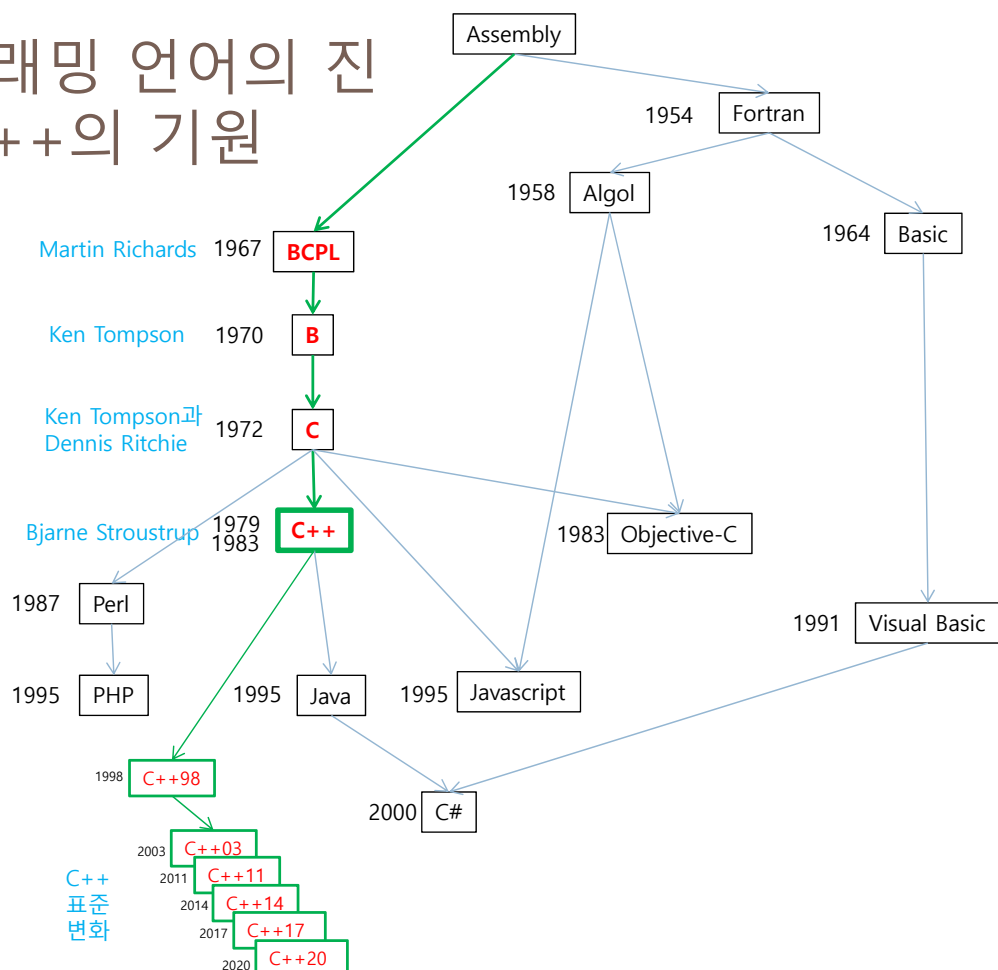




C++ 시작

## 프로그래밍 언어의 진화와 C++의 기원



# 표준 C++ 프로그램의 중요성

3

## □ C++ 언어의 표준

- 1998년 미국 표준원(ANSI, American National Standards Institute)
  - C++ 언어에 대한 표준 설정
- ISO/IEC 14882 문서에 작성됨. 유료 문서
- 표준의 진화
  - 1998년(C++98), 2003년(C++03), 2007년(C++TR1), 2011년(C++11)

## □ 표준의 중요성

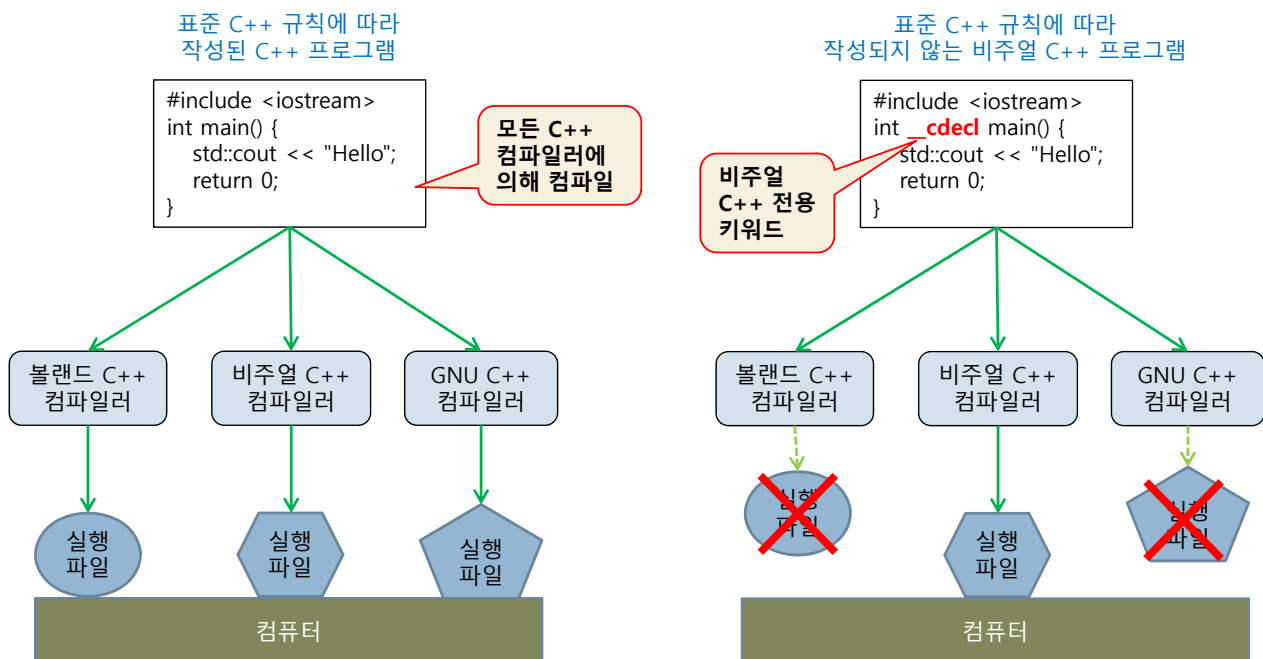
- 표준에 의해 작성된 C++ 프로그램
  - 모든 플랫폼. 모든 표준 C++ 컴파일러에 의해 컴파일
  - 동일한 실행 결과 보장
  - 운영체제와 컴파일러의 종류에 관계없는 높은 호환성

## □ 비 표준 C++ 프로그램

- Visual C++, Borland C++ 등 컴파일러 회사 고유의 비 표준 구문
  - 특정 C++ 컴파일러에서만 컴파일
- 호환성 결여

# 표준/비표준 C++ 프로그램의 비교

4



비주얼 C++ 도구를 이용하되 C++ 언어 표준에 준하여 프로그래밍 하면 컴파일러나 플랫폼에 상관없이 컴파일되고 실행가능하다.

# C++ 언어의 주요한 설계 목적

5

## 1) C 언어와의 호환성

- C 언어의 문법 체계 계승
  - 소스 레벨 호환성 - 기존에 작성된 C 프로그램을 그대로 가져다 사용
  - 링크 레벨 호환성 - C 목적 파일과 라이브러리를 C++ 프로그램에서 링크

## 2) 객체 지향 개념 도입

- 캡슐화, 상속, 다형성
- 소프트웨어의 재사용을 통해 생산성 향상
- 복잡하고 큰 규모의 소프트웨어의 작성, 관리, 유지보수 용이

## 3) 엄격한 타입 체크

- 실행 시간 오류의 가능성을 줄임
- 디버깅 편리

## 4) 실행 시간의 효율성 저하 최소화

- 실행 시간을 저하시키는 요소와 해결
  - 작은 크기의 멤버 함수 잦은 호출 가능성 -> 인라인 함수로 실행 시간 저하 해소

# C 언어에 추가한 기능

6

## □ 함수 중복(function overloading)

- 매개 변수의 개수나 타입이 다른 동일한 이름의 함수들 선언

## □ 디폴트 매개 변수(default parameter)

- 매개 변수에 디폴트 값이 전달되도록 함수 선언

## □ 참조와 참조 변수(reference)

- 하나의 변수에 별명을 사용하는 참조 변수 도입

## □ 참조에 의한 호출(call-by-reference)

- 함수 호출 시 참조 전달

## □ new/delete 연산자

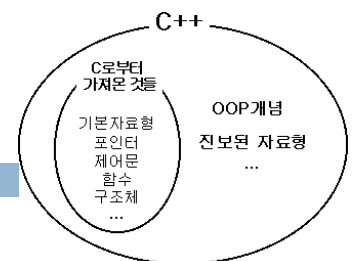
- 동적 메모리 할당/해제를 위해 new와 delete 연산자 도입

## □ 연산자 재정의

- 기존 C++ 연산자에 새로운 연산 정의

## □ 제네릭 함수와 클래스

- 데이터 타입에 의존하지 않고 일반화시킨 함수나 클래스 작성 가능



# C++ 객체 지향 특성 - ①캡슐화

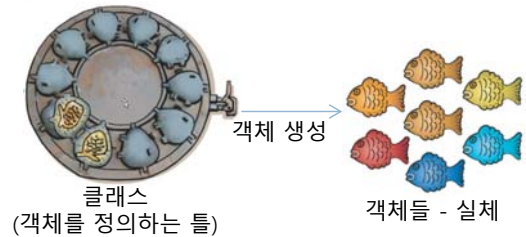
7

## □ 캡슐화(Encapsulation)

- 데이터를 캡슐로 싸서 외부의 접근으로부터 보호
- C++에서 클래스(class 키워드)로 캡슐 표현

## □ 클래스와 객체

- 클래스 - 객체를 만드는 틀
- 객체 - 클래스라는 틀에서 생겨난 실체
- 객체(object), 실체(instance)는 같은 뜻



멤버들

```
class Circle {
private:
    int radius; // 반지름 값
public:
    Circle(int r) { radius = r; }
    double getArea() { return 3.14*radius*radius; }
};
```

원을 추상화한 Circle 클래스



원 객체들(실체)

# C++ 객체 지향 특성 - ②상속성

8

## □ 객체 지향 상속(Inheritance)

- 자식이 부모의 유전자를 물려 받는 것과 유사

## □ C++ 상속

- 객체가 자식 클래스의 멤버와 부모 클래스에 선언된 모양 그대로 멤버들을 가지고 탄생



상속 관계 표현

```
class Phone {
    void call();
    void receive();
};

class MobilePhone : public Phone {
    void connectWireless();
    void recharge();
};

class MusicPhone : public MobilePhone {
    void downloadMusic();
    void play();
};
```

C++로 상속 선언



## C++ 객체 지향 특성 - ③다형성

9

### □ 다형성(Polymorphism)

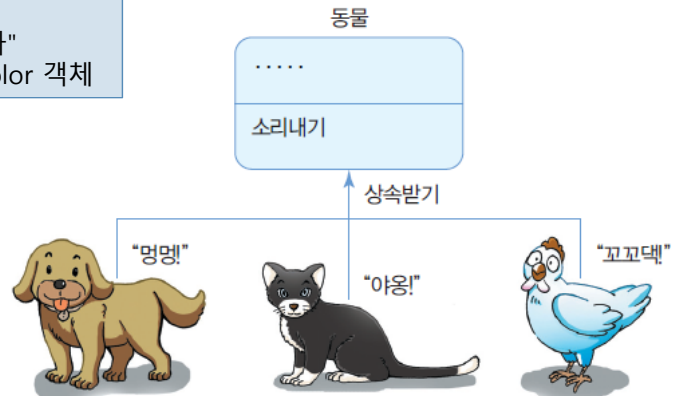
- 하나의 기능이 경우에 따라 다르게 보이거나 다르게 작동하는 현상
- 연산자 중복, 함수 중복, 함수 재정의(overriding)

```
2 + 3          --> 5
"남자" + "여자" --> "남자여자"
redColor 객체 + blueColor 객체 --> purpleColor 객체
```

+ 연산자 중복

```
void add(int a, int b) { ... }
void add(int a, int b, int c) { ... }
void add(int a, double d) { ... }
```

add 함수 중복



함수 재정의(오버라이딩)

## C++ 언어에서 객체 지향을 도입한 목적

10

### □ 소프트웨어 생산성 향상

- 소프트웨어의 생명 주기 단축 문제 해결 필요
- 기 작성된 코드의 재사용 필요
- C++ 클래스 상속 및 객체 재사용으로 해결

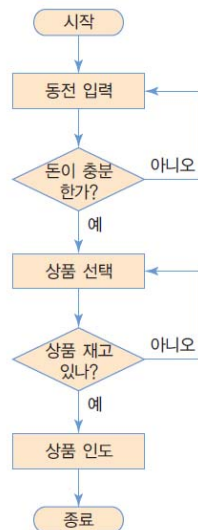
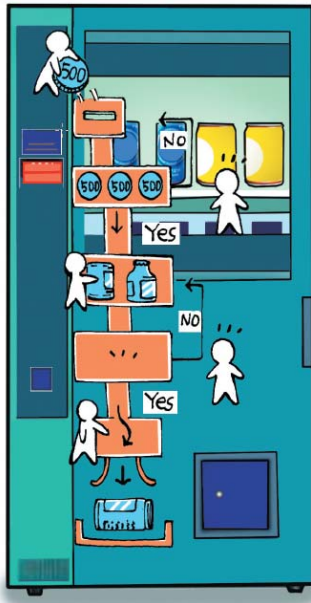
### □ 실세계에 대한 쉬운 모델링

- 과거의 소프트웨어
  - 수학 계산이나 통계 처리에 편리한 절차 지향 언어가 적합
- 현대의 소프트웨어
  - 물체 혹은 객체의 상호 작용에 대한 묘사가 필요
  - 실세계는 객체로 구성된 세계
  - 객체를 중심으로 하는 객체 지향 언어 적합

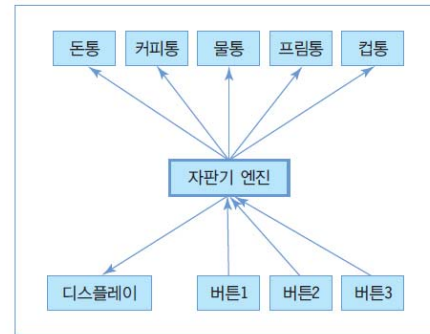


# 절차 지향 프로그래밍과 객체 지향 프로그래밍

11



(a) 절차 지향적 프로그래밍으로 구현할 때의 흐름도



(b) 객체 지향적 프로그래밍으로 구현할 때의 객체 관계도

- 실행하고자 하는 절차대로 일련의 명령어 나열.
- 흐름도를 설계하고 흐름도에 따라 프로그램 작성

- 객체들을 정의하고, 객체들의 상호 관계, 상호 작용으로 구현

## C++와 제네릭 프로그래밍

12

제네릭(generic)이란 데이터의 타입(data type)을 일반화한다(generalize)는 것을 의미

### □ 제네릭 함수와 제네릭 클래스

#### □ 제네릭 함수(generic function)

- 동일한 프로그램 코드에 다양한 데이터 타입을 적용할 수 있게 일반화 시킨 함수

#### □ 제네릭 클래스(generic class)

- 동일한 프로그램 코드에 다양한 데이터 타입을 적용할 수 있게 일반화 시킨 클래스

#### □ template 키워드로 선언

- 템플릿 함수 혹은 템플릿 클래스라고도 부름

#### □ Java, C# 등 다른 언어에도 동일한 개념 있음

### □ 제네릭 프로그래밍(generic programming)

- 제네릭 함수와 제네릭 클래스를 활용하여 프로그램을 작성하는 새로운 프로그래밍 패러다임
- 점점 중요성이 높아지고 있음

# C++ 언어의 아킬레스

13

## □ C++ 언어는 C 언어와의 호환성 추구

### ▣ 장점

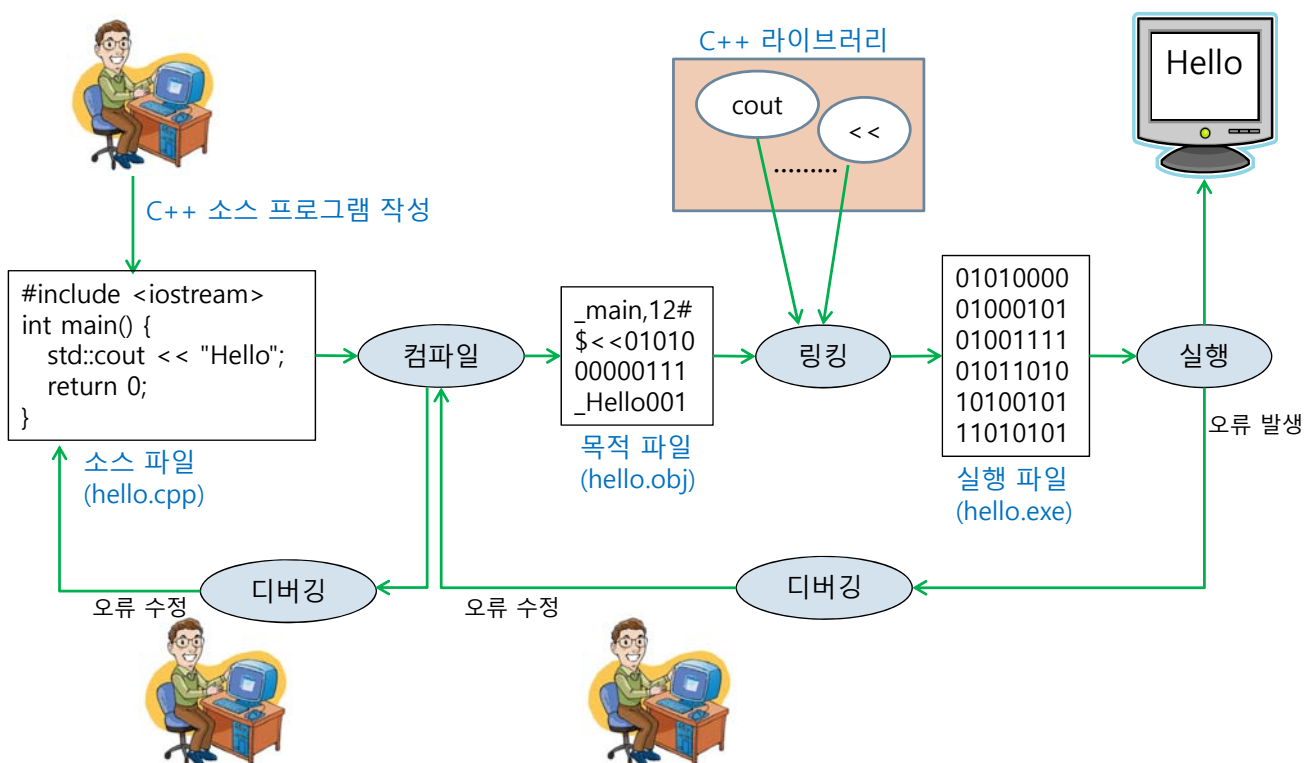
- 기존에 개발된 C 프로그램 코드 활용

### ▣ 단점

- 캡슐화의 원칙이 무너지기
  - C++에서 전역 변수와 전역 함수를 사용할 수 밖에 없음
  - 부작용(side effect) 발생 염려

# C++ 프로그램 개발 과정

14



# C++ 프로그램 작성 및 컴파일

15

## □ 편집

- C++ 소스 프로그램은 텍스트 파일
  - 아무 텍스트 편집기로 편집 가능
- C++ 소스 프로그램의 표준 확장자는 **.cpp**
- C++ **통합 개발 소프트웨어 이용** 추천
  - C++ 소스 편집, 컴파일, 링킹, 실행, 디버깅 등 모든 단계 통합 지원
  - 대표적인 소프트웨어 - **Visual Studio**

## □ 컴파일

- C++ 소스 프로그램을 기계어를 가진 목적 파일로 변환
  - cpp 파일을 **obj** 파일로 변환

**int main() {** 라인을 컴파일한 기계어 코드

어셈블리어 코드

```
_main PROC
; 3 : int main() {
00000 55      push    ebp
00001 8b ec    mov     ebp, esp
00003 81 ec c0 00 00 sub     esp, 192
00009 53      push    ebx
0000a 56      push    esi
0000b 57      push    edi
0000c 8d bd 40 ff ff lea     edi, DWORD PTR [ebp-192]
00012 b9 30 00 00 00 mov     ecx, 48
00017 b8 cc cc cc cc mov     eax, -858993460
0001c f3 ab    rep stosd
; COMDAT
; 000000c0H
; 00000030H
; ccccccccH
; 4 : std::cout << "Hello";
0001e 68 00 00 00 00 push   OFFSET ??_C@_05COLMCDPH@Hello?$AA@
00023 a1 00 00 00 00 mov     eax, DWORD PTR __imp_?cout@std@@@3V?$basic_ostream@DU?$
00028 50      push    eax
00029 e8 00 00 00 00 call   ??$?6U?$char_traits@D@std@@@YAAAV?$basic_ostream@DU?$
0002e 83 c4 08    add     esp, 8
; 5 : return 0;
00031 33 c0     xor     eax, eax
; 6 : }
00033 5f      pop     edi
00034 5e      pop     esi
00035 5b      pop     ebx
00036 81 c4 c0 00 00 add     esp, 192
0003c 3b ec    cmp     ebp, esp
0003e e8 00 00 00 00 call   __RTC_CheckEsp
00043 8b e5    mov     esp, ebp
00045 5d      pop     ebp
00046 c3      ret     0
_main ENDP
```

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Hello";
5     return 0;
6 }
```



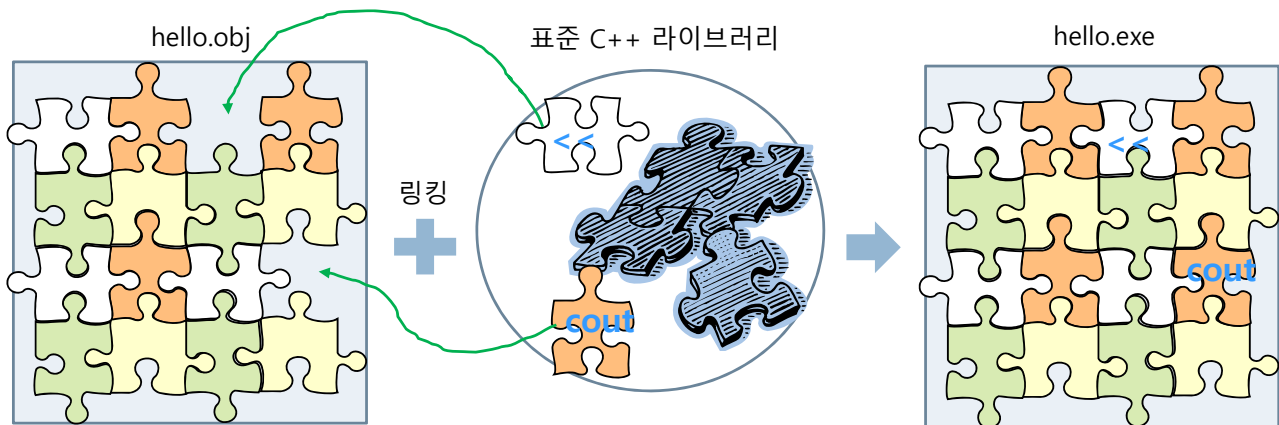
# 링킹

17

## □ 링킹

- 목적 파일끼리 합쳐 실행 파일을 만드는 과정
  - 목적 파일은 바로 실행할 수 없음
- 목적 파일과 C++ 표준 라이브러리의 함수 연결, 실행 파일을 만드는 과정

hello.obj + cout 객체 + << 연산자 함수 => hello.exe를 만듦



## CheckTime

18

## □ C++ 프로그램 개발에서 링킹이 필요한 이유

- ① C++ 프로그램에서 표준 C++ 라이브러리의 함수를 호출할 경우, 개발자가 작성한 코드와 표준 라이브러리 코드를 합쳐 실행 파일을 만드는 과정이 필요하기 때문
- ② C++ 프로그램을 여러 개의 C++ 소스 파일로 나누어 작성할 때, 한 소스 파일에서 다른 소스 파일의 함수를 호출하면 두 프로그램을 합치는 과정이 필요하기 때문
- ③ C++ 언어로 작성된 목적 파일과 C 언어로 작성된 목적 파일을 합쳐 실행 파일을 만드는 과정이 필요하기 때문
- ④ C++ 코드의 디버깅을 효율적으로 하기 위해서는 아님

# 프로그램 실행과 디버깅

19

- 실행 파일은 독립적으로 바로 실행 가능
- 실행 중에 발생하는 오류
  - 원하는 결과가 나오지 않거나 실행 중에 프로그램의 비정상 종료
- 디버깅
  - 실행 중에 발생한 오류를 찾는 과정
  - 디버거
    - 디버깅을 도와주는 프로그램
    - 컴파일러를 만드는 회사에서 함께 공급
  - 소스 레벨 디버깅
    - C++ 소스를 한 라인씩 실행하고 변수 값의 변화를 보면서 오류 발견
    - Visual Studio는 소스 레벨 디버깅 지원

# C++ 표준 라이브러리

20

- C++ 표준 라이브러리는 3 개의 그룹으로 구분
  - ① C 라이브러리
    - 기존 C 표준 라이브러리를 수용, C++에서 사용할 수 있게 한 함수들
    - 이름이 **c**로 시작하는 헤더 파일에 선언됨
  - ② C++ 입출력 라이브러리
    - 콘솔 및 파일 입출력을 위한 라이브러리
  - ③ C++ STL 라이브러리
    - 제네릭 프로그래밍을 지원하기 위해 템플릿 라이브러리

# C++ 표준 라이브러리

21

algorithm	complex	exception	list	stack
bitset	csetjmp	fstream	locale	stdexcept
cassert	csignal	functional	map	strstream
cctype	cstdarg	iomanip	memory	streambuf
cerrno	cstddef	ios	new	string
cfloat	cstdio	iosfwd	numeric	typeinfo
ciso646	cstdlib	iostream	ostream	utility
climits	cstring	istream	queue	valarray
clocale	ctime	iterator	set	vector
cmath	deque	limits	sstream	

C 라이브러리

STL 라이브러리

C++ 입출력 라이브러리

\*〈new〉 헤더 파일은 STL에 포함되지 않는 기타 기능을 구현함

## Visual Studio 시작



### Visual Studio 2019

최근 파일 열기(R)

이번 주

ConsoleApplication1.sln

C:\Users\DSW\Documents\source\repos\ConsoleApplication1

2020-07-14 오후 4:28

시작

리포지토리 복제(C)

GitHub 또는 Azure DevOps 같은 온라인 리포지토리에서 코드 가져오기

프로젝트 또는 솔루션 열기(P)

로컬 Visual Studio 프로젝트 또는.sln 파일 열기

로컬 폴더 열기(F)

폴더 내에서 탐색 및 코드 편집

새 프로젝트 만들기(N)

시작하려면 코드 스캐폴딩과 함께 프로젝트 템플릿을 선택하세요.

코드를 사용하지 않고 계속하기 →

최근에 생성한 솔루션

기존에 생성한 솔루션

새 프로젝트 생성

# Visual Studio의 솔루션과 프로젝트

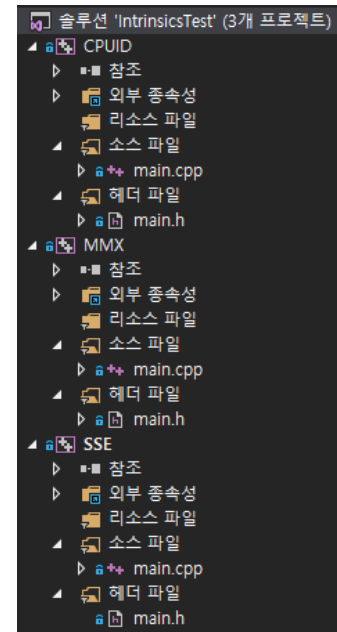
23

## □ 프로젝트(Project)

- 하나의 C++ 프로그램을 작성하기 위해 필요한 소스 파일, 헤더 파일, 리소스 파일, 그리고 컴파일된 목적 파일과 실행 파일, 이들을 관리하기 위한 메타 파일 등을 포함하는 폴더의 개념
- 말 그대로 하나의 프로그램을 만드는 묶음

## □ 솔루션(Solution)

- 하나 이상의 프로젝트가 모인 집합
- 개발자가 작성하고자 하는 소프트웨어를 구성하는 모든 프로젝트를 담는 컨테이너
- "솔루션"을 제공 = 하나 이상의 제품을 한 번에 제공
- 여러 개의 실행 파일과 라이브러리를 묶어 하나의 "솔루션"으로 관리



## 새 프로젝트 만들기



24

## 새 프로젝트 구성

Windows 데스크톱 마법사 콘솔 C++ 데스크톱 라이브러리 Windows

프로젝트 이름(N)

Hello

위치(L)

C:\Users\DSW\Documents\source\repos\

솔루션 이름(M) ⓘ

chap1

☒ 솔루션 및 프로젝트를 같은 디렉터리에 배치(B)

뒤로(B) 만들기(C)

프로젝트 이름

C:\Users\DSW\Documents\source\repos\chap1\Hello 폴더가 생긴다.

솔루션 위치

C:\Users\DSW\Documents\source\repos\chap1 폴더를 생성한다.

25

## 빈 프로젝트 선택

Windows 데스크톱 프로젝트

애플리케이션 종류(T)

콘솔 애플리케이션(.exe)

추가 옵션:

☒ 빈 프로젝트(B)

☐ 미리 컴파일된 헤더(P)

☐ 기호 내보내기(X)

☐ MFC 헤더(M)

팁: 빈 프로젝트 템플릿을 사용하여 이 종류의 프로젝트를 만들 수도 있습니다.

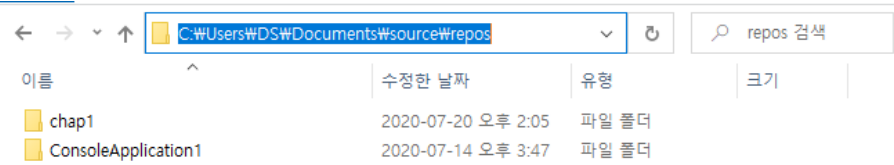
확인 취소

체크해야 함

26

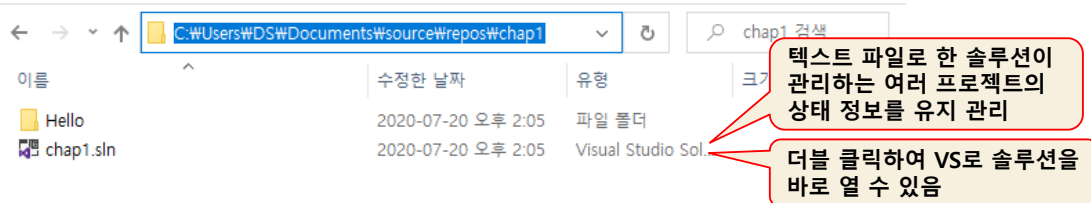
## 솔루션과 프로젝트 폴더

### VS 솔루션 폴더 : chap1 솔루션 폴더 생성



이름	수정한 날짜	유형	크기
chap1	2020-07-20 오후 2:05	파일 폴더	
ConsoleApplication1	2020-07-14 오후 3:47	파일 폴더	

### chap1 폴더 : Hello 프로젝트 폴더 생성

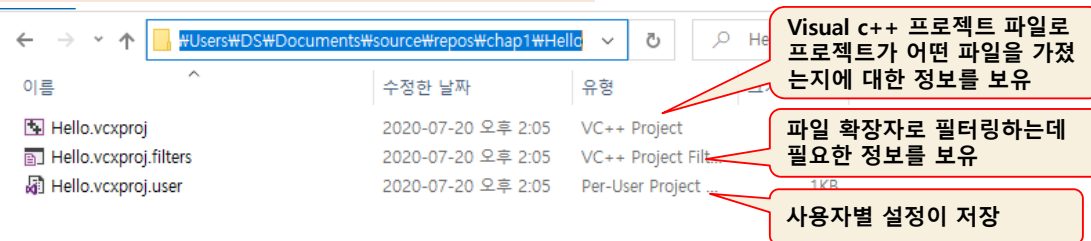


이름	수정한 날짜	유형	크기
Hello	2020-07-20 오후 2:05	파일 폴더	
chap1.sln	2020-07-20 오후 2:05	Visual Studio Sol.	

텍스트 파일로 한 솔루션이 관리하는 여러 프로젝트의 상태 정보를 유지 관리

더블 클릭하여 VS로 솔루션을 바로 열 수 있음

### Hello 폴더 : 관련 파일들이 자동 생성



이름	수정한 날짜	유형	크기
Hello.vcxproj	2020-07-20 오후 2:05	VC++ Project	
Hello.vcxproj.filters	2020-07-20 오후 2:05	VC++ Project Filters	
Hello.vcxproj.user	2020-07-20 오후 2:05	Per-User Project Settings	1KB

Visual c++ 프로젝트 파일로 프로젝트가 어떤 파일을 가졌는지에 대한 정보를 보유

파일 확장자로 필터링하는데 필요한 정보를 보유

사용자별 설정이 저장

27

## Hello 프로젝트 생성 후(1/2)

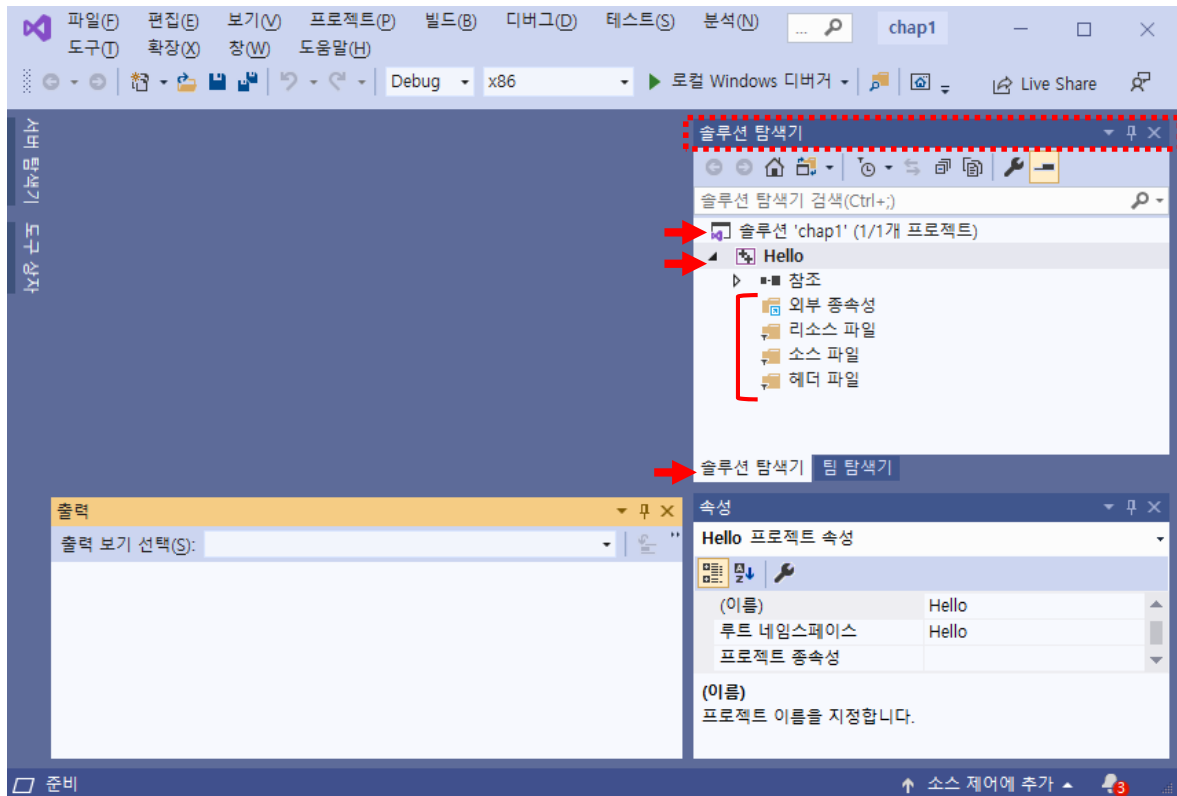
28

### □ 시작을 위한 팁:

- [솔루션 탐색기] 창을 사용하여 파일을 추가/관리합니다.
- [팀 탐색기] 창을 사용하여 소스 제어에 연결합니다.
- [출력] 창을 사용하여 빌드 출력 및 기타 메시지를 확인합니다.
- [오류 목록] 창을 사용하여 오류를 봅니다.



## Hello 프로젝트 생성 후(2/2)

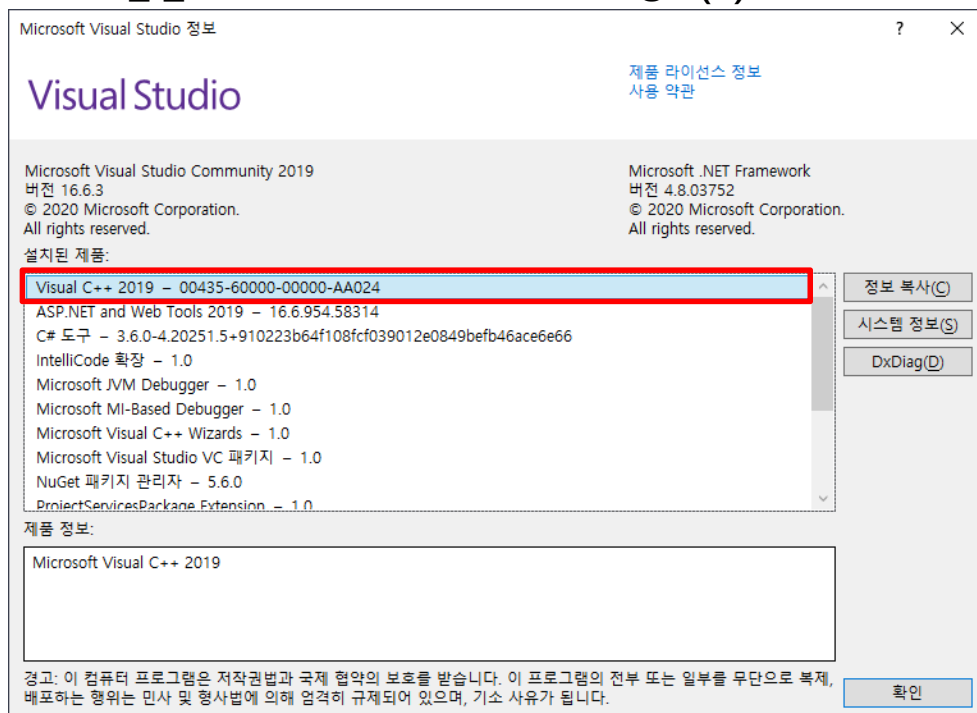


29

## Visual C++ 버전

- Visual C++ 버전 확인

도움말 -> Microsoft Visual Studio 정보(A)

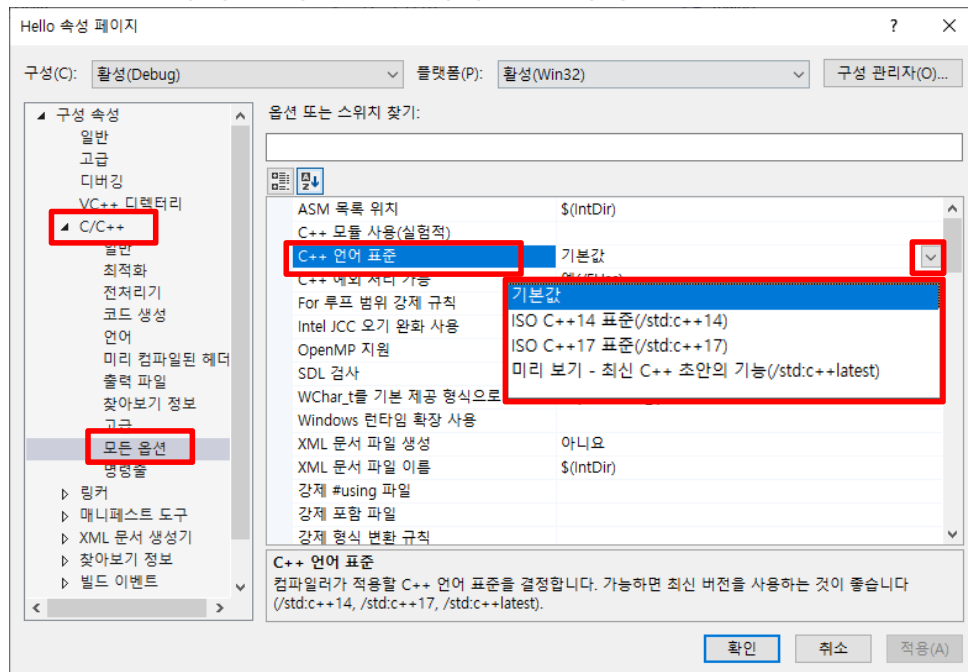


30

# Visual C++ 버전

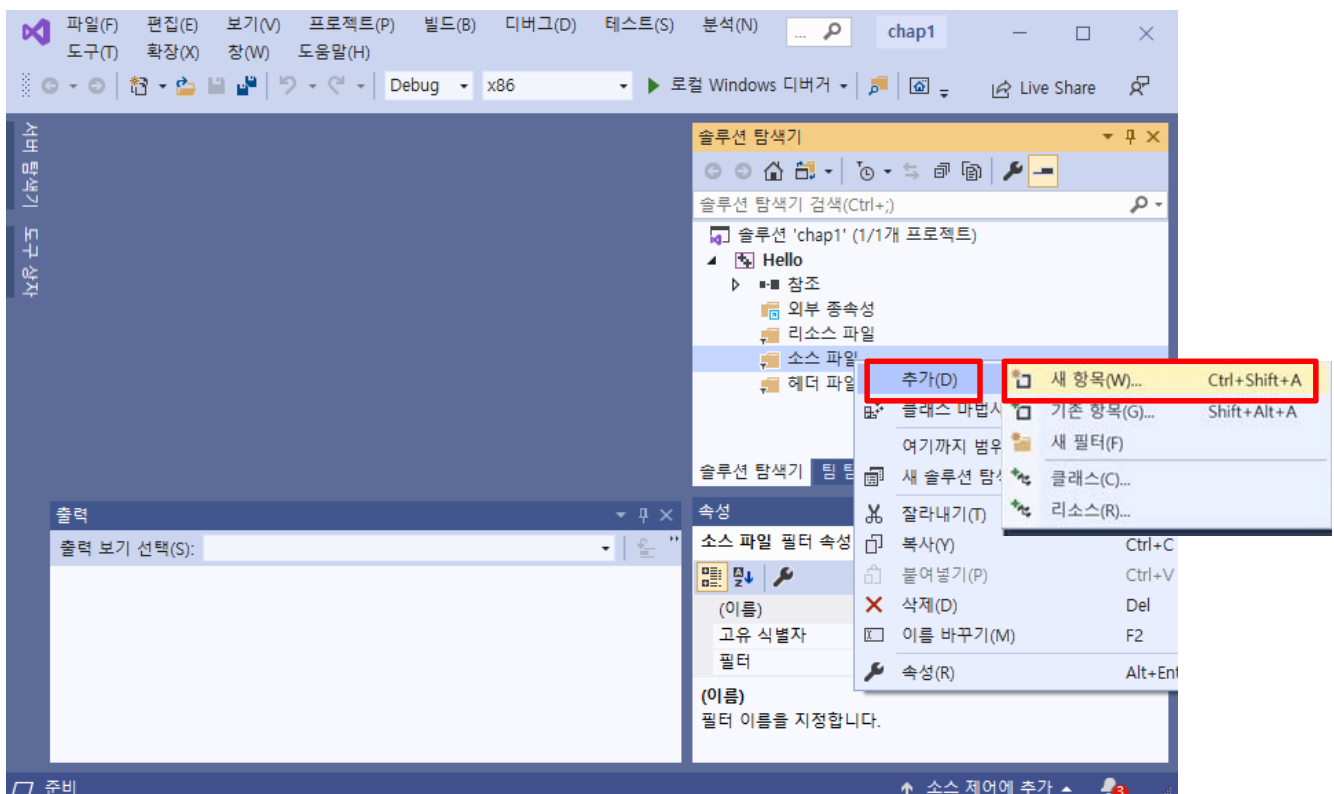
## • Visual C++ 버전 변경방법

1. 프로젝트 -> 속성 -> C/C++ -> 모든 옵션 -> C++ 언어 표준
2. 원하시는 버전을 선택하면 됩니다.

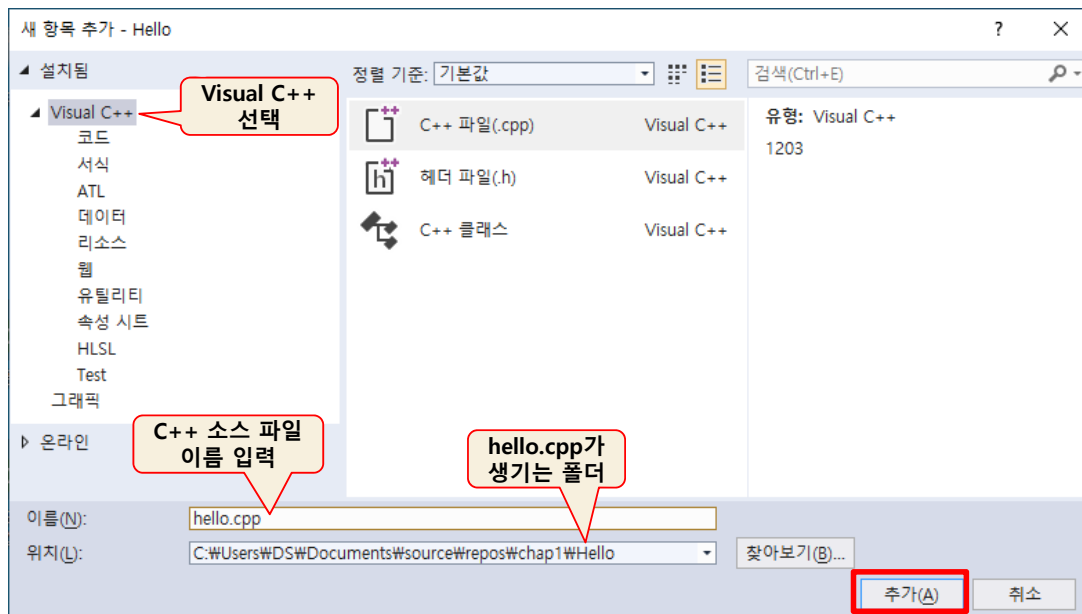


31

# 새 항목 만드는 메뉴 선택



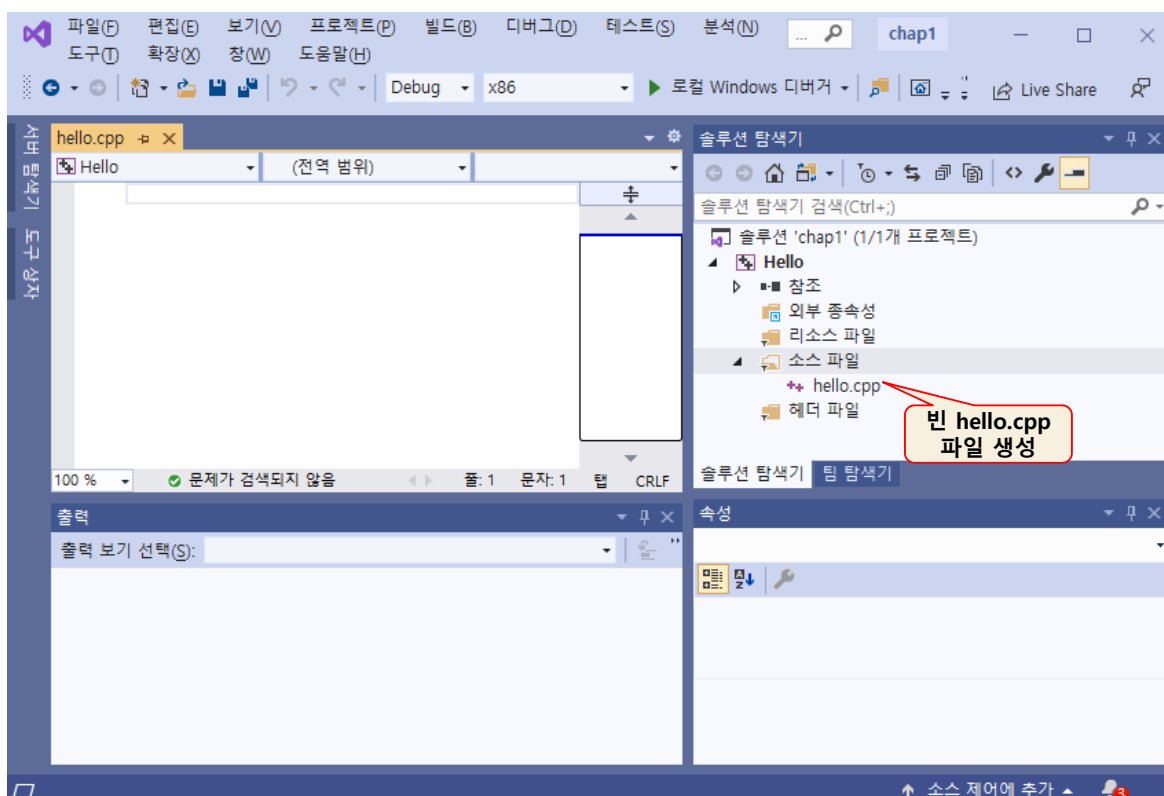
## hello.cpp 소스 파일 생성



이름	수정된 날짜	유형	크기
*.* hello.cpp	2020-06-29 오후 4:46	C++ Source	0KB
Hello.vcxproj	2020-07-20 오후 2:05	VC++ Project	8KB
Hello.vcxproj.filters	2020-07-20 오후 2:05	VC++ Project Fil...	1KB
Hello.vcxproj.user	2020-07-20 오후 2:05	Per-User Project ...	1KB

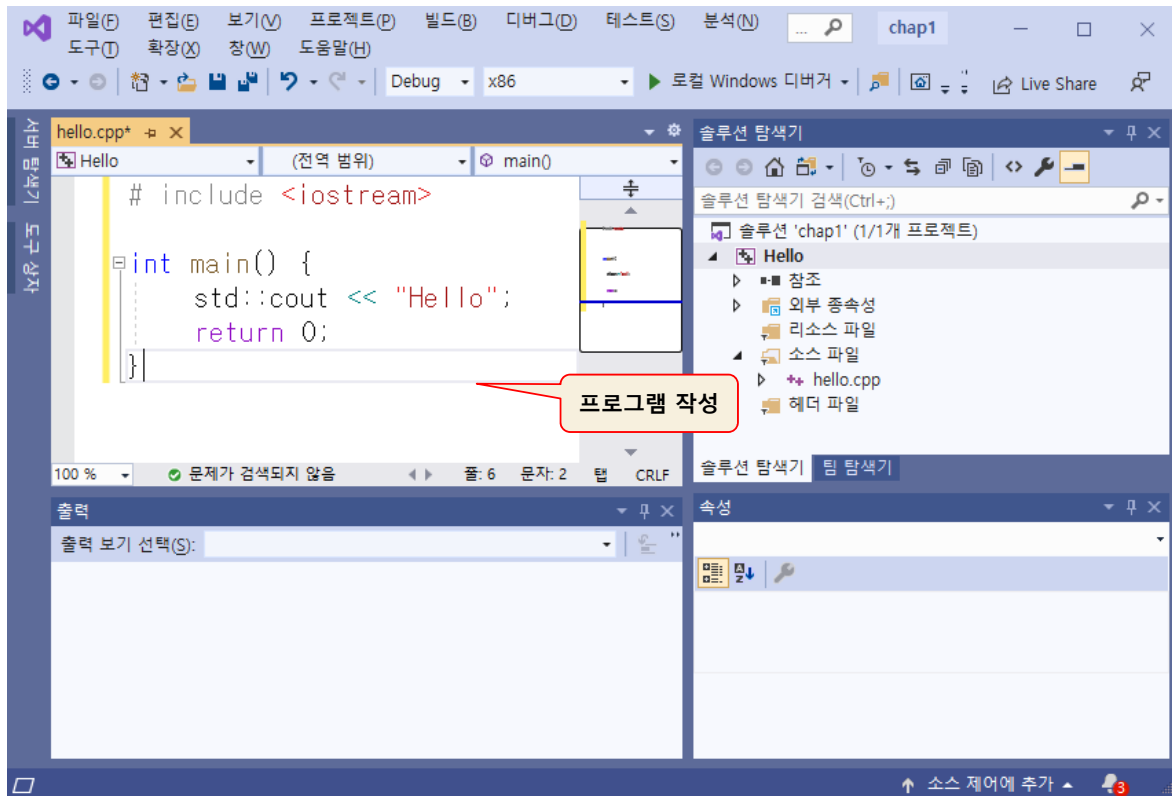
33

## hello.cpp 파일이 생성된 초기 모습



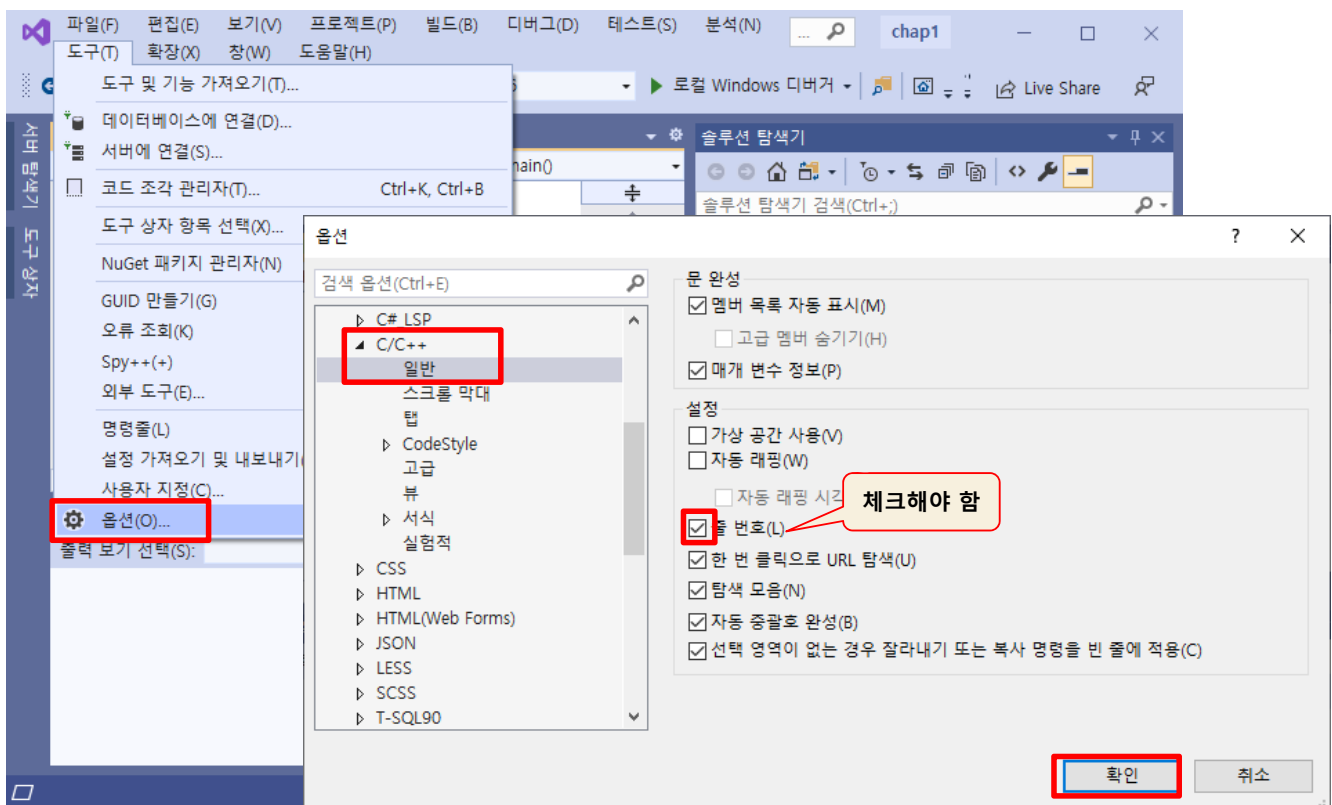
34

# hello.cpp 작성

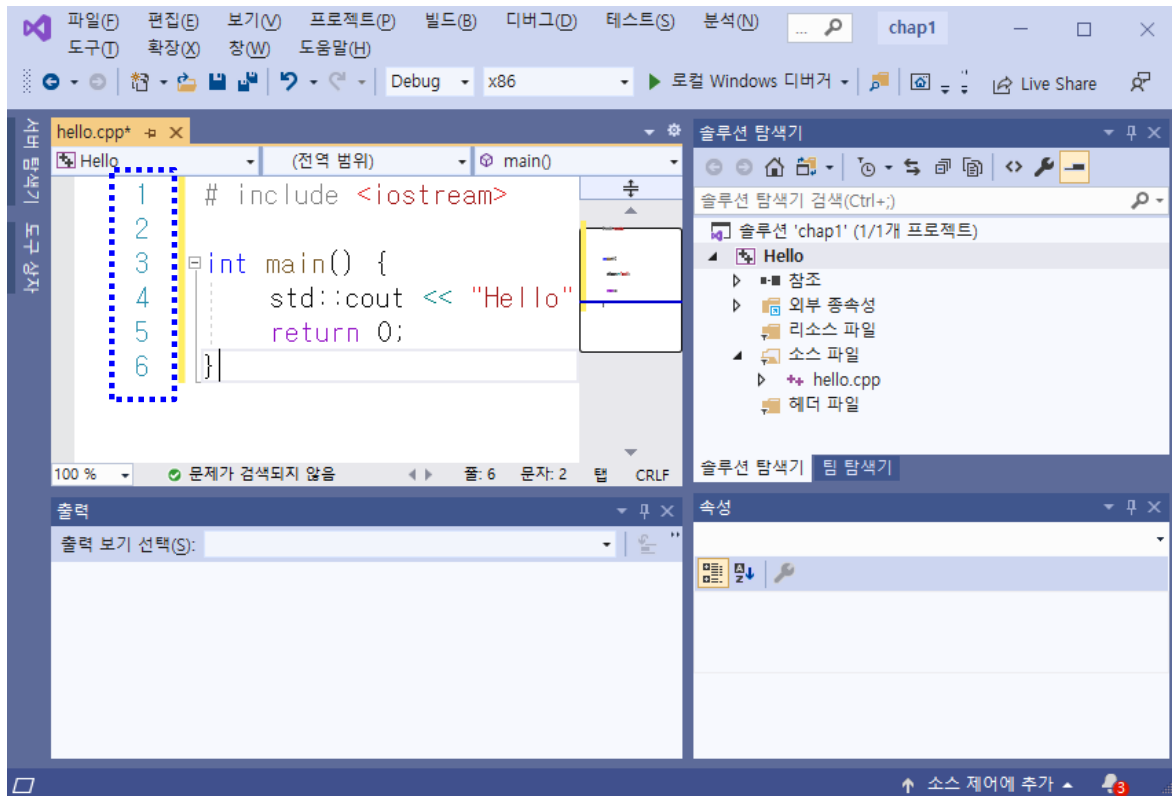


35

# 줄 번호 표시(1/2)



## 줄 번호 표시(2/2)

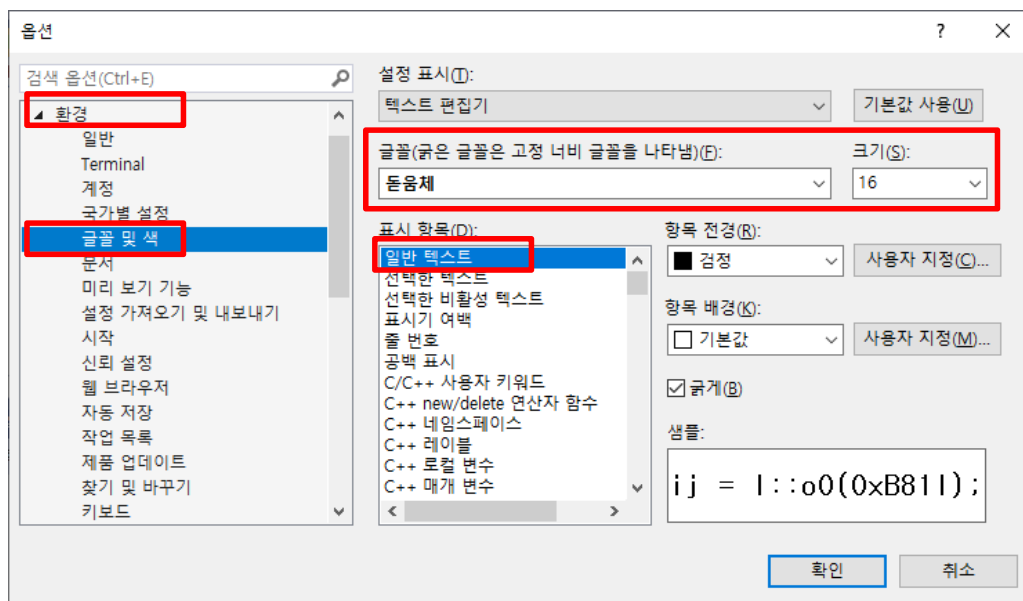


37

## 글꼴, 크기 변경

38

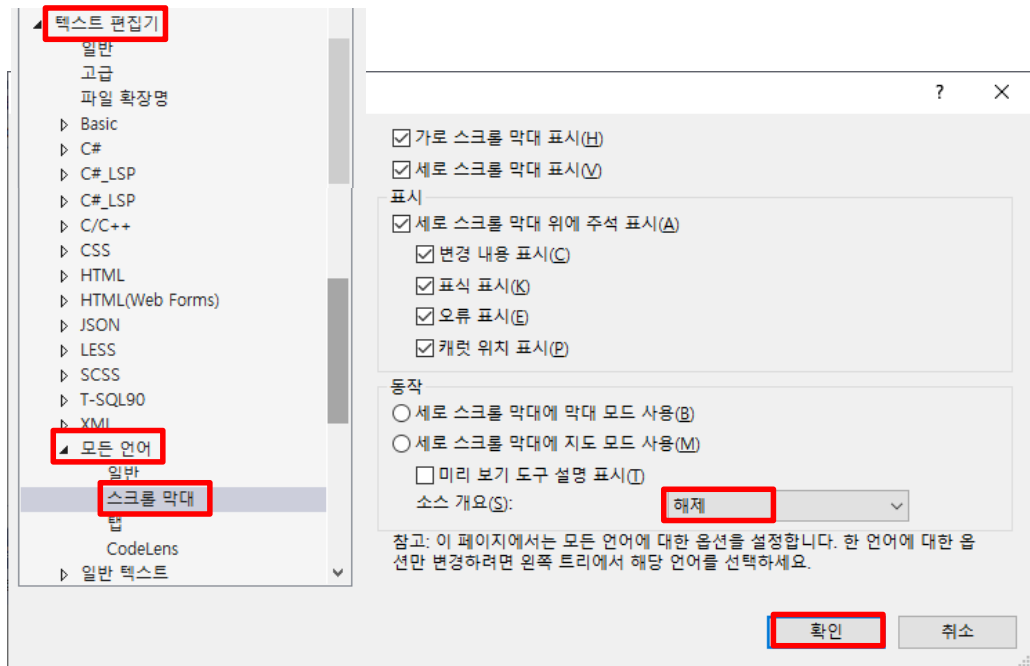
### □ 도구 > 옵션



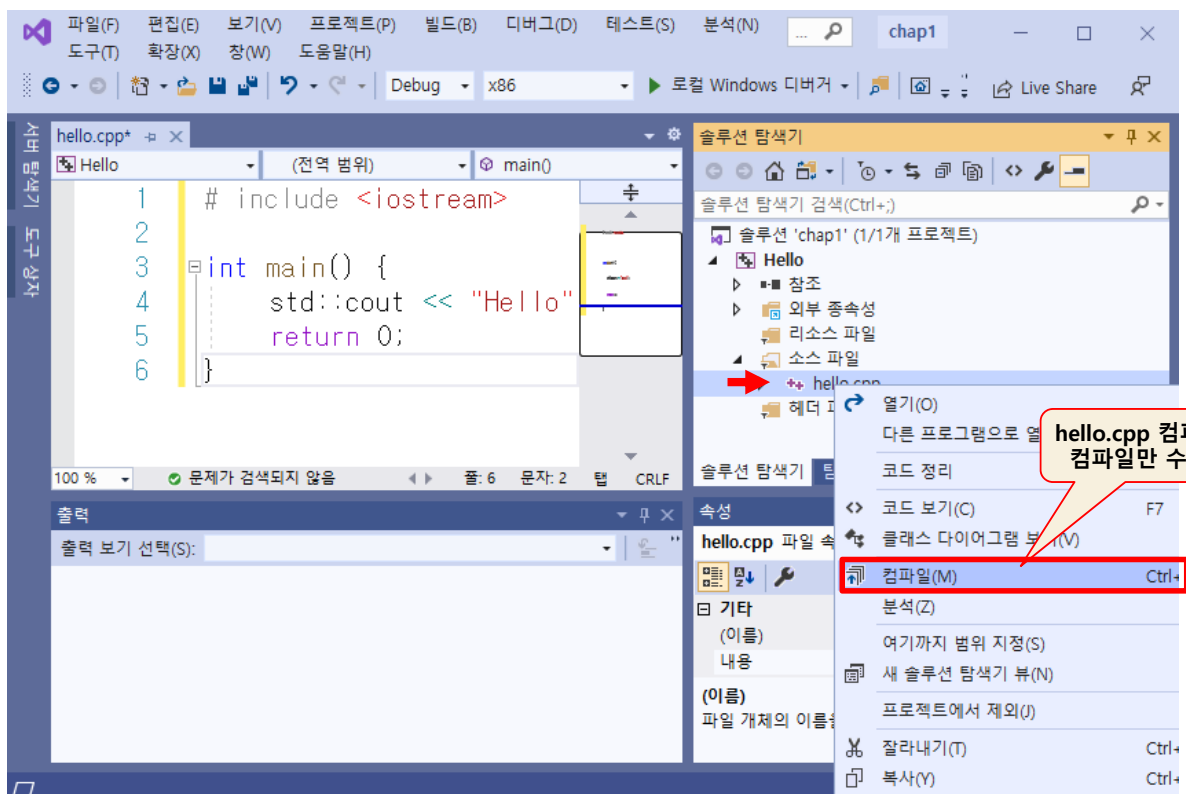
## 세로 스크롤 소스 개요 해제

39

### □ 도구 > 옵션

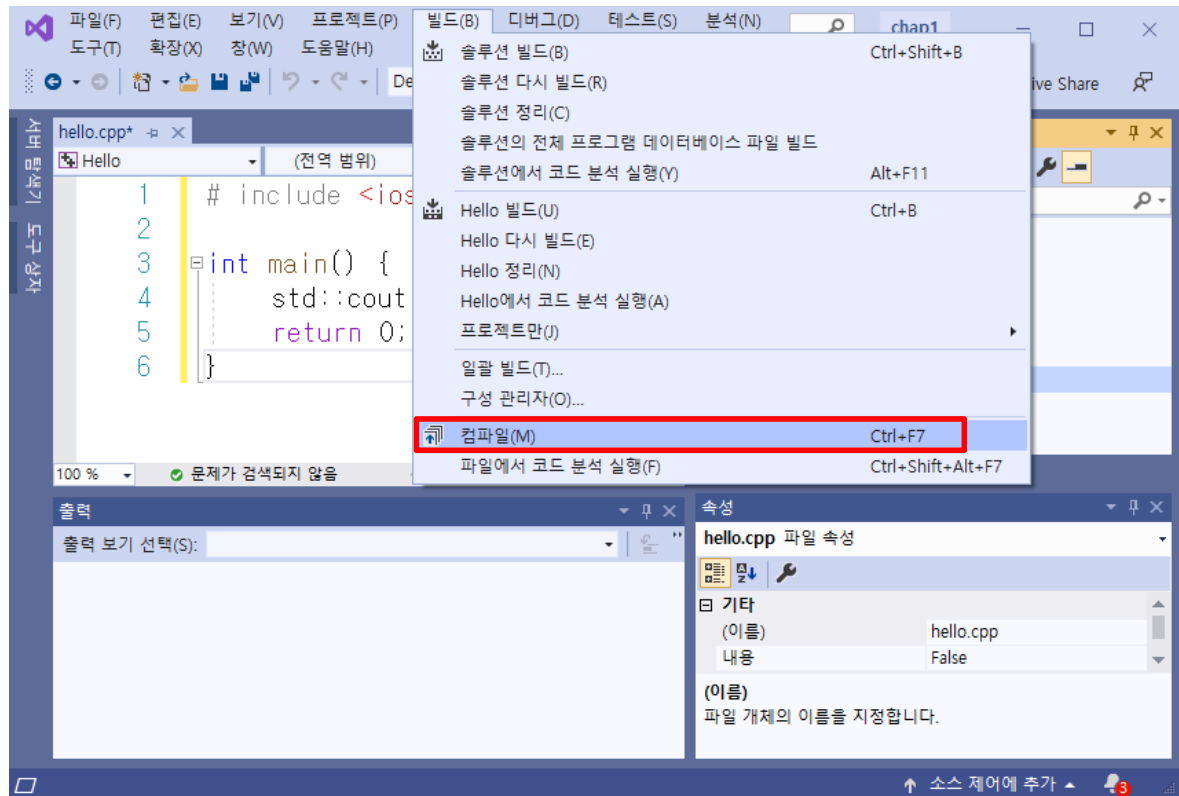


## 솔루션 탐색기에서 컴파일 메뉴 선택



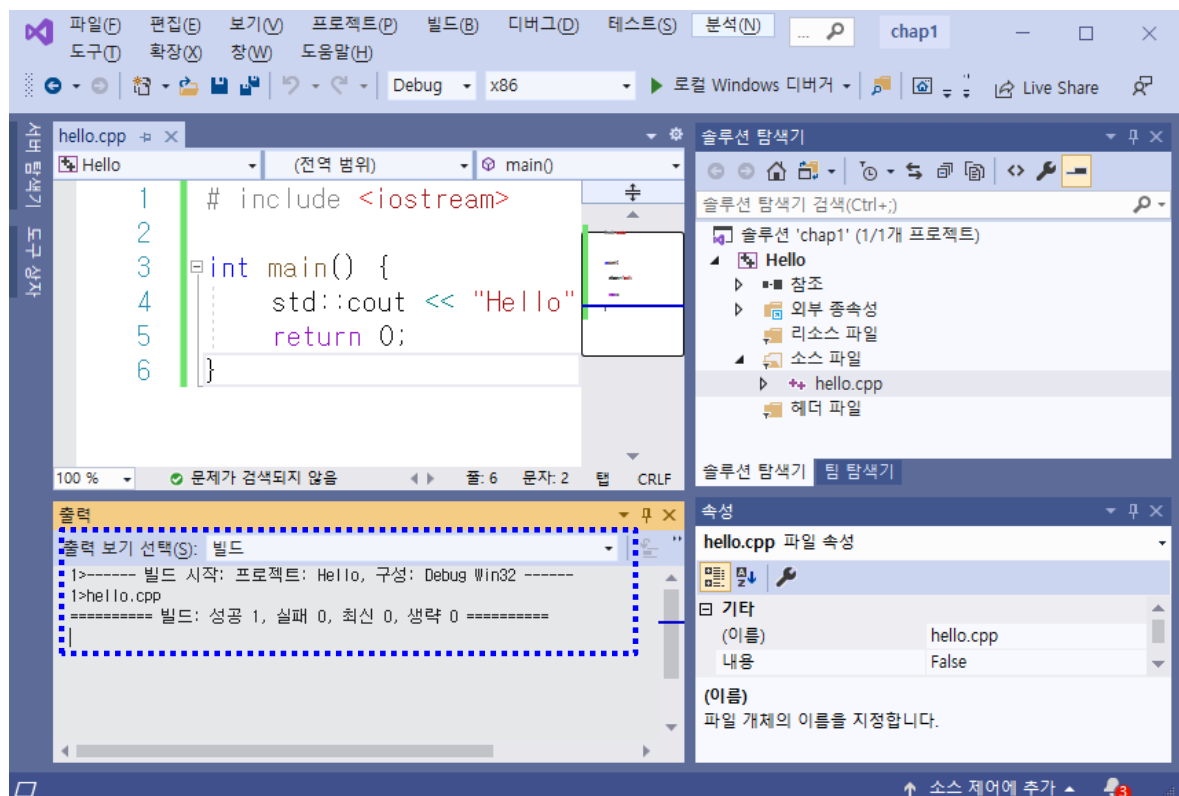


## Hello 프로젝트의 빌드로 컴파일 메뉴 선택



41

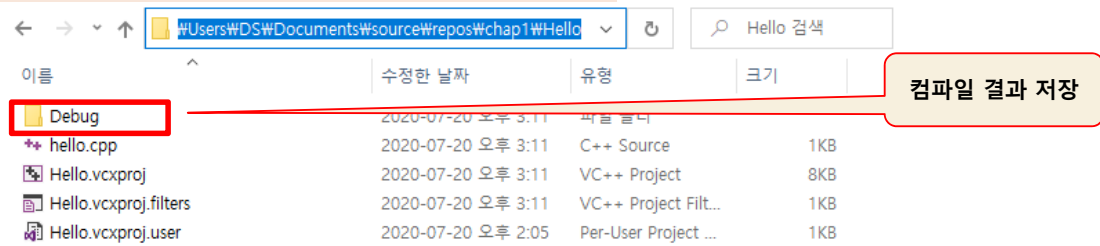
## 컴파일 결과(1/2)



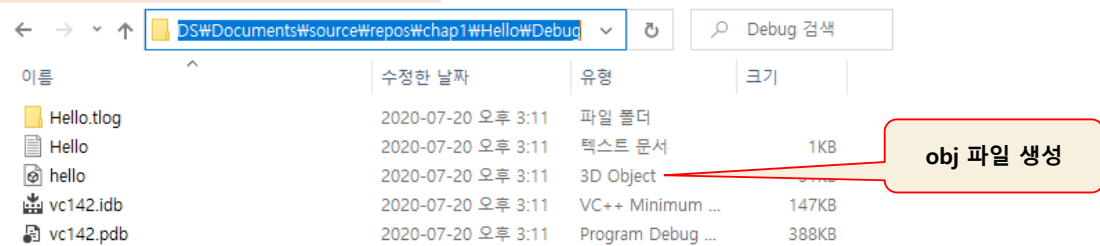
42

## 컴파일 결과(2/2)

### Hello 프로젝트 폴더 : Debug 폴더 생성

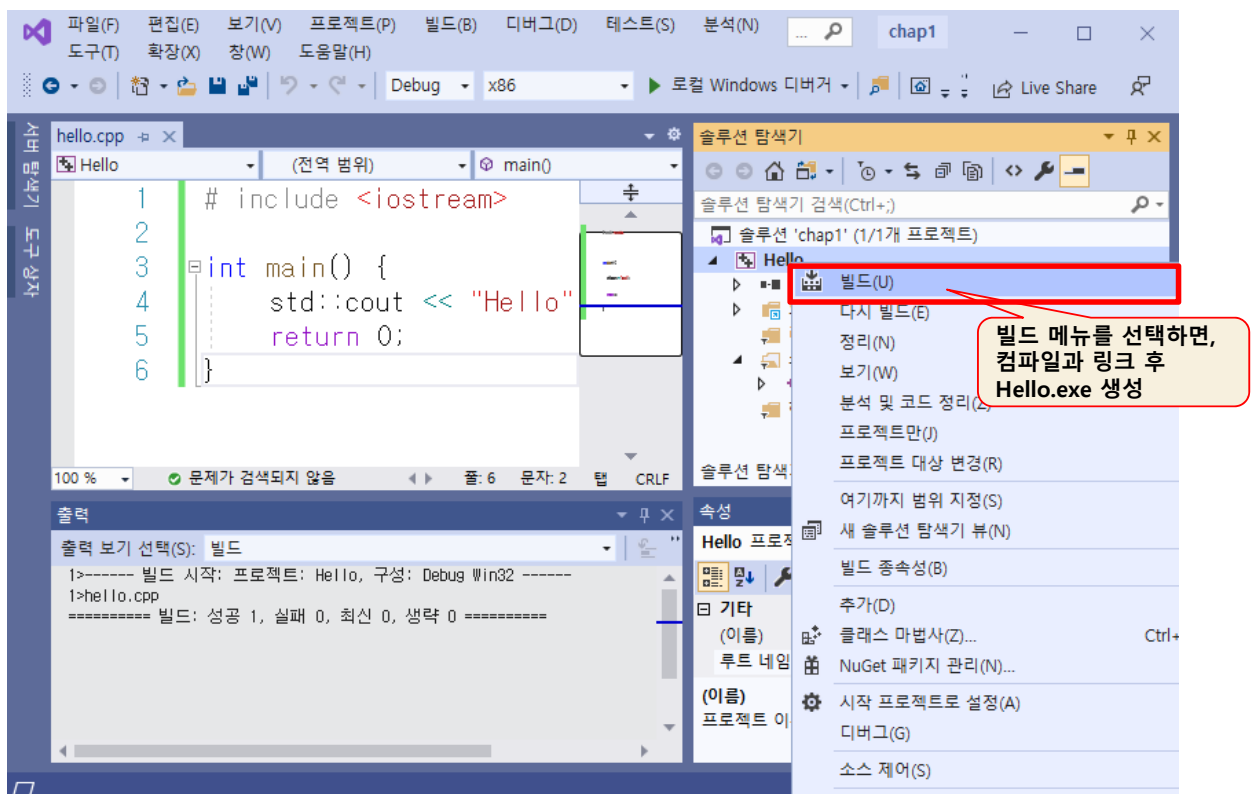


### Debug 폴더 : obj 파일 생성



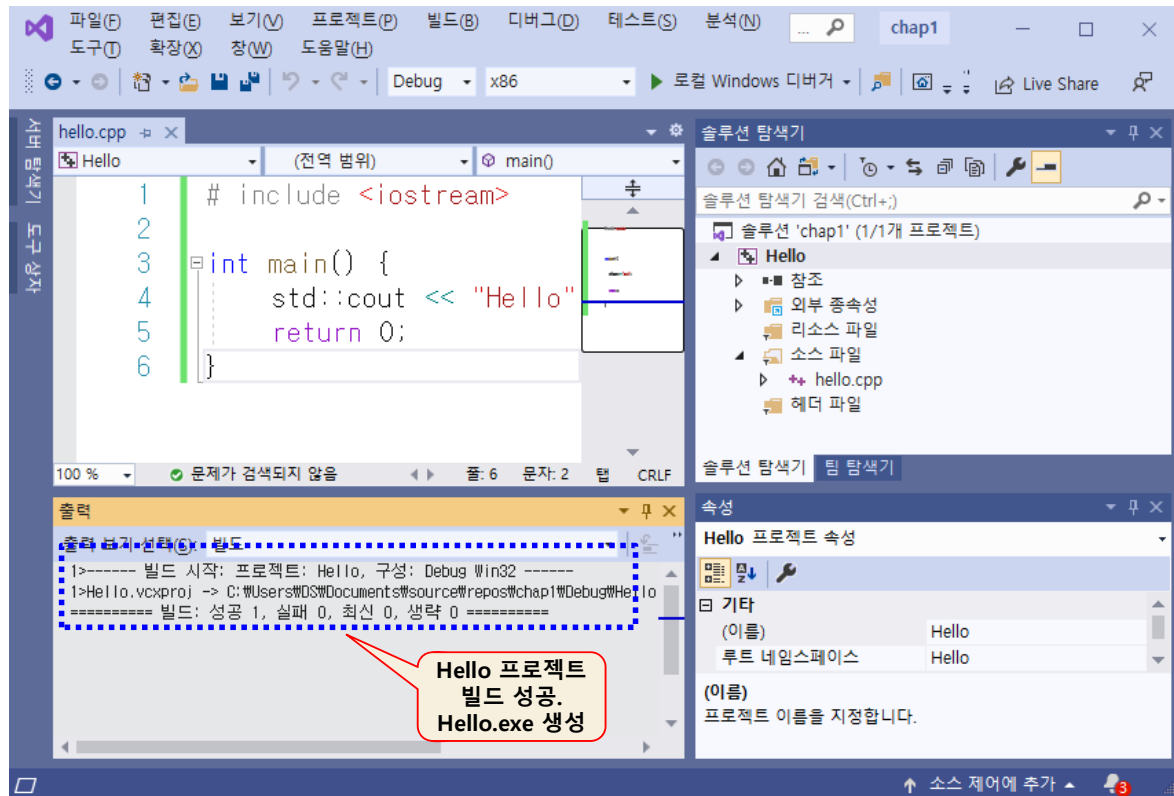
43

## Hello 프로젝트의 빌드로 Hello.exe 생성



44

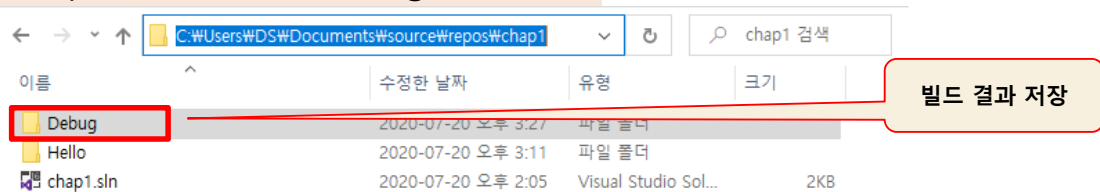
## 빌드 결과(1/2)



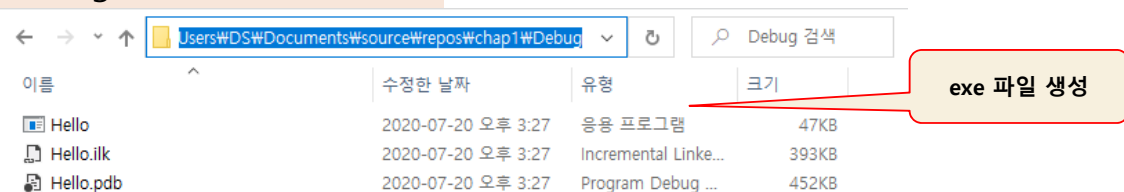
45

## 빌드 결과(2/2)

### Chap1 솔루션 폴더 : Debug 폴더 생성



### Debug 폴더 : exe 파일 생성



46

# Hello 프로젝트 실행



Hello 프로젝트가 실행되는 화면



# 프로그램 실행

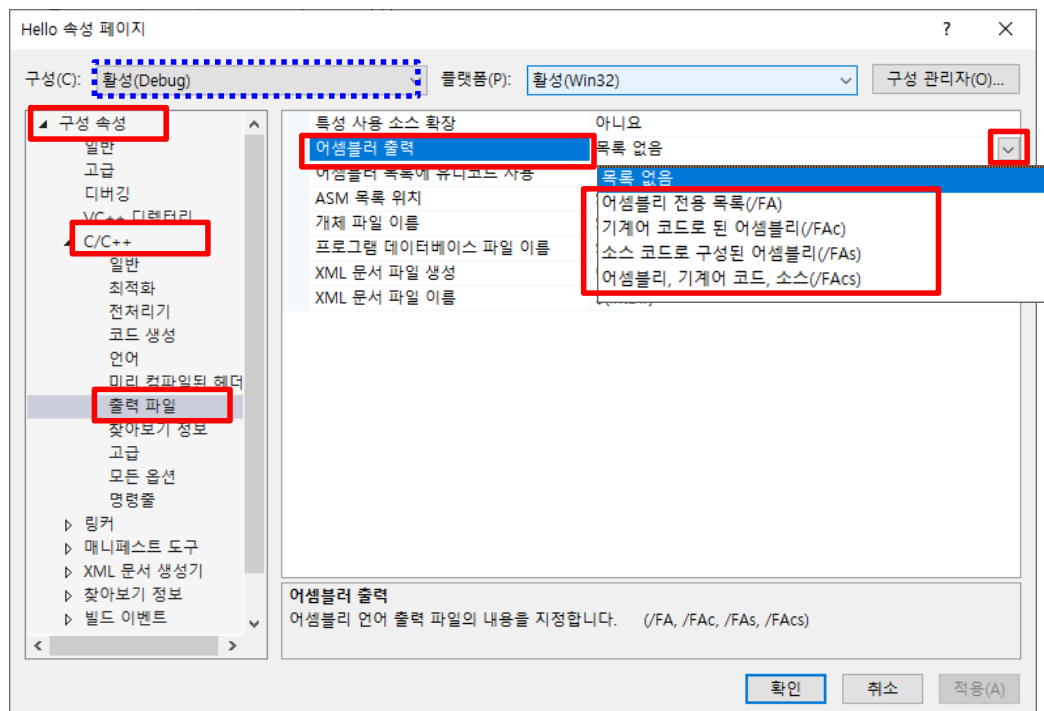
49

- 디버그 > 디버깅 시작(F5)
- 디버그 > 디버그하지 않고 시작(Ctrl+F5)
  - ▣ 소스 코드 변경이 없는 경우
  - ▣ 컴파일과 빌드를 한번에 실행
- 나중에 이 프로젝트를 다시 열려면 [파일] > [열기] > [프로젝트]로 이동하고 **.sln** 파일을 선택

## 어셈블리 코드 보기(1/3)

50

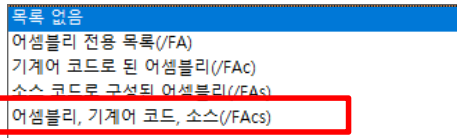
- 프로젝트 > 속성



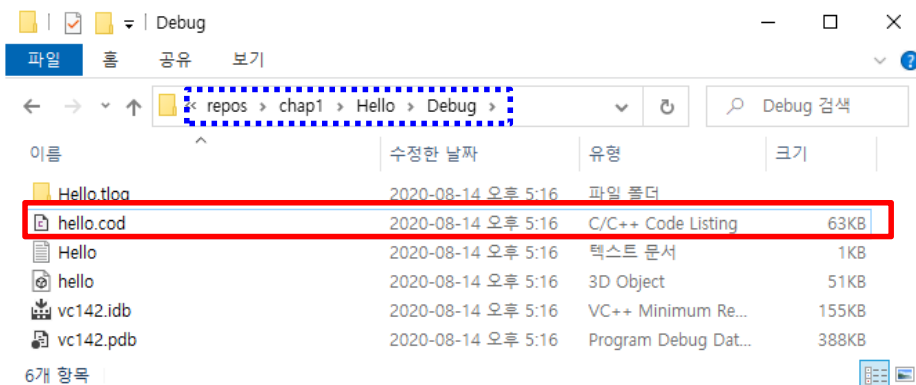
## 어셈블리 코드 보기(2/3)

51

- '어셈블리, 기계어 코드, 소스' 선택 후 확인



- 프로그램 실행
- 프로젝트의 'Debug' 폴더에 Hello.cod 파일 생성



## 어셈블리 코드

52

- Hello.cod 파일

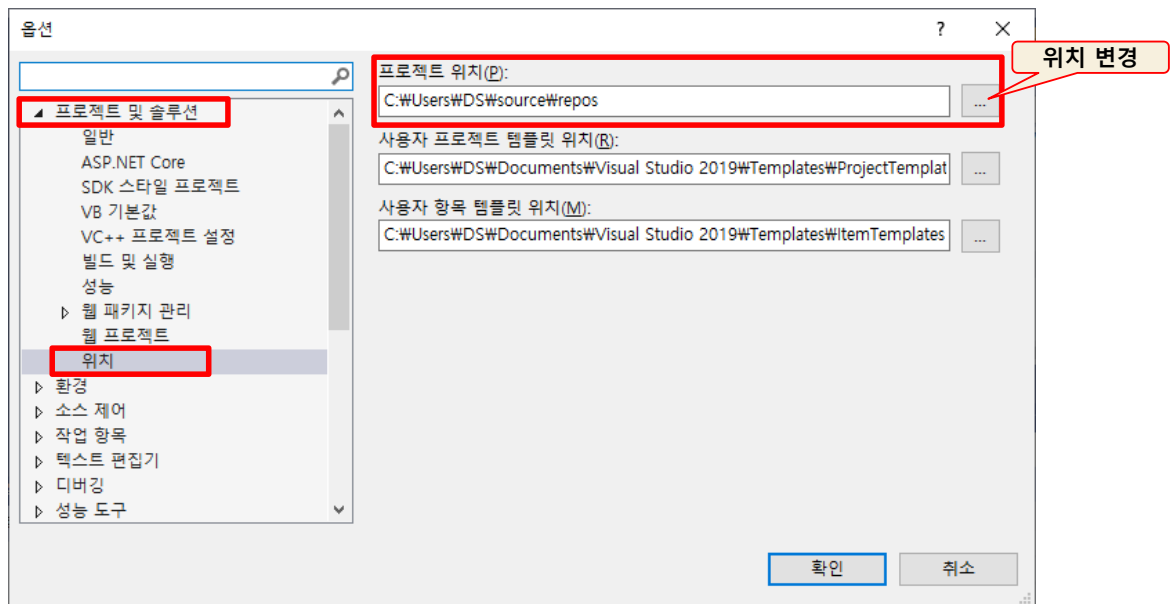
```
1364 ; 3 : int main() {
1365
1366     00000 55      push    ebp
1367     00001 8b ec    mov     ebp, esp
1368     00003 81 ec c0 00 00 sub     esp, 192 ; 000000c0H
1369     00009 53      push    ebx
1370     0000a 56      push    esi
1371     0000b 57      push    edi
1372     0000c 8d bd 40 ff ff lea     edi, DWORD PTR [ebp-192]
1373     00012 b9 30 00 00 00 mov     ecx, 48 ; 00000030H
1374     00017 b8 cc cc cc cc mov     eax, -858993460 ; ccccccccH
1375     0001c f3 ab    rep stosd
1376     0001e b9 00 00 00 00 mov     ecx, OFFSET __5B1B9135_hello@cpp
1377     00023 e8 00 00 00 00 call    @__CheckForDebuggerJustMyCode@4
1378
1379 ; 4 : std::cout << "Hello";
1380
1381     00028 68 00 00 00 00 push    OFFSET ??_C@_05COLMCDPH@Hello@
1382     0002d a1 00 00 00 00 mov     eax, DWORD PTR __imp_?cout@std@@@3V?$$bas
1383     00032 50      push    eax
1384     00033 e8 00 00 00 00 call    ??$?6U?$char_traits@D@std@@@std@@YAAAV?
1385     00038 83 c4 08    add     esp, 8
1386
1387 ; 5 : return 0;
1388
1389     0003b 33 c0      xor     eax, eax
1390
1391 ; 6 : }
1392
1393     0003d 5f      pop     edi
1394     0003e 5e      pop     esi
1395     0003f 5b      pop     ebx
1396     00040 81 c4 c0 00 00 add     esp, 192 ; 000000c0H
1397     00046 3b ec    cmp     ebp, esp
1398     00048 e8 00 00 00 00 call    __RTC_CheckEsp
1399     0004d 8b e5    mov     esp, ebp
1400     0004f 5d      pop     ebp
1401     00050 c3      ret     0
1402 _main ENDP
```



# 프로젝트 저장 위치 설정

53

## □ 디버그 > 옵션



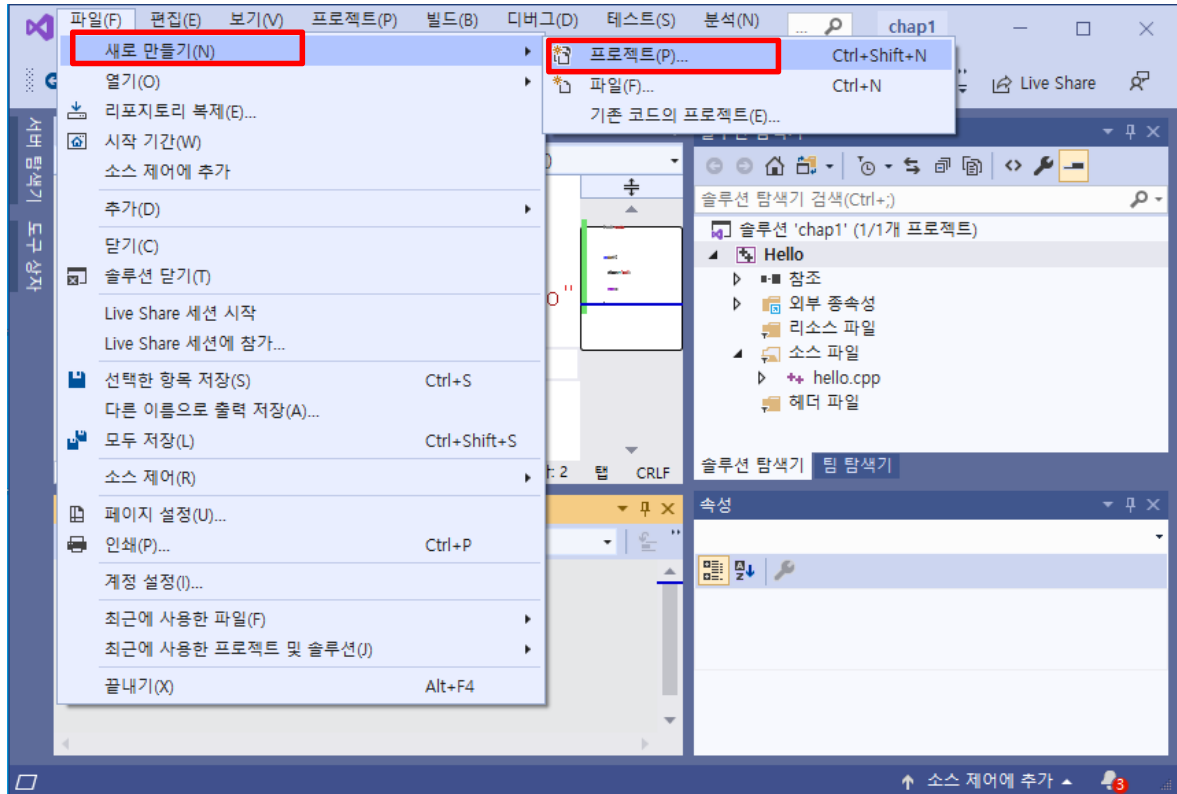
# 실습1

54

## □ 교재 47P Open Challenge

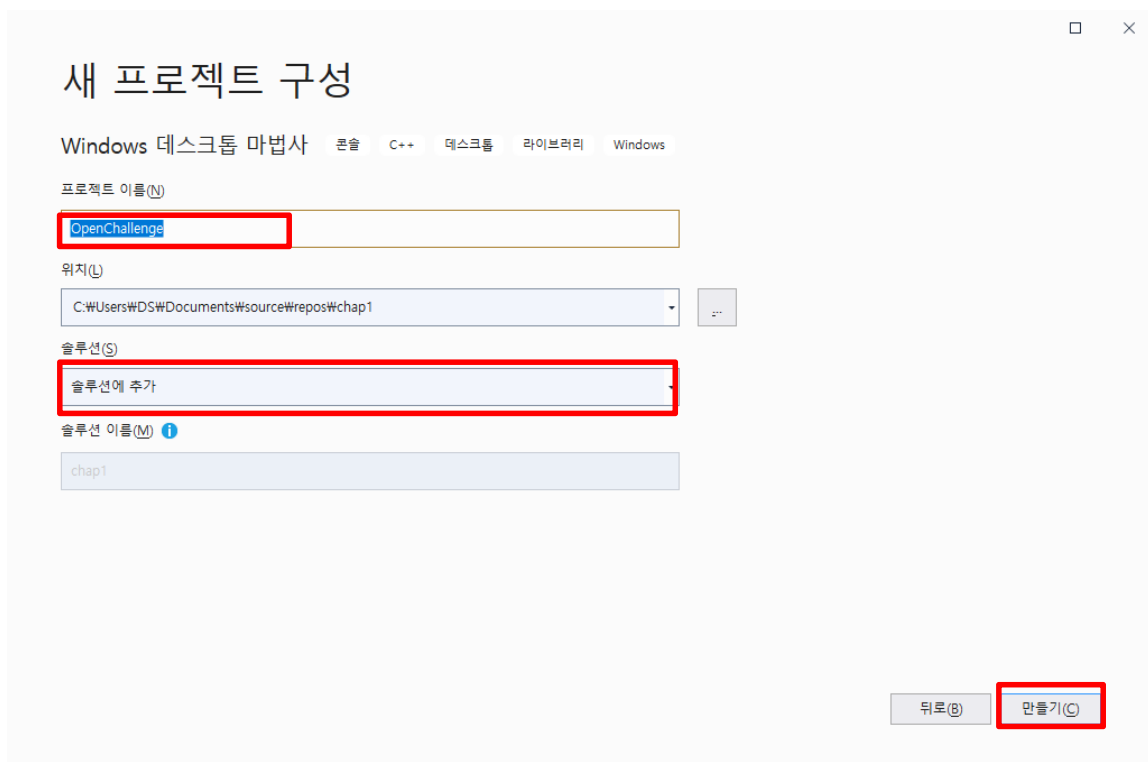
- 기본 C++ 프로그램 작성
- chap1 솔루션에 프로젝트 추가
- 프로젝트명 : OpenChallenge
- 소스파일명 : add.cpp

## 기존 솔루션에 새 프로젝트 추가(1/3)



55

## 기존 솔루션에 새 프로젝트 추가(2/3)



56

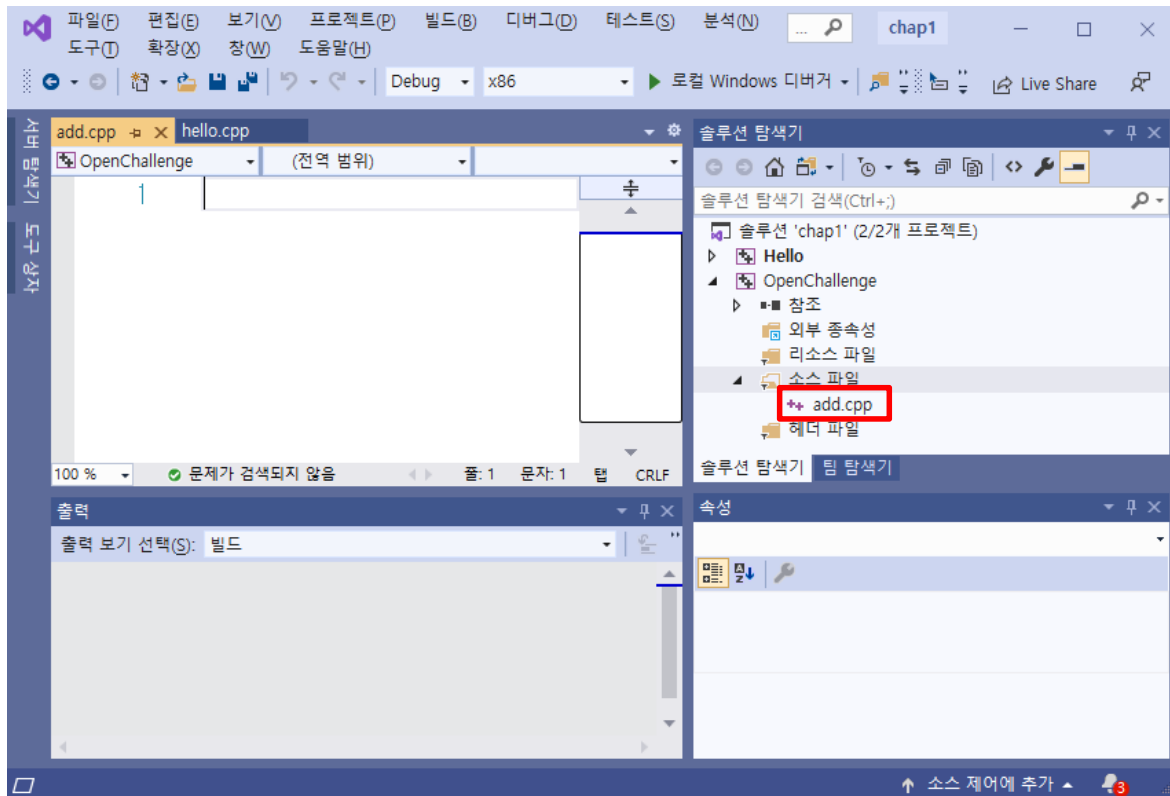
## 기존 솔루션에 새 프로젝트 추가(3/3)



## 파일 추가(1/2)

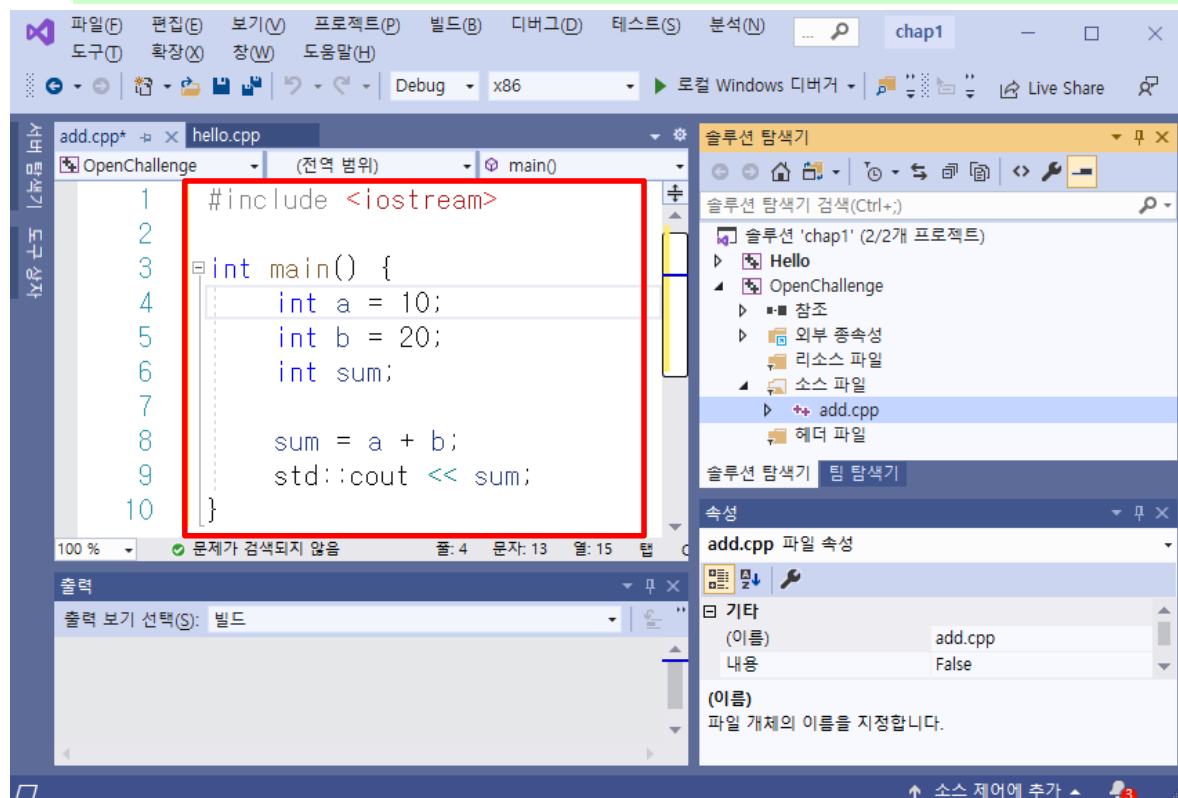


## 파일 추가(2/2)



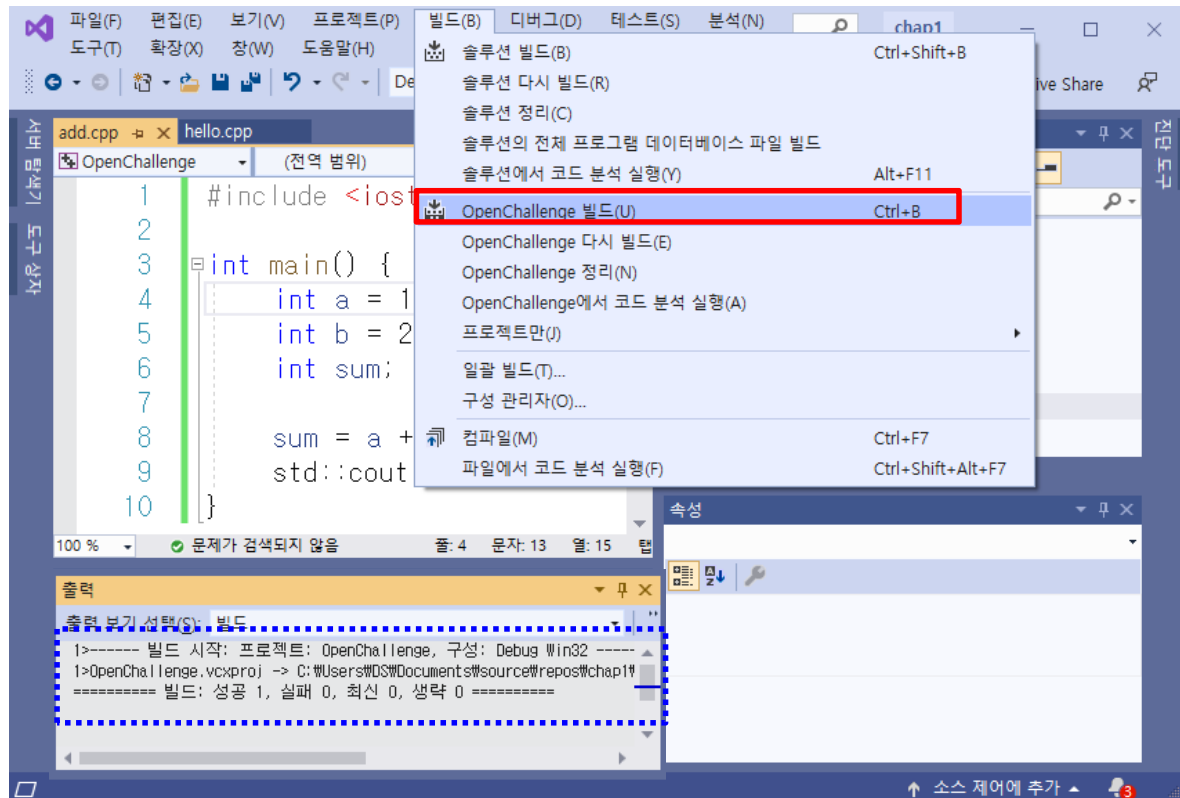
59

## add.cpp 프로그램 작성



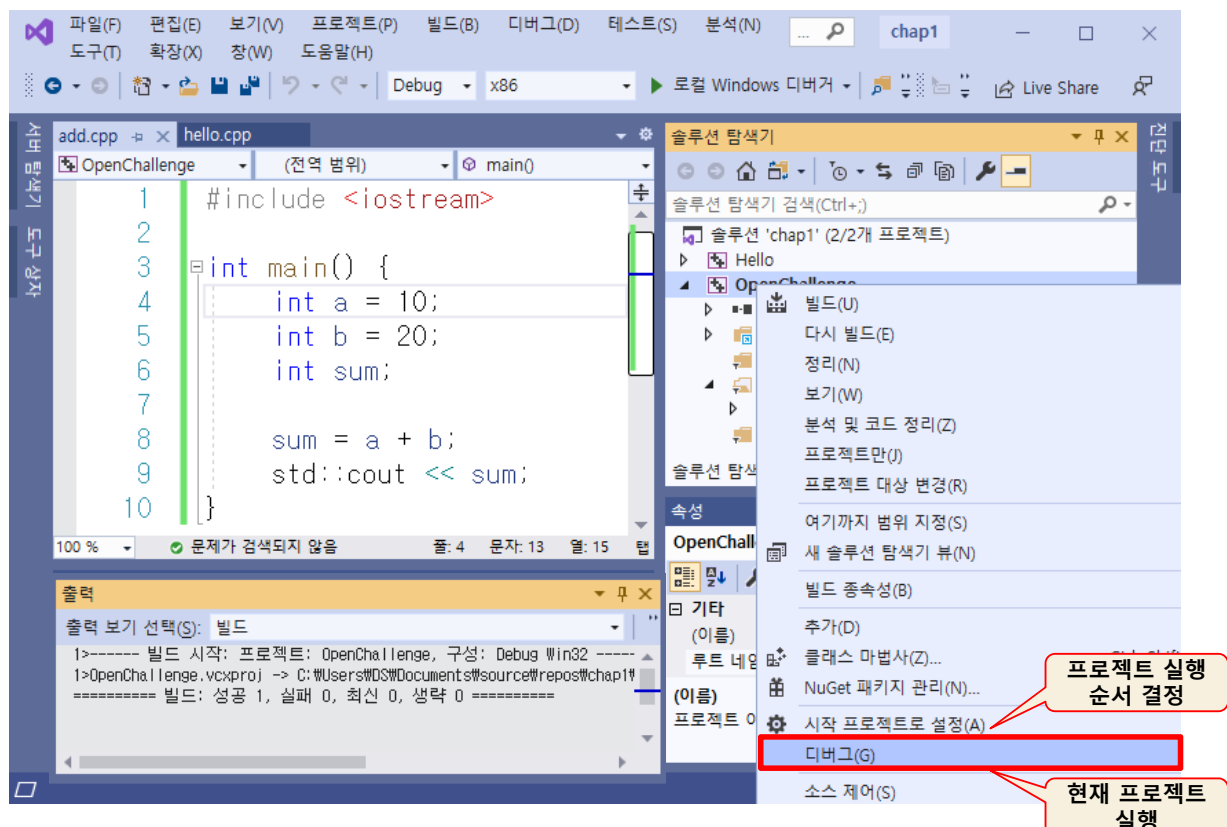
60

## add.cpp 프로그램 실행



61

## 한 솔루션 여러 프로젝트인 경우



## □ 교재 52P, 실습문제3

- ▣ 1에서 10까지 더하여 결과를 출력하는 프로그램
- ▣ chap1 솔루션에 프로젝트 추가
- ▣ 프로젝트명 : Ex1-3
- ▣ 소스파일명 : sum1to10.cpp