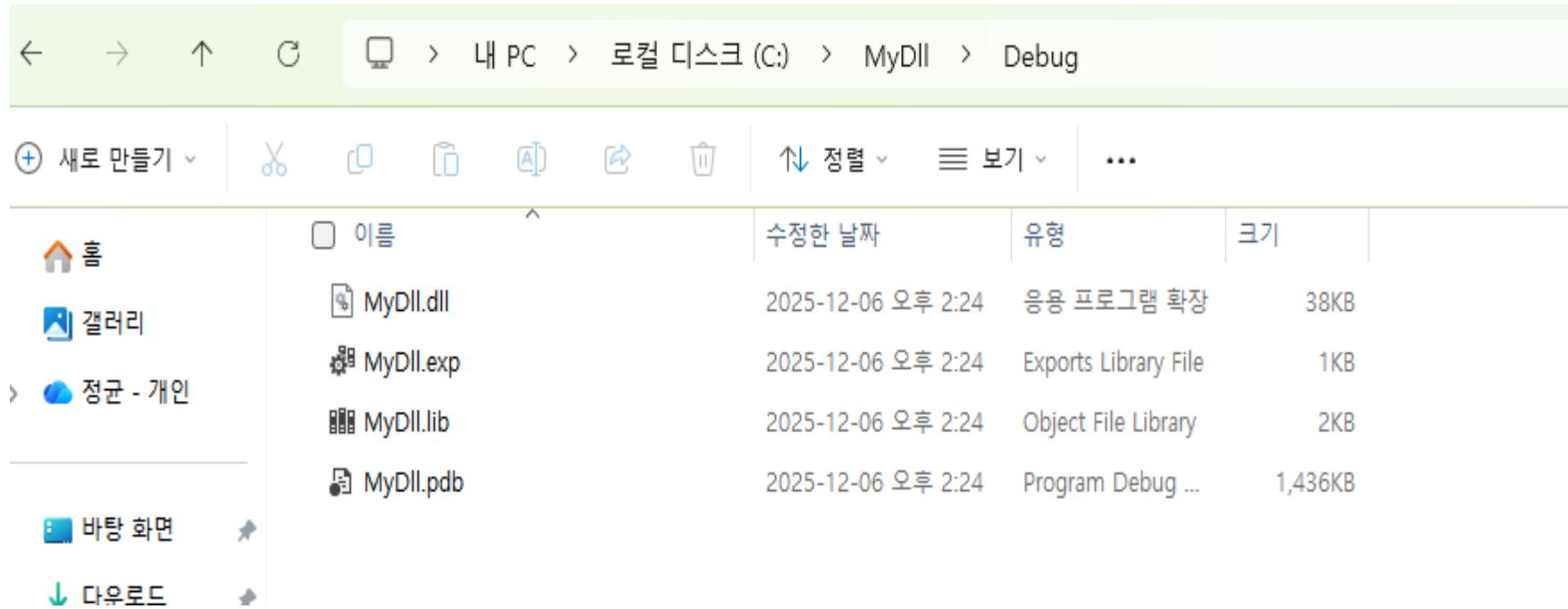
아까 만들어 둔 AsmStatic.lib을 MyDll 프로젝트에 링킹 해줍니다



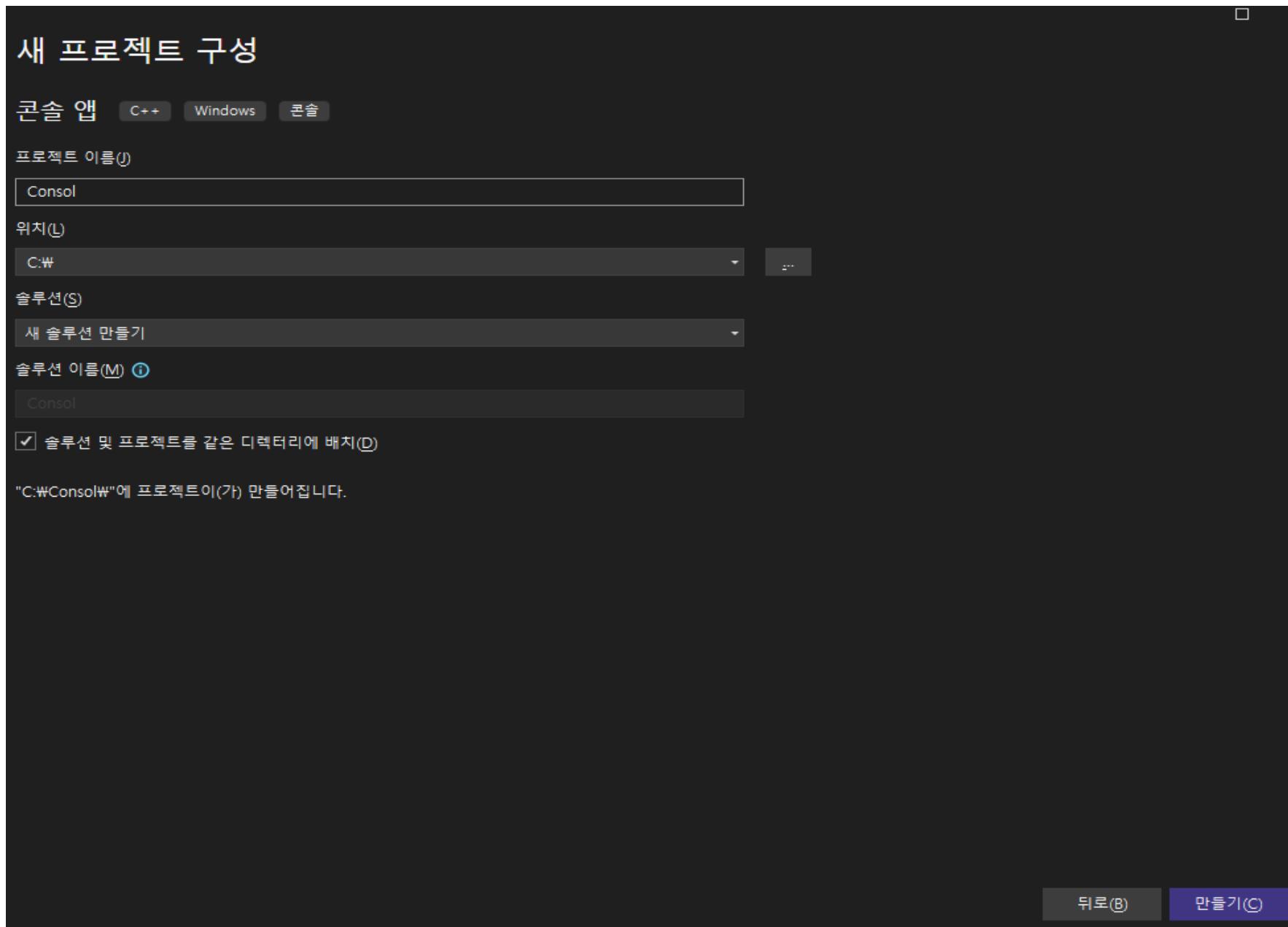
솔루션 빌드를 할 때 .lib의 파일 위치를 알아야 솔루션 빌드가 성공합니다. AsmStatic.lib의 위치는 C\AsmStatic\Debug 폴더 안에 위치하여 “추가 라이브러리 딕터리”에 위치 추가



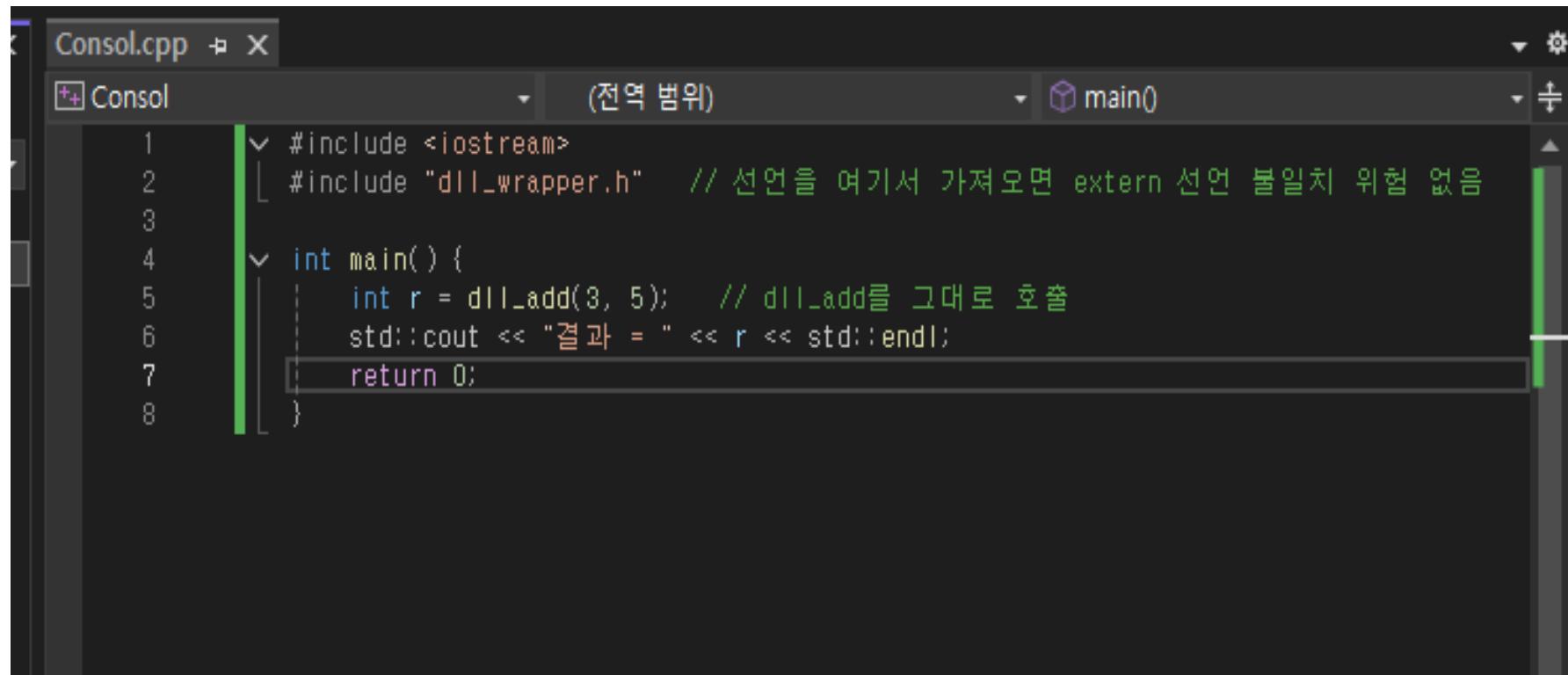
이전 설정 그대로 솔루션 빌드를 하면 C\MyDll\Debug 폴더에 .exp, .lib, .dll파일이 생성이 됩니다.



마지막으로 exe파일을 만들 프로젝트를 만듭니다. (C\WConsol)



실행할 코드를 Consol.cpp에 넣습니다.

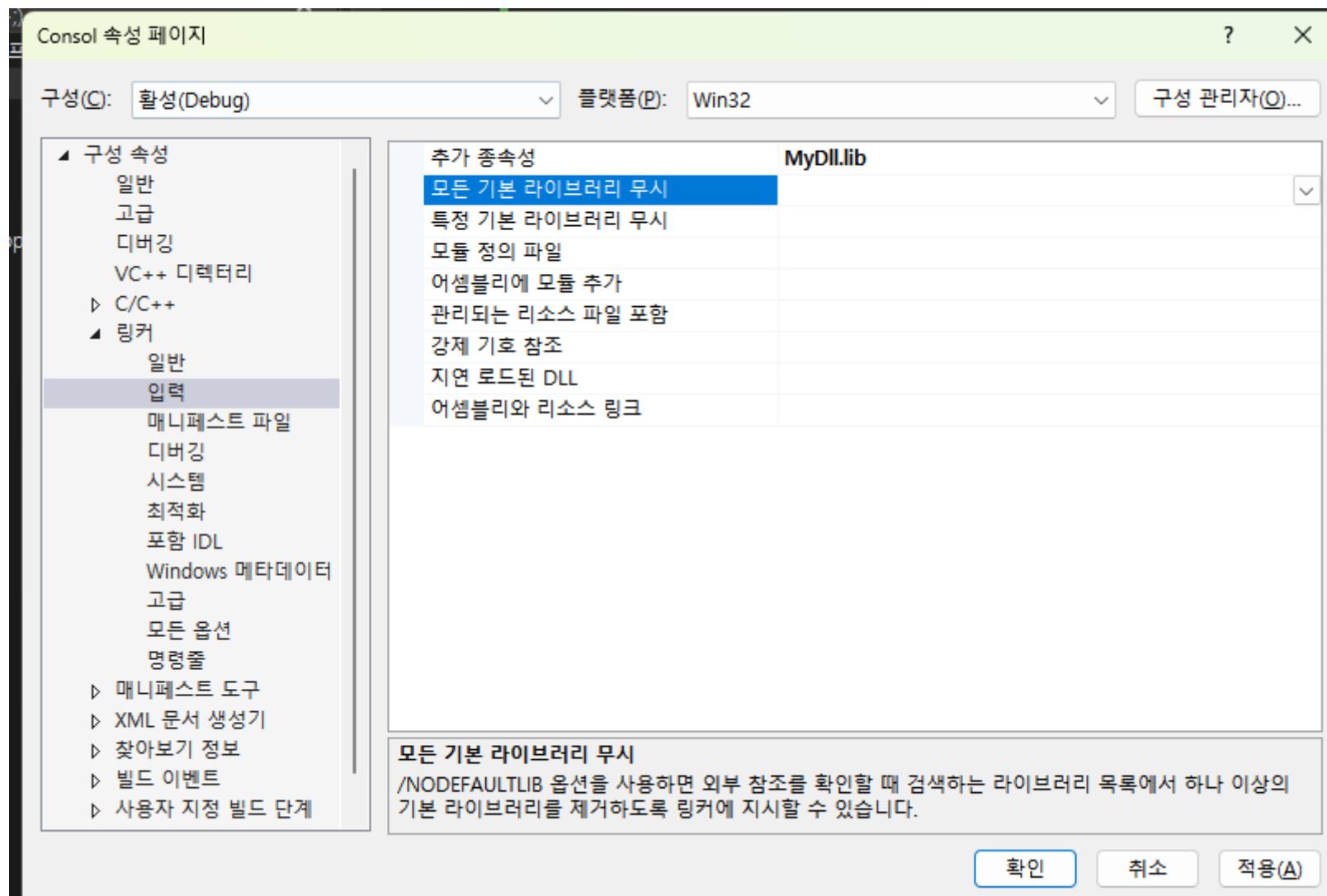


The screenshot shows a code editor window with the following details:

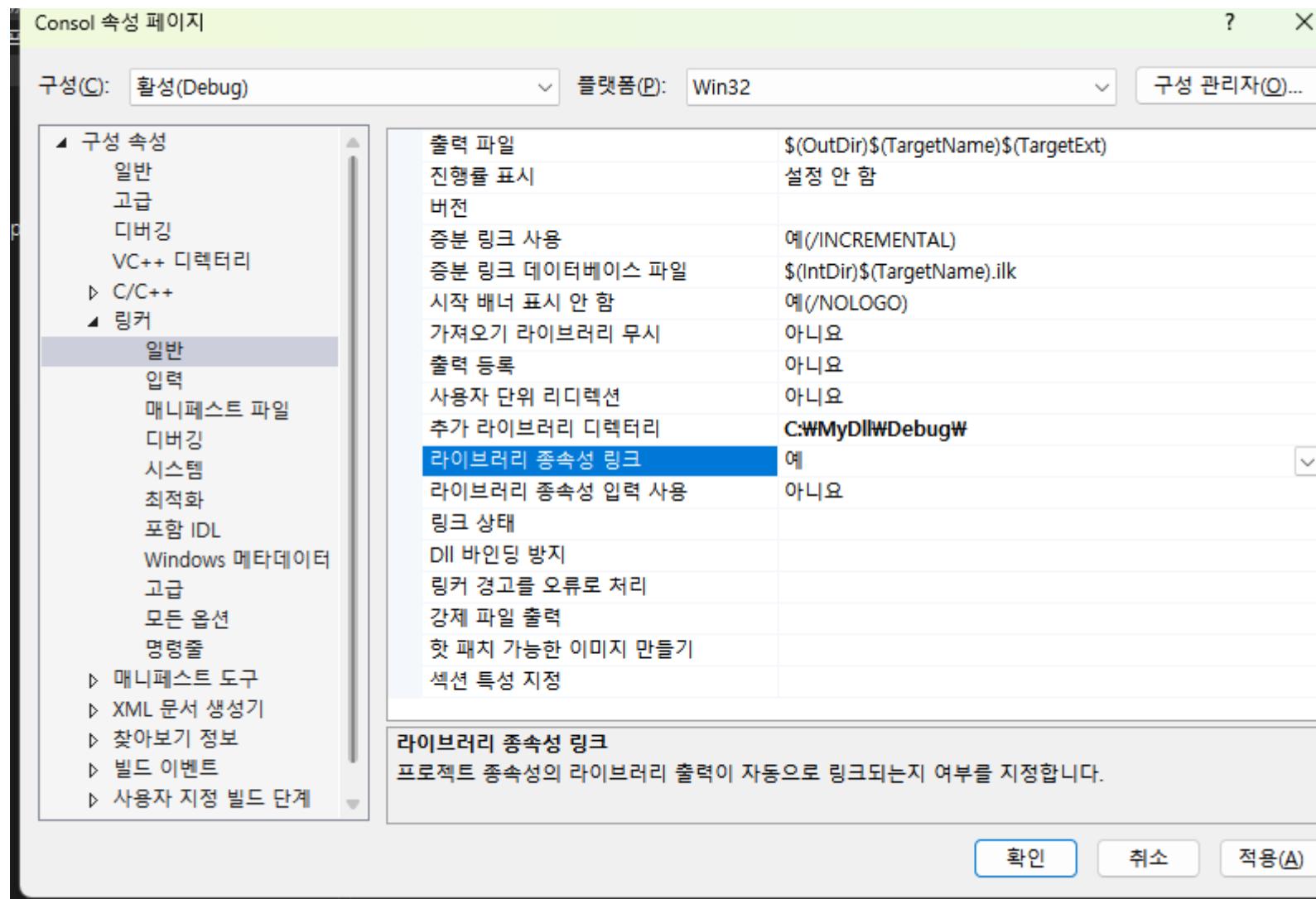
- Title Bar:** Consol.cpp
- Project Selector:** Consol
- Search Bar:** (전역 범위)
- Function Selector:** main()
- Code Content:**

```
1 #include <iostream>
2 #include "dll_wrapper.h"    // 선언을 여기서 가져오면 extern 선언 불필요 위험 없음
3
4 int main() {
5     int r = dll_add(3, 5);    // dll_add를 그대로 호출
6     std::cout << "결과 = " << r << std::endl;
7     return 0;
8 }
```

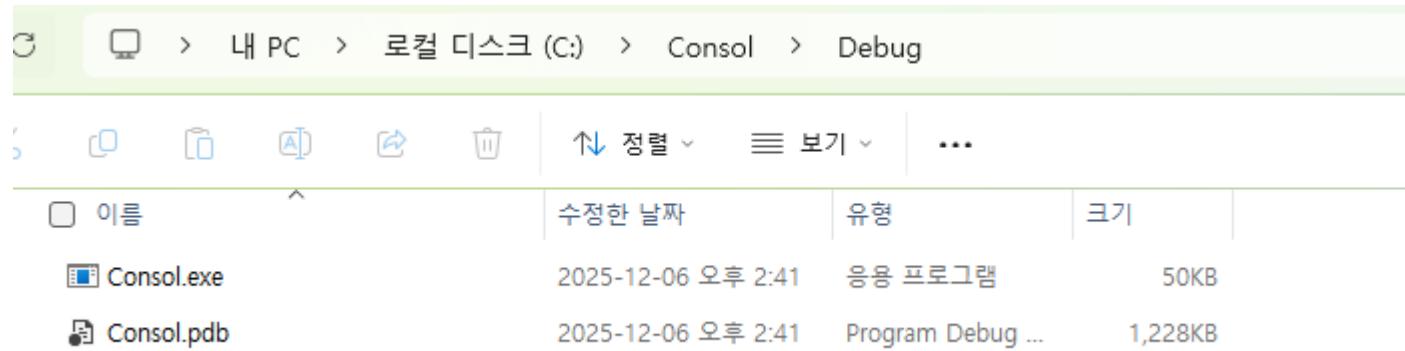
아까 MyDll 프로젝트에 만든 MyDll.lib을 링킹 합니다



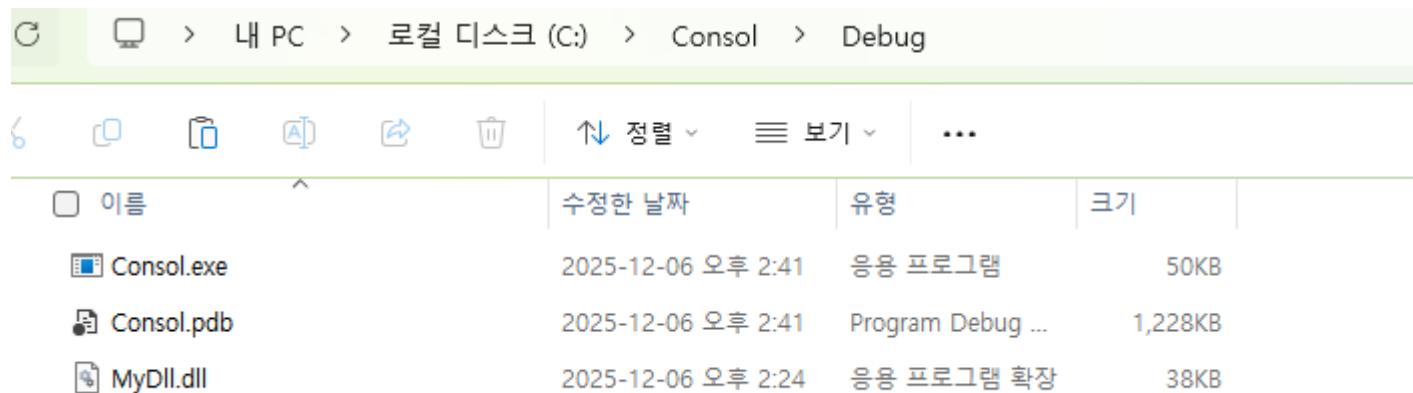
MyDll.lib의 주소를 "추가 라이브러리 딕렉터리"에 넣습니다.



그 후 솔루션 빌드를 하면 C\Consol\Debug 폴더에 .exe가 생성이 됩니다.



하지만 .exe만으로 실행을 못하기에 C\MyDll\Debug 폴더에 있는 MyDll.dll을 복사를 하여 .exe파일과 같은 폴더에 위치 시킵니다.



그 후 visual studio 2022에 들어가 실행을 시키면 코드에 작성한 `dll_add(3,5)`가 작동하여 `r`에 전달이 되고, 결과값 `r`을 콘솔에 나타내 줍니다.

The screenshot shows the Visual Studio 2022 IDE interface. The top bar displays the file name "Consol.cpp". The main window shows the C++ code for the "main" function:

```
1 #include <iostream>
2 #include "dll_wrapper.h" // 선언을 여기서 가져오면 extern 선언 불필요 위험 없음
3
4 int main() {
5     int r = dll_add(3, 5); // dll_add를 그대로 호출
6     std::cout << "결과 = " << r << std::endl;
7     return 0;
8 }
```

The screenshot shows the Visual Studio 2022 debugger window. The title bar says "Visual Studio 디버그". The main area displays the output of the program's execution:

```
결과 = 8
Debug\Consol.exe(프로세스 10656)이 (가) 0 코드(0x0)와 함께 종료되었습니다.
```