

Ch01. C++ 시작



학습 목표

1. 소프트웨어의 중요성을 안다.
2. C++ 언어의 역사를 이해한다.
3. C++ 언어의 특징을 이해한다
4. C++ 프로그램의 개발 과정을 안다.
5. C++ 표준 라이브러리에 대해 안다.
6. Visual Studio를 이용하여 C++ 프로그램을 개발하는 과정을 안다.

세상을 먹어 치우는 소프트웨어

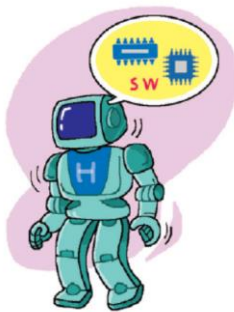
3

□ 소프트웨어 기업의 세상

Software is eating the world.

eBay, Facebook, Groupon, Skype, Twitter, Android, Netflix, Google, Apple, Samsung

- 4차 산업의 핵심에 소프트웨어가 있다
 - ▣ 무인 자동차, AI(구글의 알파고, IBM 왓슨), IoT



프로그래밍과 프로그래밍 언어

4

□ 프로그래밍 언어

▣ 기계어(machine language)

- 0, 1의 이진수로 구성된 언어
- 컴퓨터의 CPU는 본질적으로 기계어만 처리 가능

▣ 어셈블리어

- 기계어의 명령을 ADD, SUB, MOVE 등과 같은 상징적인 니모닉 기호(mnemonic symbol)로 일대일 대응시킨 언어
- 어셈블러 : 어셈블리어 프로그램을 기계어 코드로 변환

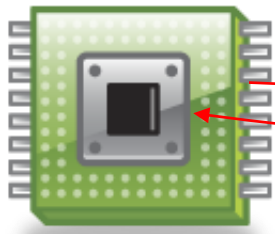
▣ 고급언어

- 사람이 이해하기 쉽고 복잡한 작업, 자료 구조, 알고리즘을 표현하기 위해 고안된 언어
- Pascal, Basic, C/C++ , Java, C#
- 컴파일러 : 고급 언어로 작성된 프로그램을 기계어 코드로 변환

사람과 컴퓨터, 기계어와 고급 언어

5

$35 + 56 = ?$



```
01010000
00101001
11101011
```

기계어

```
LOAD AX 35
LOAD BX 56
ADD AX BX
```

어셈블리어

```
int i;
i = 35 + 56;
```

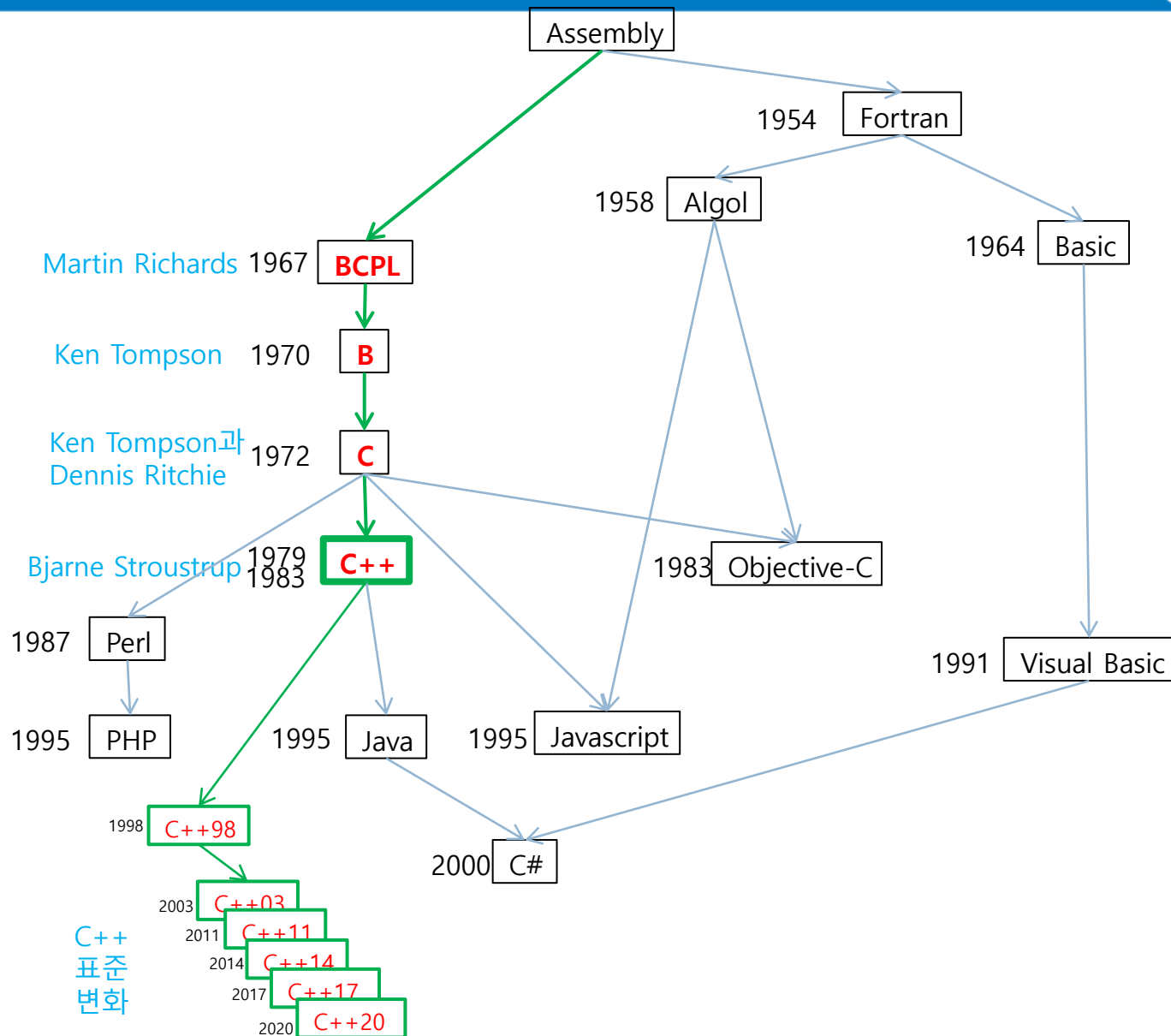
C++



어셈블

컴파일

프로그래밍 언어의 진화와 C++의 기원



표준 C++ 프로그램의 중요성

7

□ 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

8

표준 C++ 규칙에 따라
작성된 C++ 프로그램

```
#include <iostream>
int main() {
    std::cout << "Hello";
    return 0;
}
```

모든 C++
컴파일러
에 의해
컴파일

볼랜드 C++
컴파일러

비주얼 C++
컴파일러

GNU C++
컴파일러

실행
파일

실행
파일

실행
파일

컴퓨터

표준 C++ 규칙에 따라
작성되지 않는 비주얼 C++ 프로그램

```
#include <iostream>
int _cdecl main() {
    std::cout << "Hello";
    return 0;
}
```

비주얼
C++ 전용
키워드

볼랜드 C++
컴파일러

비주얼 C++
컴파일러

GNU C++
컴파일러

~~실행
파일~~

실행
파일

~~실행
파일~~

컴퓨터

C++ 언어의 주요한 설계 목적

9

- 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
- 객체 지향 개념 도입
 - ▣ 캡슐화, 상속, 다형성
 - ▣ 소프트웨어의 재사용을 통해 생산성 향상
 - ▣ 복잡하고 큰 규모의 소프트웨어의 작성, 관리, 유지보수 용이
- 엄격한 타입 체크
 - ▣ 실행 시간 오류의 가능성을 줄임
 - ▣ 디버깅 편리
- 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

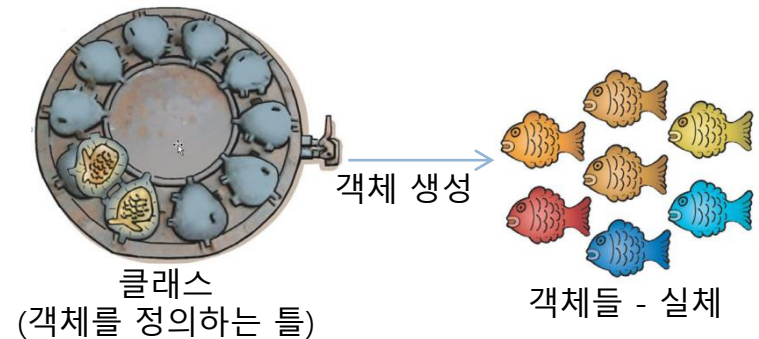
10

- ▣ 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 (Operator override)
 - 기존 C++ 연산자에 새로운 연산 정의
- ▣ 제네릭 함수와 클래스
 - 데이터 타입에 의존하지 않고 일반화시킨 함수나 클래스 작성 가능

C++ 객체 지향 특성 - 캡슐화

11

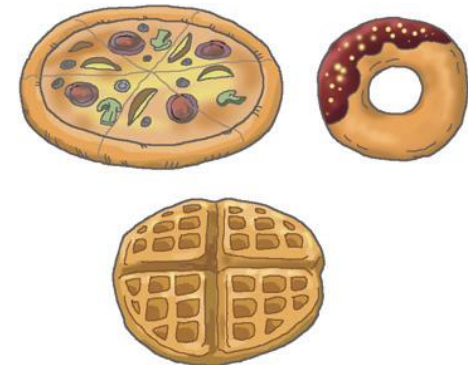
- 캡슐화(Encapsulation)
 - ▣ 데이터를 캡슐로 싸서 외부의 접근으로부터 보호
 - ▣ C++에서 클래스(class 키워드)로 캡슐 표현
- 클래스와 객체
 - ▣ 클래스 - 객체를 만드는 틀
 - ▣ 객체 - 클래스라는 틀에서 생겨난 실체
 - ▣ 객체(object), 실체(instance)는 같은 뜻



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

멤버들

원을 추상화한 Circle 클래스



원 객체들(실체)

C++ 객체 지향 특성 - 상속성

12

- 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



```

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

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

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

Phone을 상속받는다.

MobilePhone을 상속받는다.

C++로 상속 선언



전화기



휴대 전화기



음악 기능
전화기

C++ 객체 지향 특성 - 다형성

13

□ 다형성(Polymorphism)

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

$2 + 3 \rightarrow 5$
 "남자" + "여자" \rightarrow "남자여자"
 redColor 객체 + blueColor 객체 \rightarrow 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

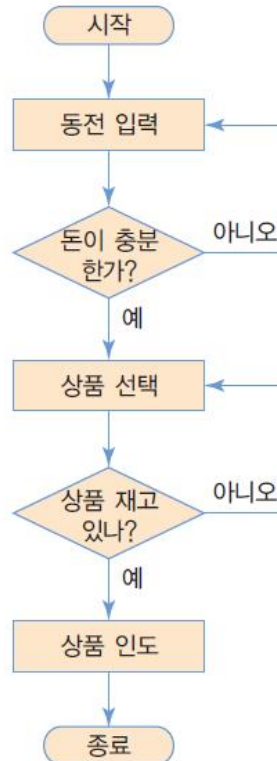
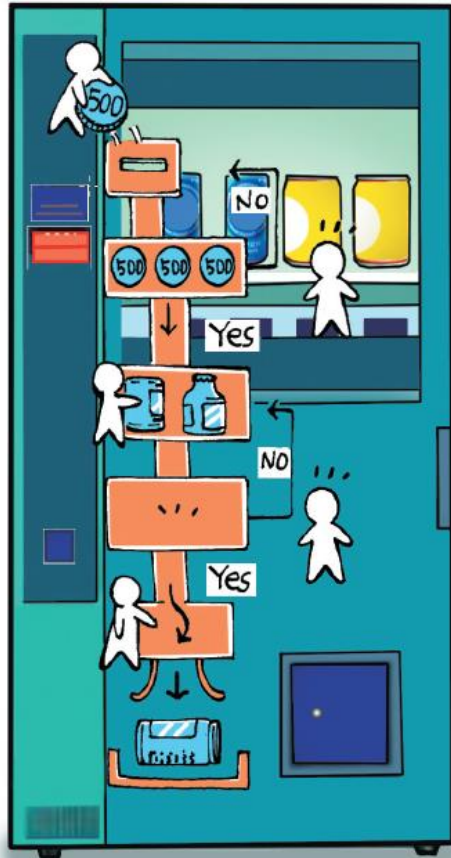
14

- 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

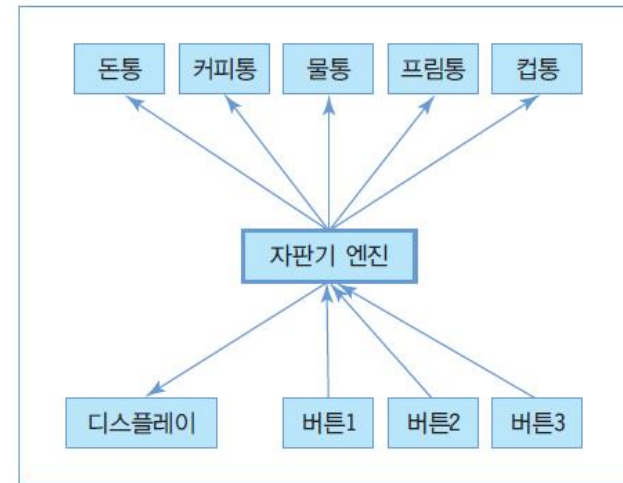
절차 지향 프로그래밍과 객체 지향 프로그래밍

15



(a) 절차 지향적 프로그래밍으로 구현할 때의 흐름도

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



(b) 객체 지향적 프로그래밍으로 구현할 때의 객체 관계도

- 객체들을 정의하고, 객체들의 상호 관계, 상호 작용으로 구현

C++와 제네릭 프로그래밍

16

- 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

17

□ C++ 언어는 C 언어와의 호환성 추구

□ 장점

- 기존에 개발된 C 프로그램 코드 활용

□ 단점

■ 캡슐화의 원칙이 무너짐

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

C++ 프로그램 개발 과정

18



C++ 소스 프로그램 작성

```
#include <iostream>
int main() {
    std::cout << "Hello";
    return 0;
}
```

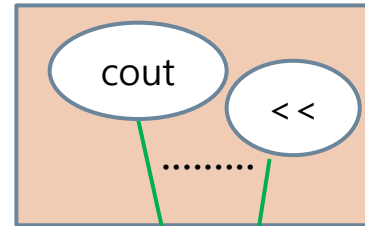
소스 파일
(hello.cpp)

컴파일

```
_main,12#
$<<01010
00000111
_Hello001
```

목적 파일
(hello.obj)

C++ 라이브러리



링킹

```
01010000
01000101
01001111
01011010
10100101
11010101
```

실행 파일
(hello.exe)

실행



오류 발생

디버깅

오류 수정



C++ 프로그램 작성 및 컴파일

19

□ 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

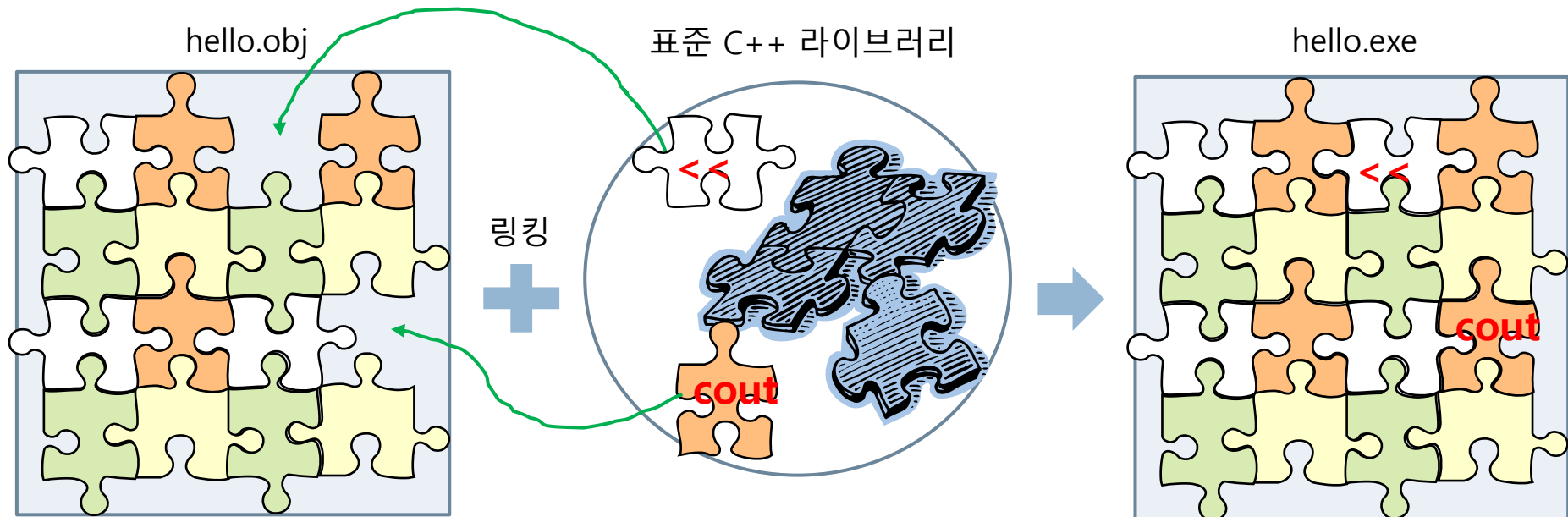
링킹

21

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



프로그램 실행과 디버깅

22

- 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

23

- C++ 표준 라이브러리는 3 개의 그룹으로 구분
 - ▣ C 라이브러리
 - 기존 C 표준 라이브러리를 수용, C++에서 사용할 수 있게 한 함수들
 - 이름이 c로 시작하는 헤더 파일에 선언됨
 - ▣ C++ 입출력 라이브러리 (<https://cplusplus.com/reference/>)
 - 콘솔 및 파일 입출력을 위한 라이브러리
 - ▣ C++ STL 라이브러리 (<https://learn.microsoft.com/ko-kr/cpp/standard-library/>)
 - 제네릭 프로그래밍을 지원하기 위해 템플릿 라이브러리

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	

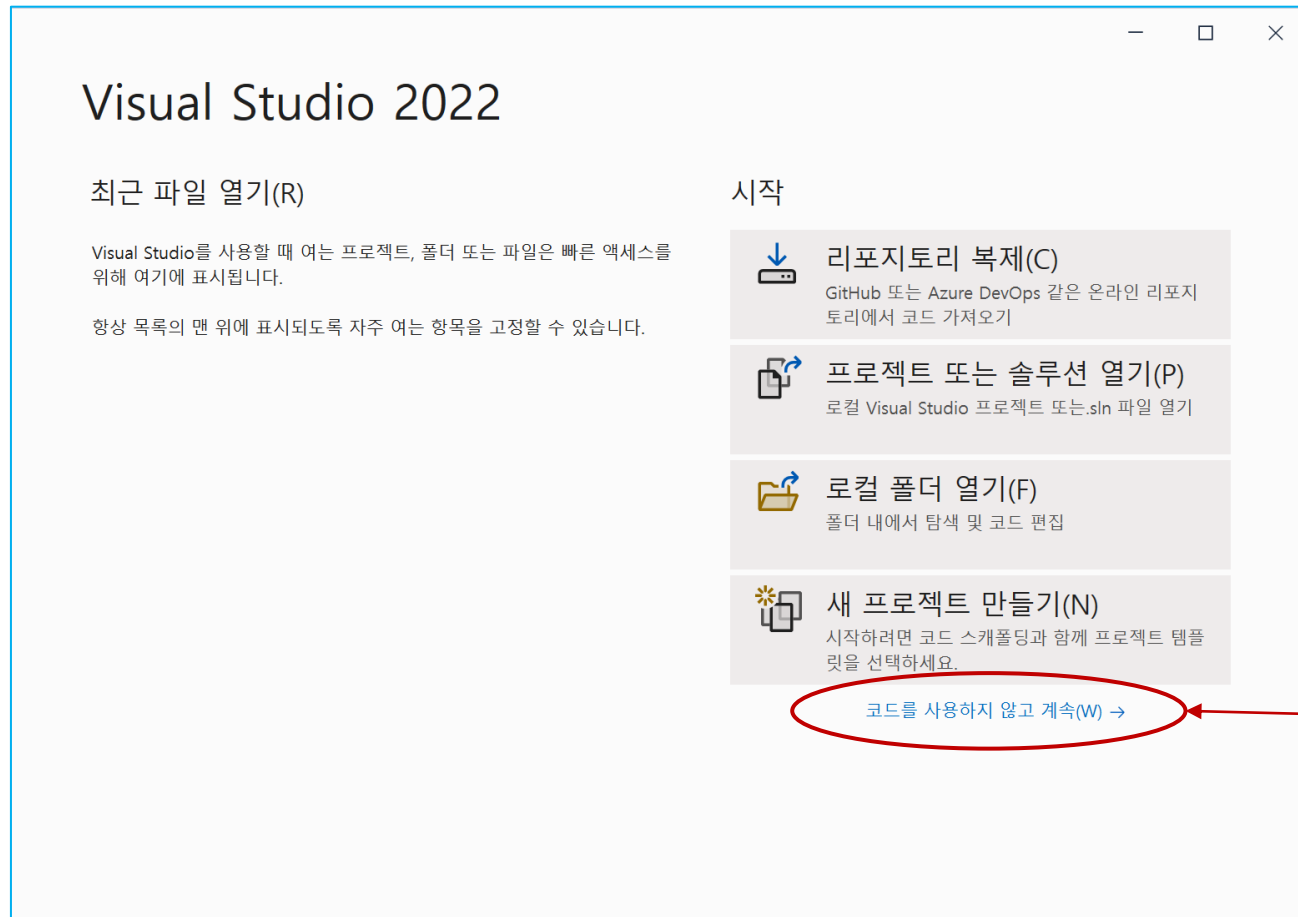
C 라이브러리

STL 라이브러리

C++ 입출력 라이브러리

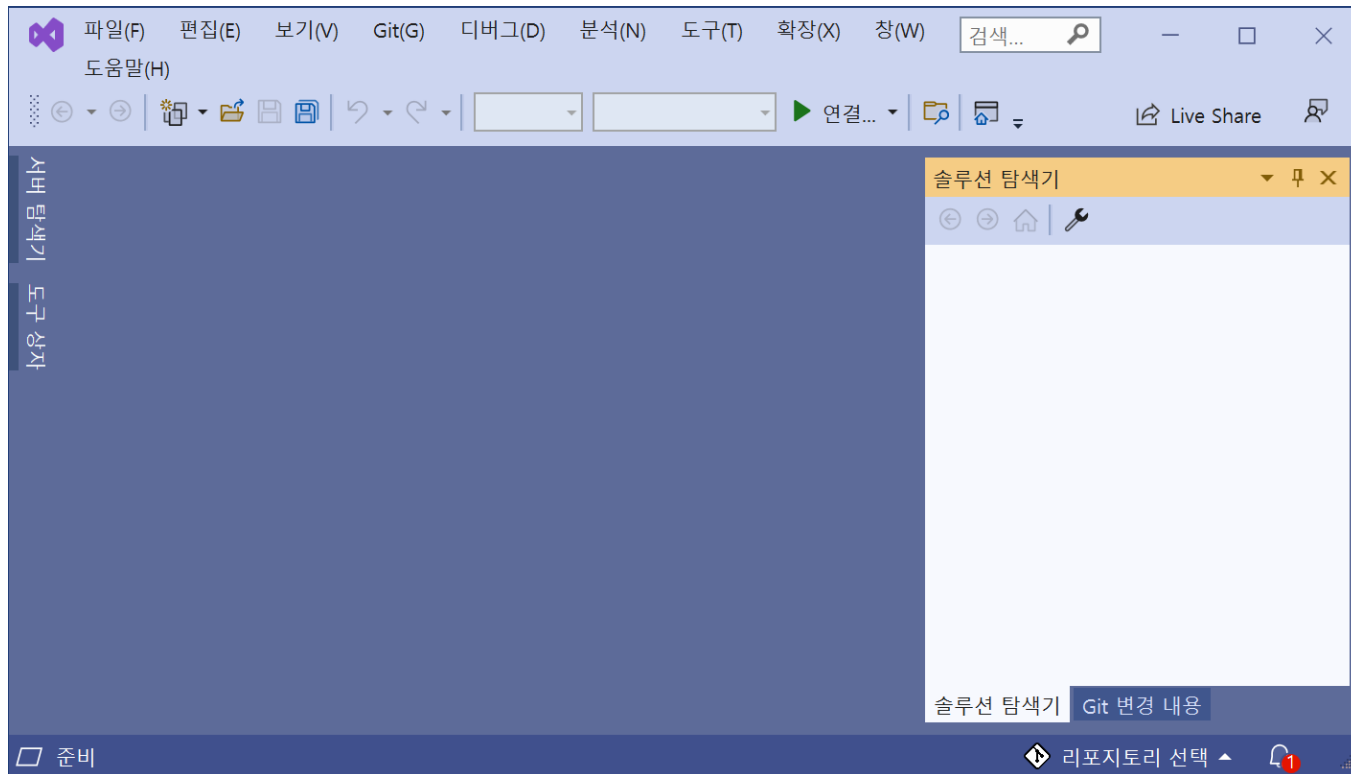
*〈new〉 헤더 파일은 STL에 포함되지 않는 기타 기능을 구현함

Visual Studio 시작



클릭하면
다음 슬라이드로

Visual Studio 스크린



최근 프로젝트 템플릿(R)

콘솔 앱 C#

모두 지우기(

모든 프로젝트 형식(T)

콘솔

콘솔

콘솔

라이브러리

26

새 프로젝트 구성

새 프로젝트 구성

빈 프로젝트

C++

Windows

콘솔

프로젝트 이름(J)

C:\WC++\Wchap1\Hello

Hello

C:\WC++\Wchap1\Hello
폴더가 생긴다.

위치(L)

C:\WC++

솔루션 위치

...

솔루션 이름(M) ⓘ

chap1

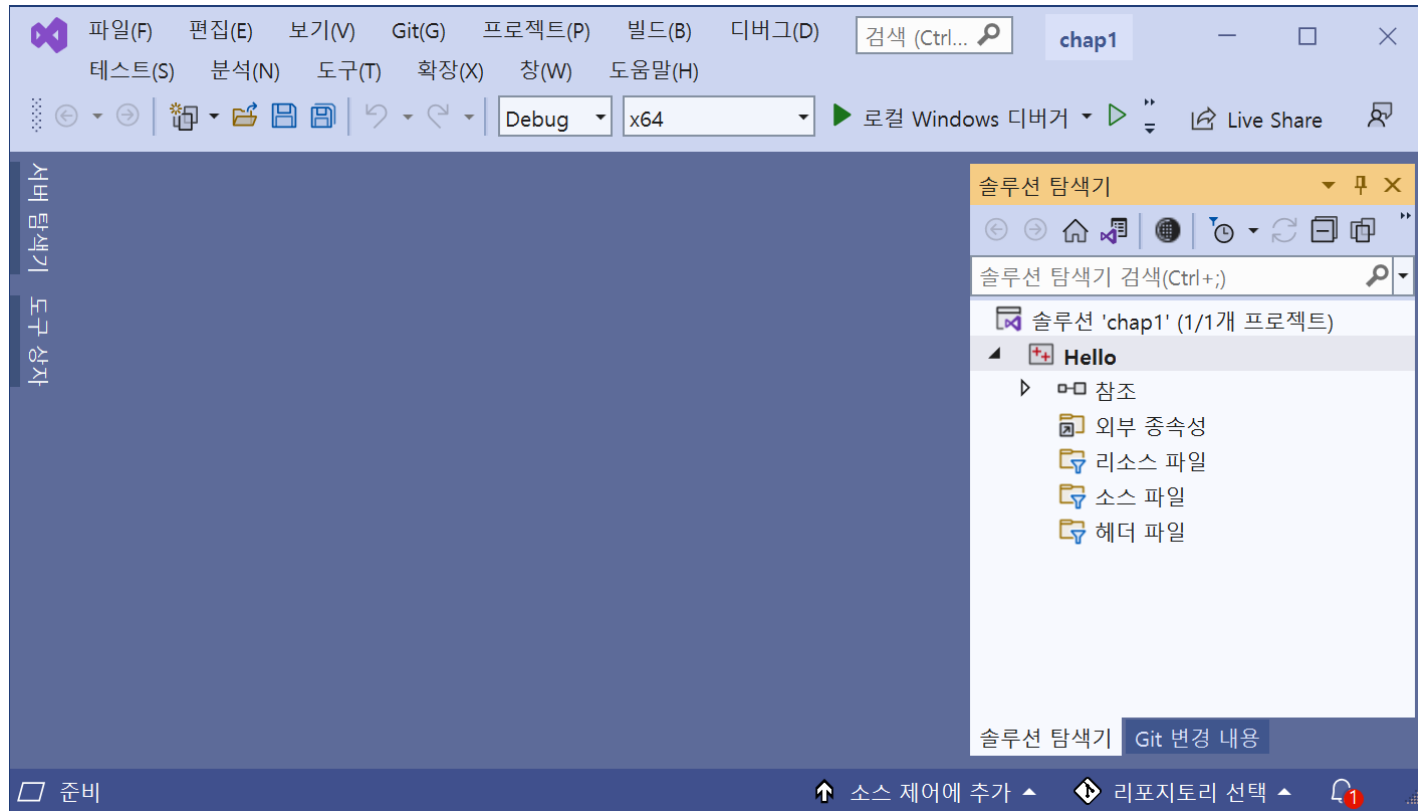
C:\WC++\Wchap1
폴더를 생성한다.

☐ 솔루션 및 프로젝트를

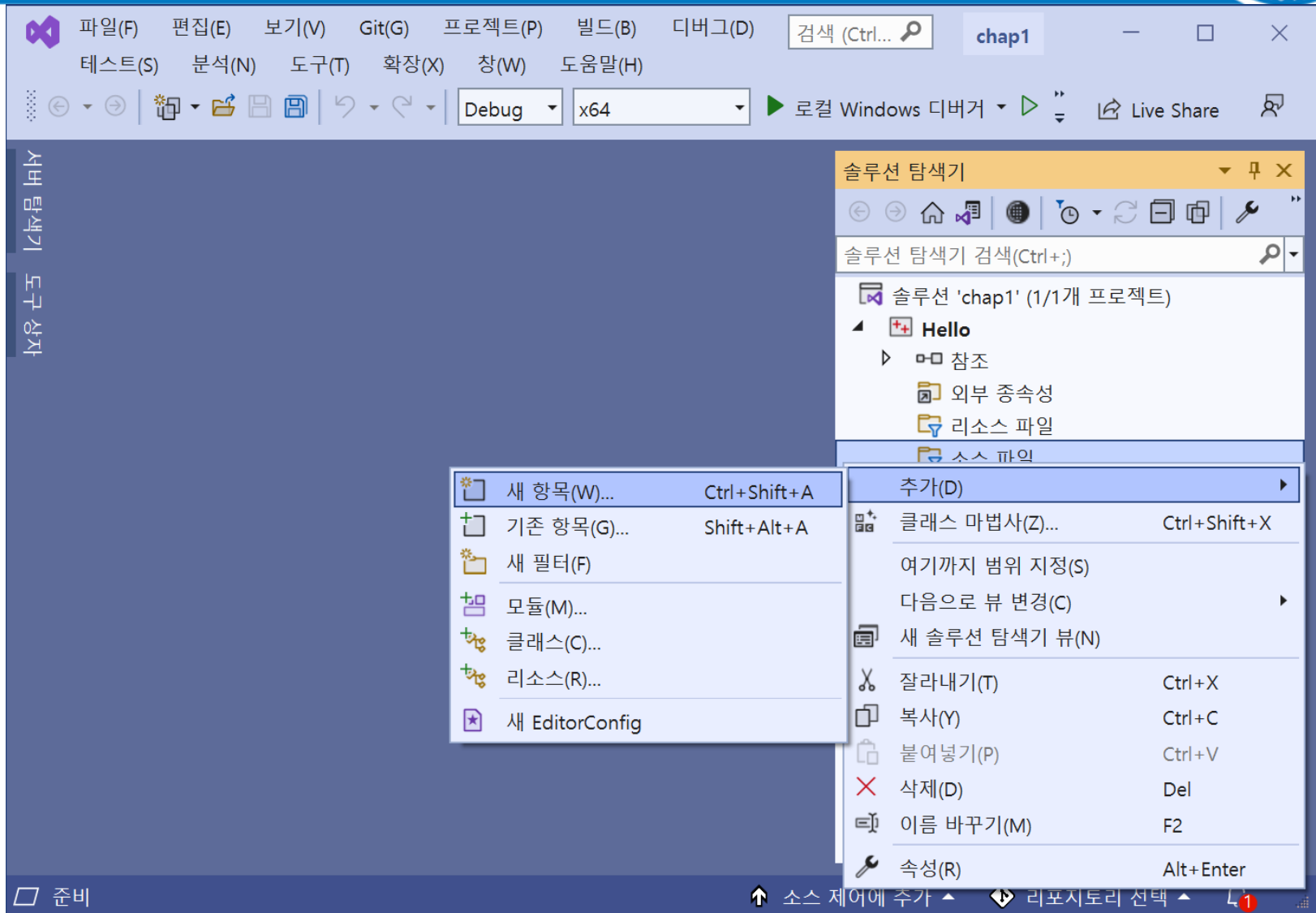
뒤로(B)

만들기(C)

Hello 프로젝트 생성 후



새 항목 만드는 메뉴 선택



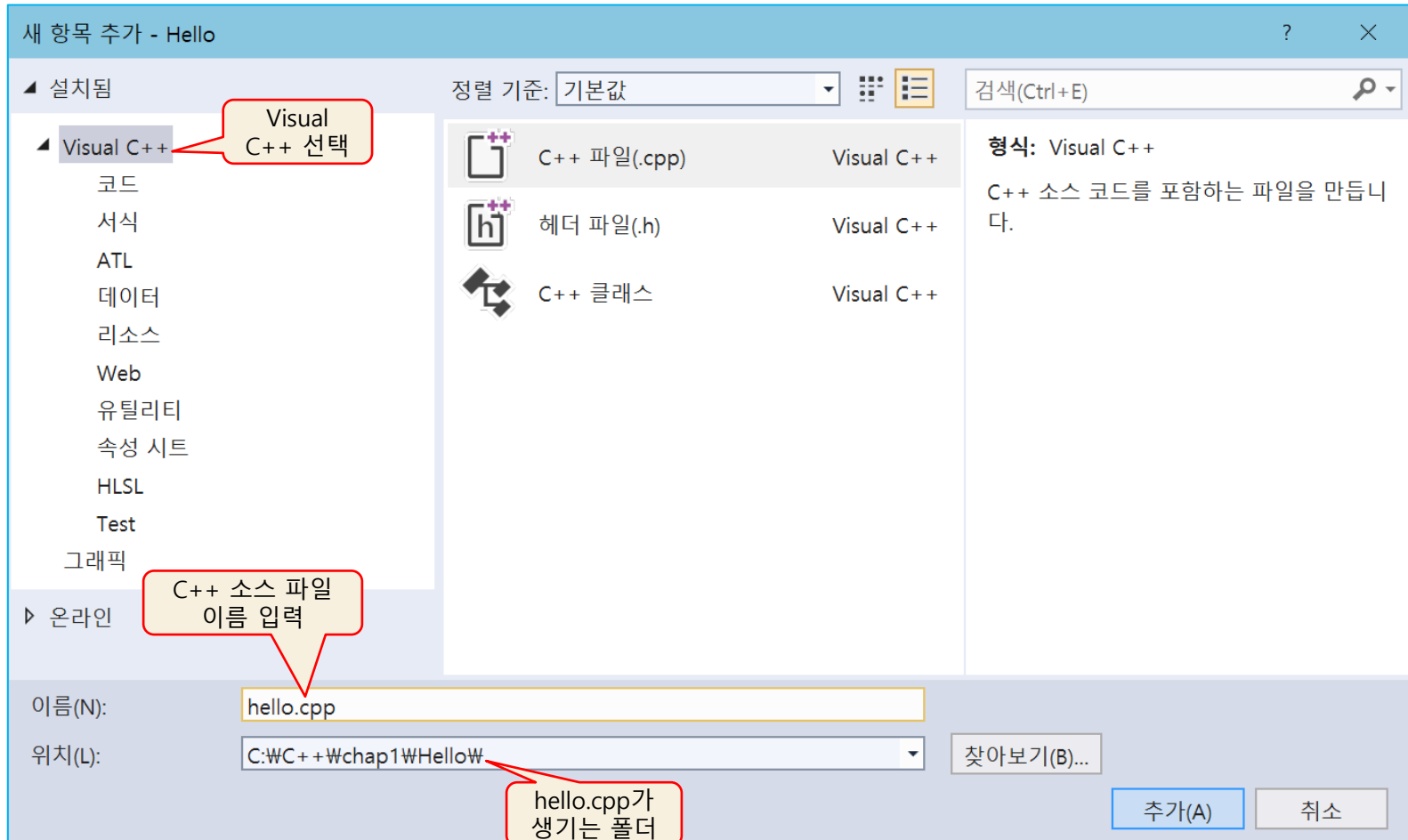
The screenshot shows the Visual Studio IDE interface. The 'File' menu is open, and the 'Add' option is selected, which has opened a submenu. The submenu contains the following items:

- 새 항목(W)... (Ctrl+Shift+A)
- 기존 항목(G)... (Shift+Alt+A)
- 새 필터(F)
- 모듈(M)...
- 클래스(C)...
- 리소스(R)...
- 새 EditorConfig

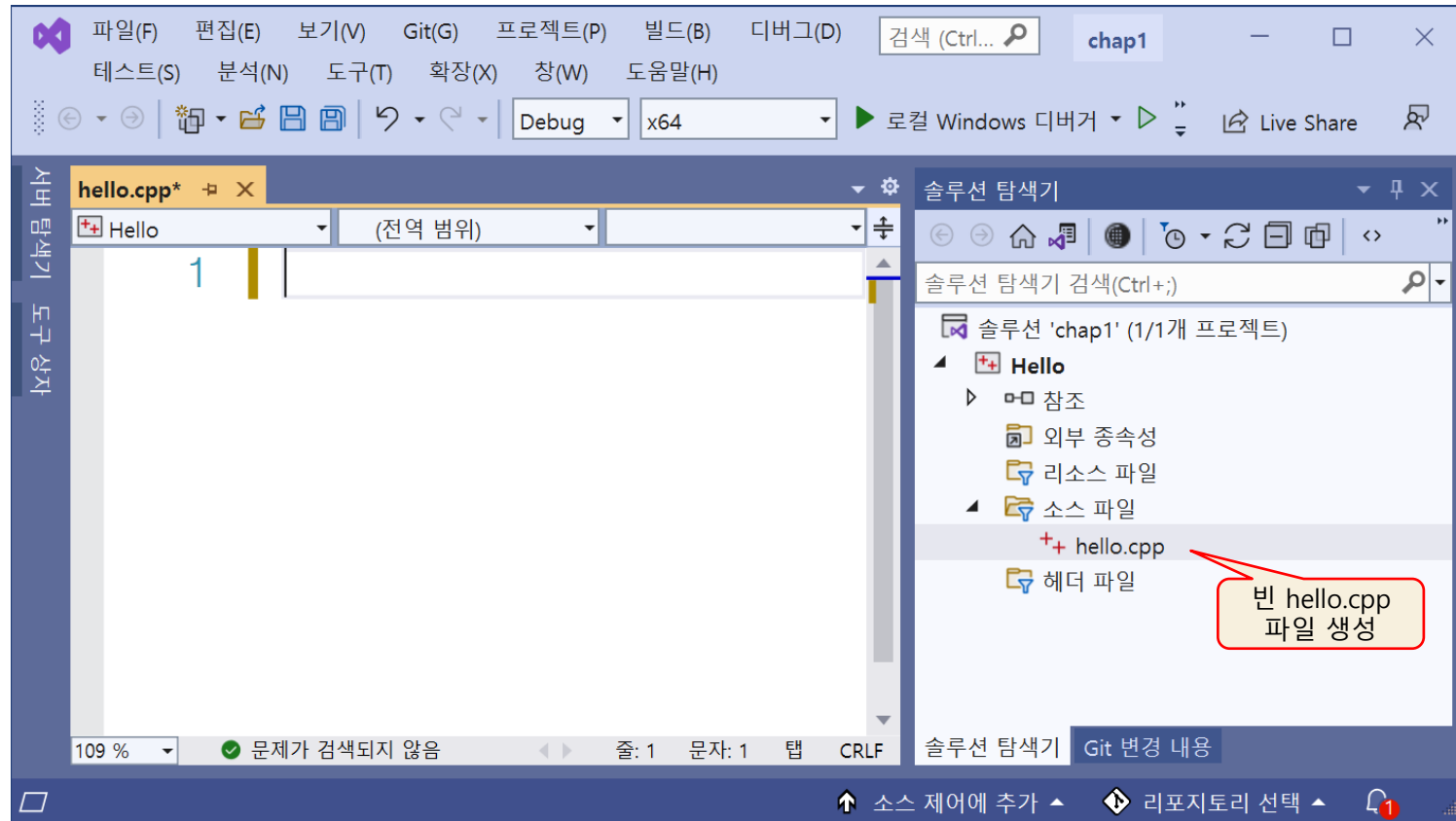
The 'Add' submenu is also open, showing the following items:

- 추가(D)
- 클래스 마법사(Z)... (Ctrl+Shift+X)
- 여기까지 범위 지정(S)
- 다음으로 뷰 변경(C)
- 새 솔루션 탐색기 뷰(N)
- 잘라내기(T) (Ctrl+X)
- 복사(Y) (Ctrl+C)
- 붙여넣기(P) (Ctrl+V)
- 삭제(D) (Del)
- 이름 바꾸기(M) (F2)
- 속성(R) (Alt+Enter)

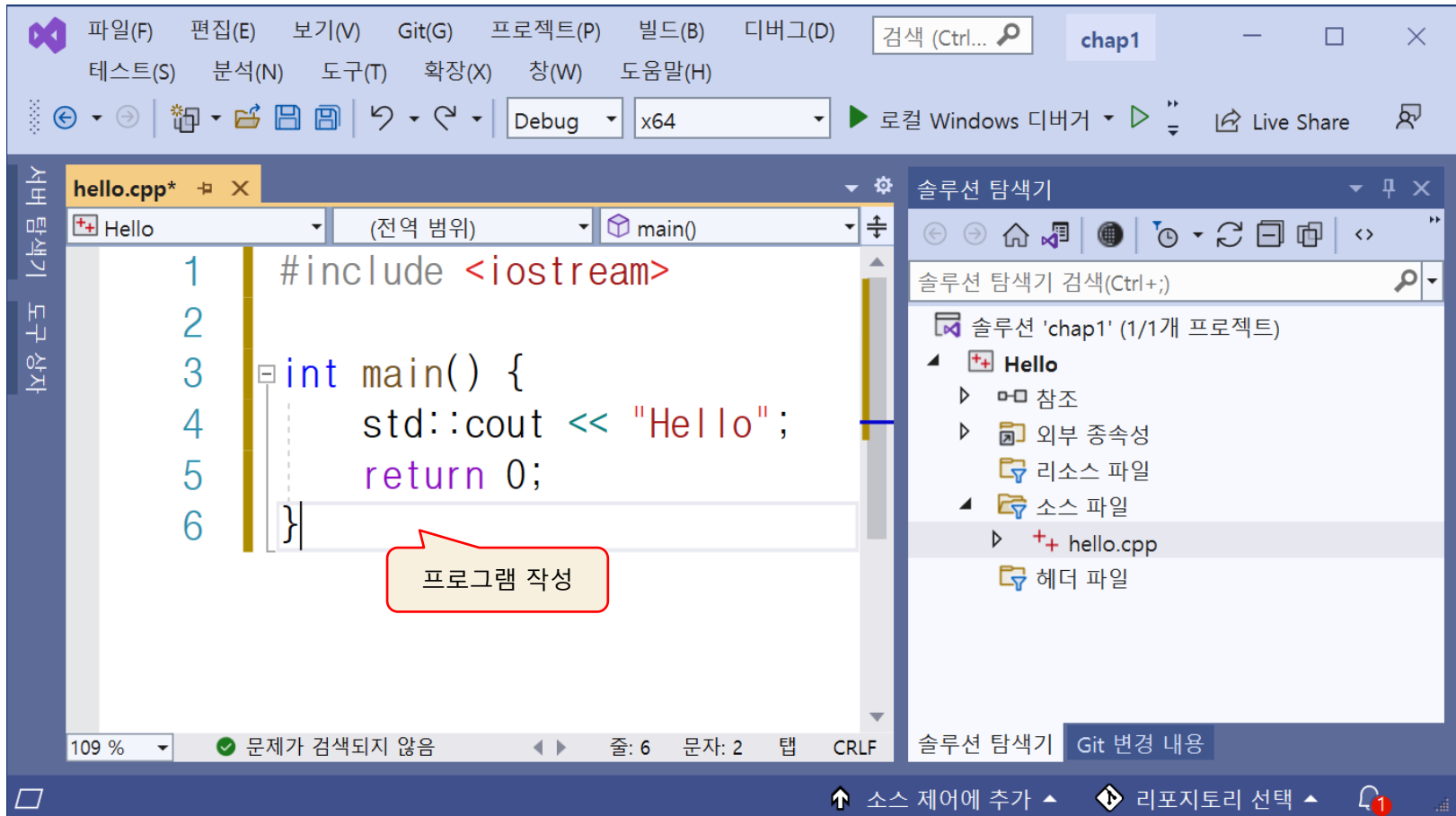
hello.cpp 소스 파일 생성



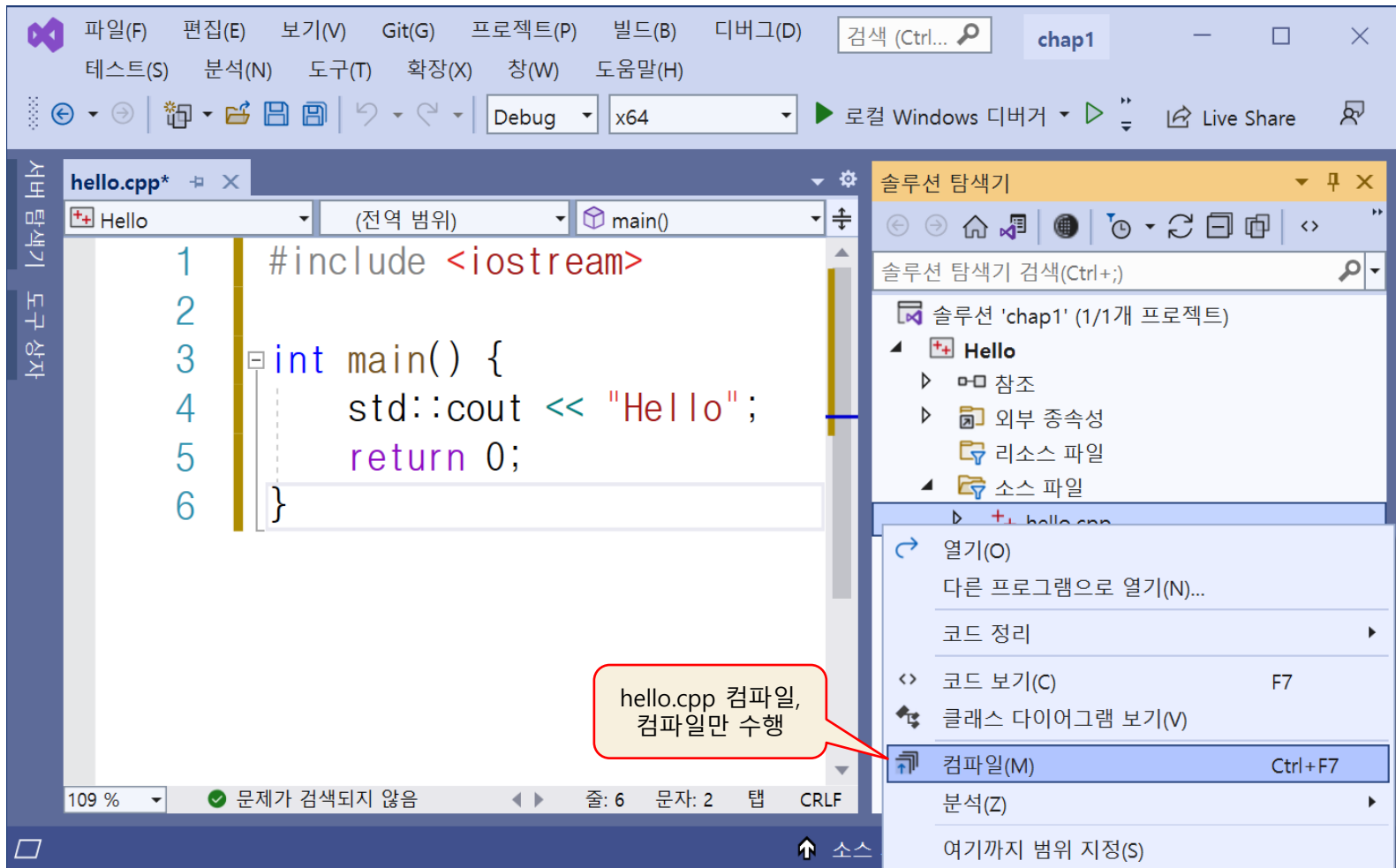
hello.cpp 파일이 생성된 초기 모습



hello.cpp 작성



솔루션 탐색기에서 컴파일 메뉴 선택



The screenshot shows the Visual Studio IDE with the 'hello.cpp' file open in the editor. The Solution Explorer on the right shows the project structure. The 'Compile' menu is highlighted, and a red callout box points to it with the text 'hello.cpp 컴파일, 컴파일만 수행'.

File(F) 편집(E) 보기(V) Git(G) 프로젝트(P) 빌드(B) 디버그(D) 검색 (Ctrl+...) chap1

테스트(S) 분석(N) 도구(T) 확장(X) 창(W) 도움말(H)

Debug x64 로컬 Windows 디버거 Live Share

hello.cpp* (전역 범위) main()

```

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

솔루션 탐색기

솔루션 탐색기 검색(Ctrl+;)

솔루션 'chap1' (1/1개 프로젝트)

- Hello
 - 참조
 - 외부 종속성
 - 리소스 파일
 - 소스 파일
 - hello.cpp
 - 열기(O)
 - 다른 프로그램으로 열기(N)...
 - 코드 정리
 - 코드 보기(C) F7
 - 클래스 다이어그램 보기(V)
 - 컴파일(M) Ctrl+F7**
 - 분석(Z)
 - 여기까지 범위 지정(S)

109 % 문제 없음 줄: 6 문자: 2 탭 CRLF

hello.cpp 컴파일, 컴파일만 수행

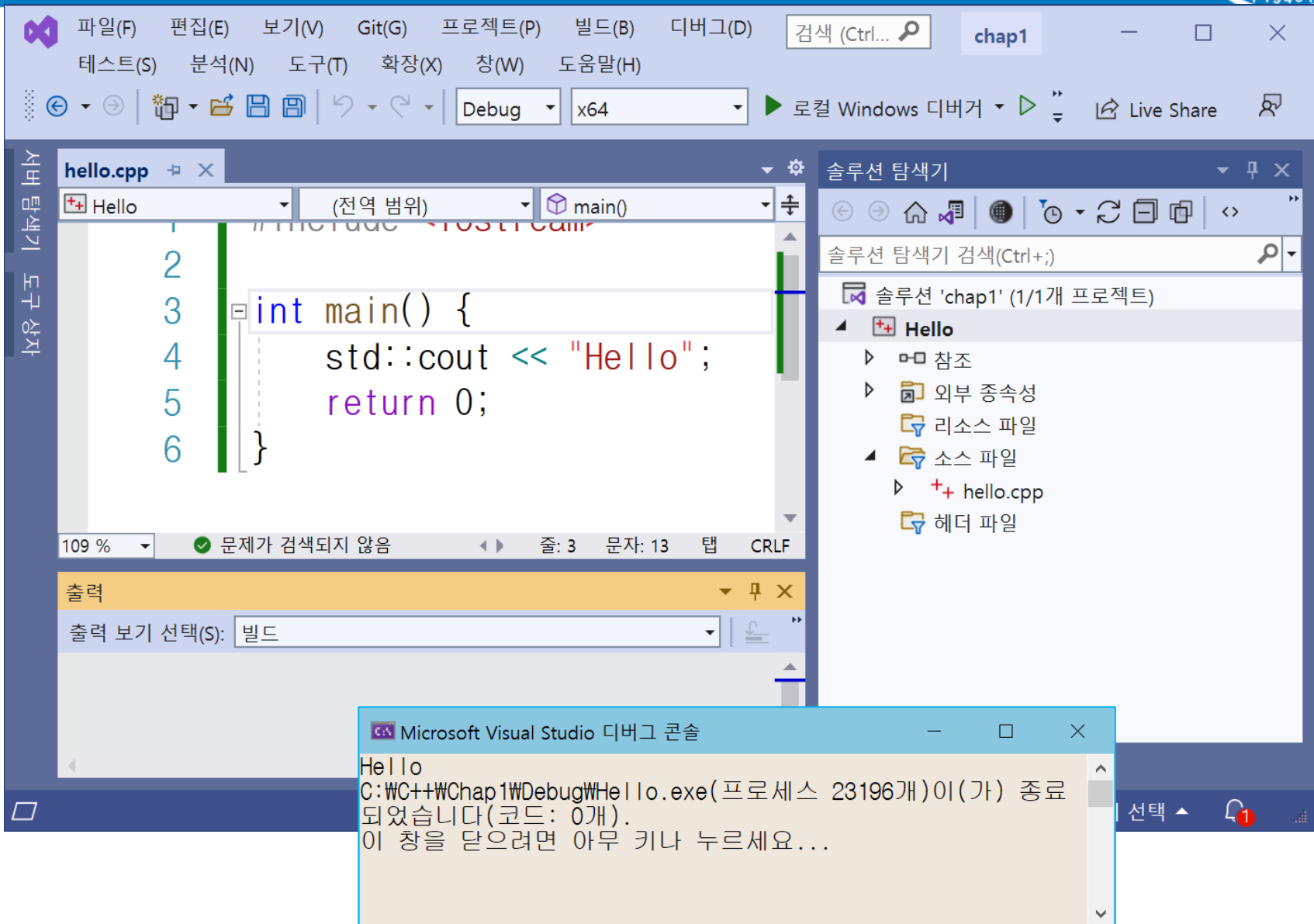
Hello 프로젝트의 빌드로 Hello.exe 생성



The screenshot shows the Visual Studio IDE with the 'Hello' project open. The 'Build' menu is open, and the 'Build (U)' option is highlighted. A red callout box points to this option with the text: "빌드 메뉴를 선택하면, 컴파일과 링크 후 Hello.exe 생성". The output window at the bottom shows the build process:

```
출력 보기 선택(S): 빌드
1>----- 빌드 시작: 프로젝트: Hello, 구성: Debug x64
1>hello.cpp
1>Hello.vcxproj -> C:\WC++\wchap1\w64\Debug\Hello.exe
===== 빌드: 성공 1, 실패 0, 최신 0, 생략 0 =====
```

Hello 프로젝트가 실행되는 화면





Thank You

