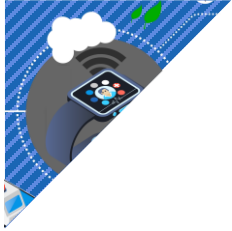


프로그래밍 언어 및 실습

2023년도 1학기 2주차





C++의 시작

C++의 개요





■ 세상을 먹어 치우는 소프트웨어

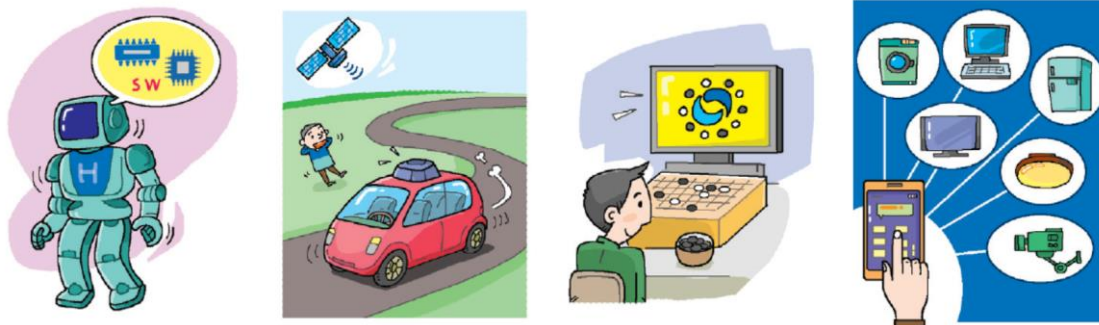
▶ 소프트웨어 기업의 세상

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, IBM의 왓슨), IoT





■ 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

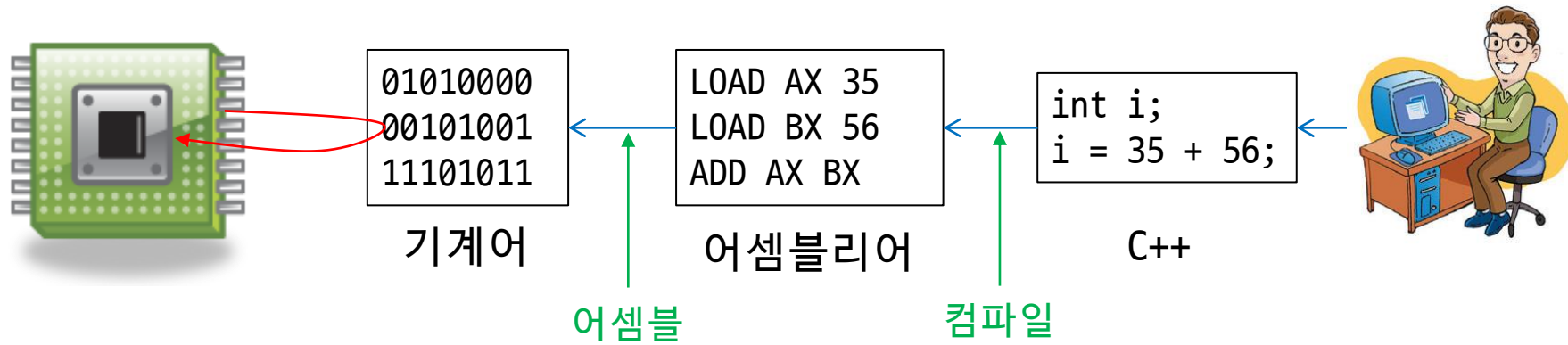
- ▶ 사람이 이해하기 쉽고 복잡한 작업, 자료구조, 알고리즘을 표현하기 위해 고안된 언어
- ▶ Pascal, Basic, C/C++, Java, C#
- ▶ 컴파일러 : 고급언어로 작성된 프로그램을 기계어 코드로 변환



프로그래밍 언어

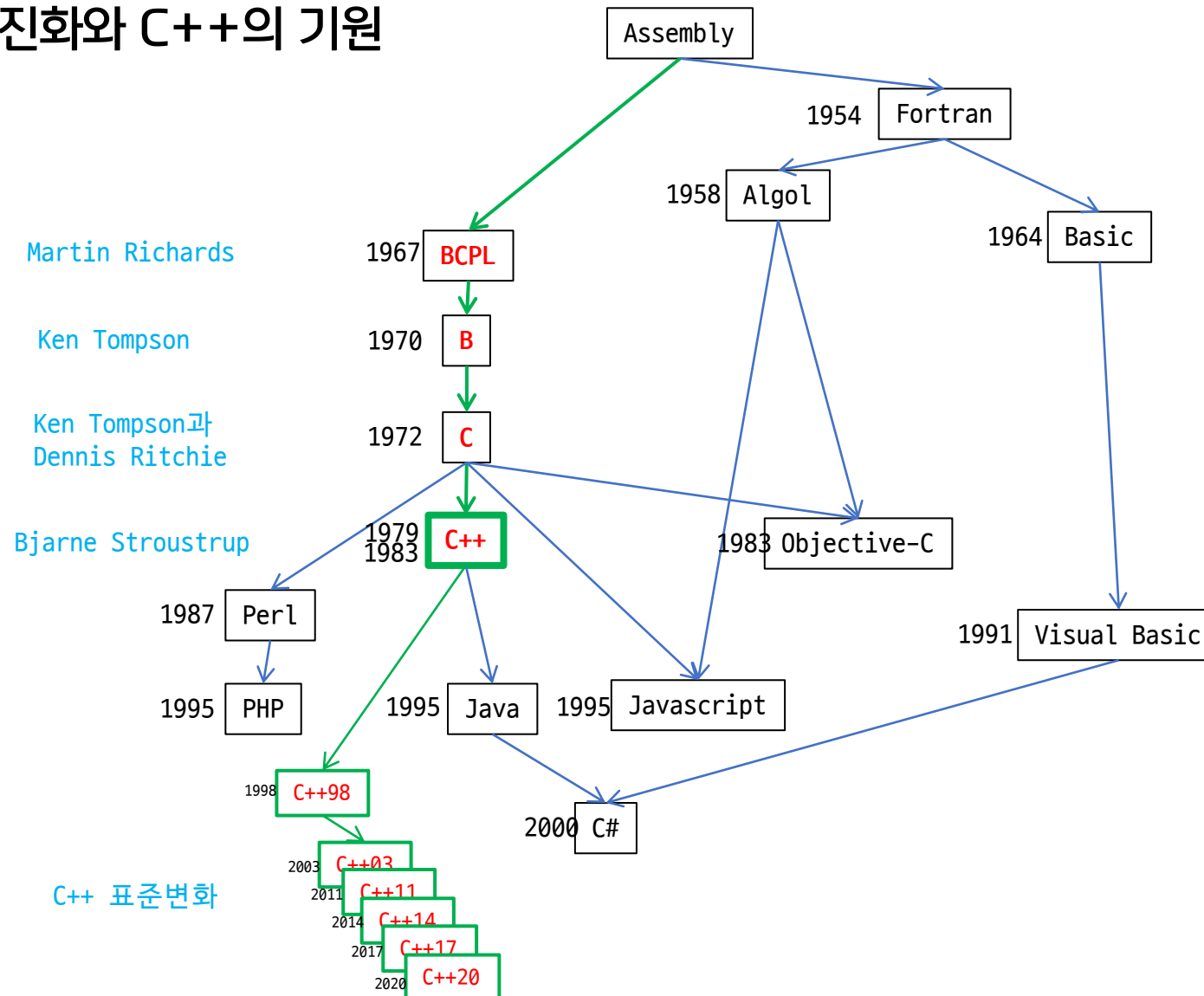
▶ 사람과 컴퓨터, 기계어와 고급 언어

35 + 56 = ?



C++ 언어의 역사

프로그래밍 언어의 진화와 C++의 기원





■ 프로그래밍 언어의 진화와 C++의 기원

▶ 프로그래밍 언어의 진화

- ▶ 1950년대 부터 어셈블리어의 한계를 극복한 고급 언어들이 개발됨.
- ▶ 1954년 Fortran이 개발되고, 1967년 운영체제나 컴파일러와 같은 시스템 소프트웨어(system software)를 작성하기 위한 용도로 BCPL이라는 언어가 Martin Richards가 개발함.
- ▶ 1970년 Ken Thompson이 BCPL을 개선한 B언어가 만들어짐.
- ▶ 1972년 Ken Thompson과 Dennis Ritchie가 DEC PDP-11 컴퓨터에서 실행되는 UNIX 운영체제를 작성하기 위해 B언어를 개선한 C언어가 처음 탄생. 이후 지금까지 사랑받고 있다.
 - ▶ C언어의 사용 : 시스템 소프트웨어, 임베디드 시스템, 모바일, 게임, 그래픽 등 광범위한 분야에서 사용되고 있음.



■ 프로그래밍 언어의 진화와 C++의 기원

▶ C++의 탄생 배경

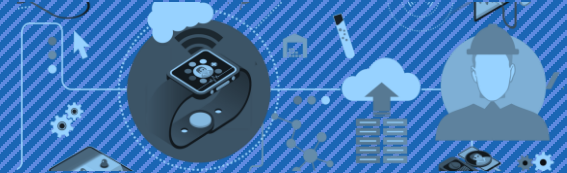
- ▶ 컴퓨터의 속도가 빨라짐에 따라 소프트웨어의 크기도 커지게 되고, C언어로 큰 덩치의 소프트웨어를 개발할 때 프로그램 코드 관리에 어려움이 있음.
- ▶ 1979년 Bell 연구소의 Bjarne Stroustrup가 객체지향 특성(object oriented programming) 및 기능을 추가한 새로운 언어를 만들어 1983년에 C++로 명명함.
- ▶ 객체지향 프로그래밍 기법은 클래스와 상속을 이용하는 소프트웨어의 재사용성을 높이고 소프트웨어의 개발과 관리를 쉽게 한다.
- ▶ C++ 언어는 C 언어의 모든 기능을 가지고 있기 때문에 이미 작성된 C 소스 프로그램을 그대로 사용하거나, 조금만 수정하면 C++ 프로그램으로 재사용할 수 있고, 이미 컴파일된 C 목적코드(object)도 C++ 프로그램에서 링크하여 사용할 수 있다.



■ 프로그래밍 언어의 진화와 C++의 기원

▶ C++의 다른 언어 탄생의 기여

- ▶ 객체 지향 특성의 장점은 다른 언어가 태동하는데 큰 기여를 함.
- ▶ 1995년 C++의 영향을 받은 또 하나의 객체 지향 언어인 Java가 SUN Micro system(현재는 오라클)의 제임스 고슬링에 의해 만들어짐.
- ▶ 2000년에는 MS에서 C++과 Java의 개념을 섞은 C# 언어를 만들었으며, .NET 프레임워크가 설치된 플랫폼 상에서 실행.
- ▶ C++, Java, C#은 모두 C언어를 바탕을 두기 때문에 서로 매우 유사하며, C언어를 잘 알면 이들 언어의 습득은 매우 쉽다.



표준 C++ 프로그램의 중요성

▶ C++ 언어의 표준

- ▶ 1998년 미국 표준원(ANSI; American National Standards Institute)
 - ▶ C++ 언어에 대한 표준 설정
- ▶ ISO/IEC 14882 문서에 작성됨. 유로임(π.π)
- ▶ 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



표준 C++ 프로그램의 중요성

▶ 비 표준 C++ 프로그램

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

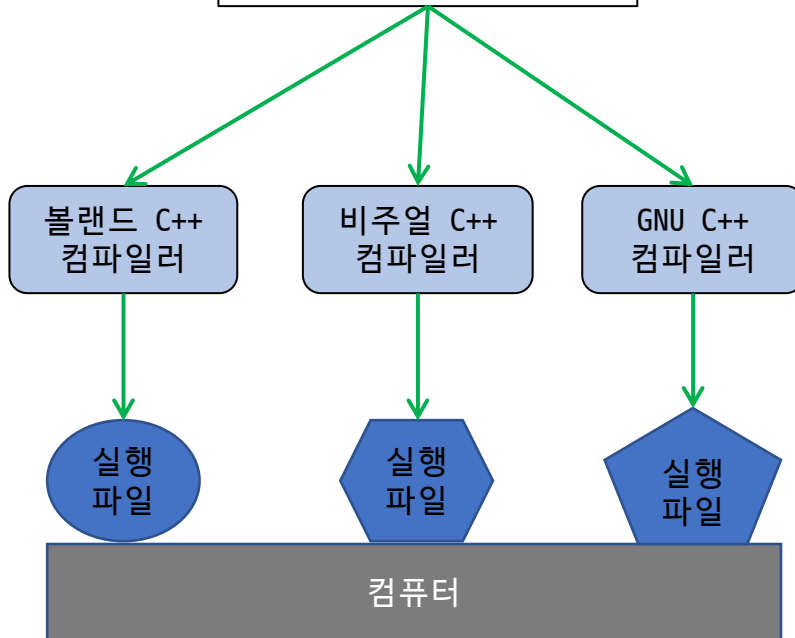


표준/비표준 C++ 프로그램의 비교

표준 C++ 규칙에 따라
작성된 C++ 프로그램

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

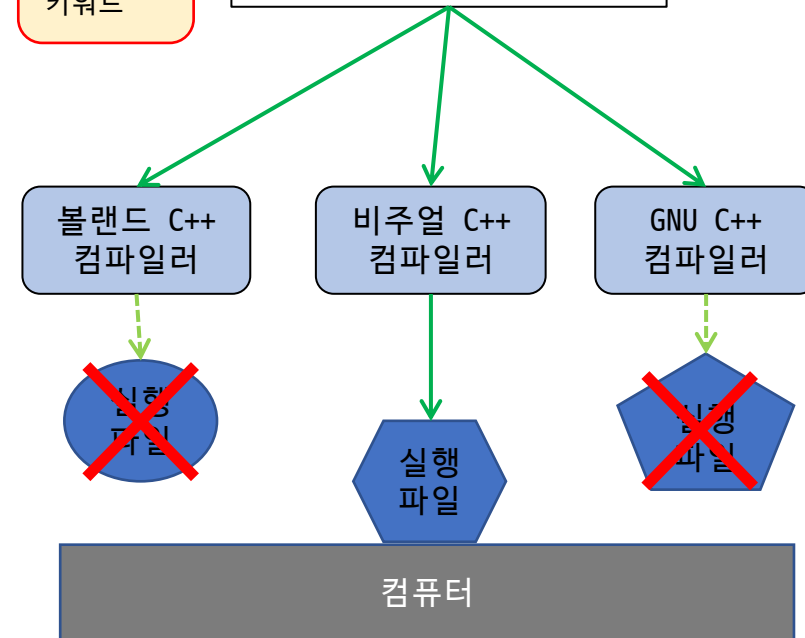
모든 C++
컴파일러
에 의해
컴파일

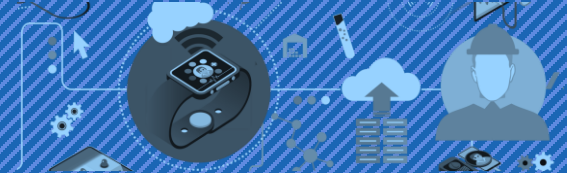


표준 C++ 규칙에 따라
작성되지 않는 비주얼 C++ 프로그램

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

비주얼
C++ 전용
키워드





■ C++ 언어의 주요한 설계 목적

▶ C 언어와의 호환성

▶ C 언어의 문법 체계 계승

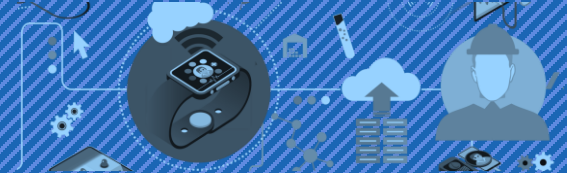
- ▶ 소스 레벨 호환성 : 기존에 작성된 C 프로그램을 그대로 가져다 사용
- ▶ 링크 레벨 호환성 : C 목적 파일과 라이브러리를 C++ 프로그램에서 링크

▶ 객체 지향 개념 도입

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

▶ 엄격한 타입 체크

- ▶ 실행 시간 오류의 가능성을 줄임
- ▶ 디버깅 편리



■ C++ 언어의 주요한 설계 목적

▶ 실행 시간의 효율성 저하 최소화

▶ 실행 시간을 저하시키는 요소와 해결

- ▶ 작은 크기의 멤버 함수 잦은 호출 가능성 → 인라인 함수로 실행시간 저하 해소

▶ 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



■ C++ 언어의 주요한 설계 목적

▶ 실행 시간의 효율성 저하 최소화

▶ 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 : 캡슐화

▶ 캡슐화(encapsulation)

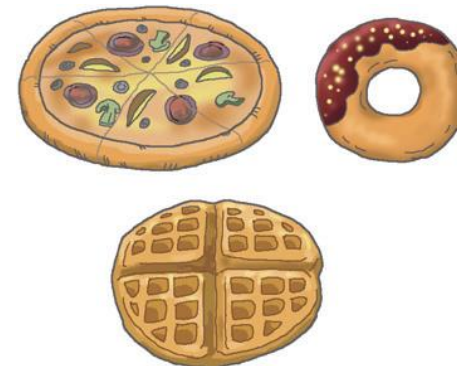
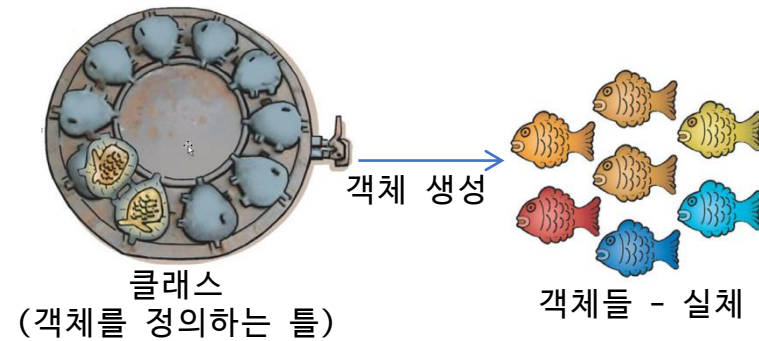
- ▶ 데이터를 캡슐로 싸서 외부의 접근으로부터 보호
- ▶ C++에서 클래스(class 키워드)로 캡슐 표현

▶ 클래스와 객체

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

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

멤버들



원 객체들(실체)

C++ 객체 지향 특성 : 상속성

▶ 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();  
};
```

Phone을 상속받는다.

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

MobilePhone을 상속받는다.

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

C++로 상속 선언





C++ 객체 지향 특성 : 다형성

▶ 다형성(polymorphism)

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

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

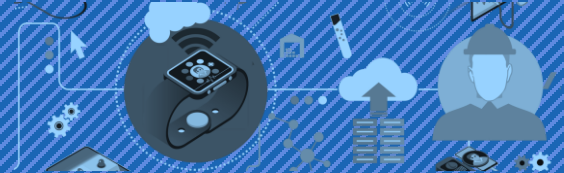
+ 연산자 중복

```
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

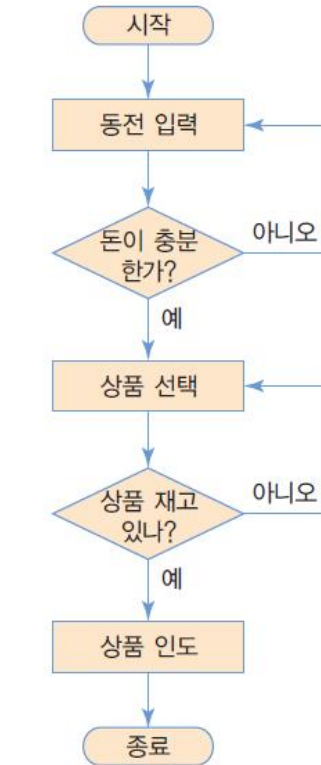
▶ 소프트웨어 생산성 향상

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

▶ 실세계에 대한 쉬운 모델링

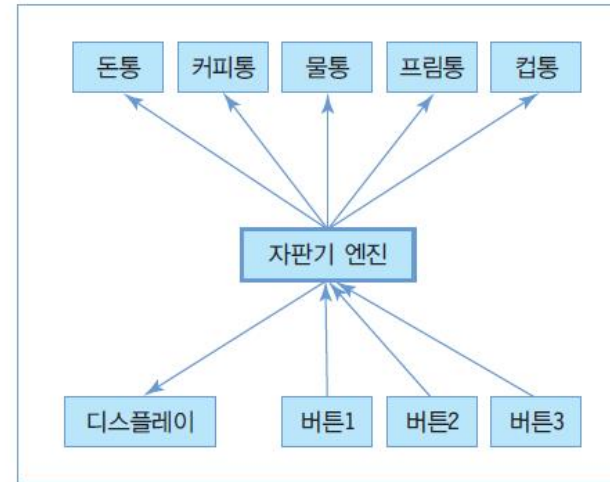
- ▶ 과거의 소프트웨어
 - ▶ 수학 계산이나 통계 처리에 편리한 절차 지향 언어가 적합
- ▶ 현대의 소프트웨어
 - ▶ 물체 혹은 객체의 상호 작용에 대한 묘사가 필요
 - ▶ 실세계는 객체로 구성된 세계
 - ▶ 객체를 중심으로 하는 객체 지향 언어가 적합

절차 지향 프로그래밍과 객체 지향 프로그래밍



(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

▶ 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

▶ 제네릭 프로그래밍(generic programming)

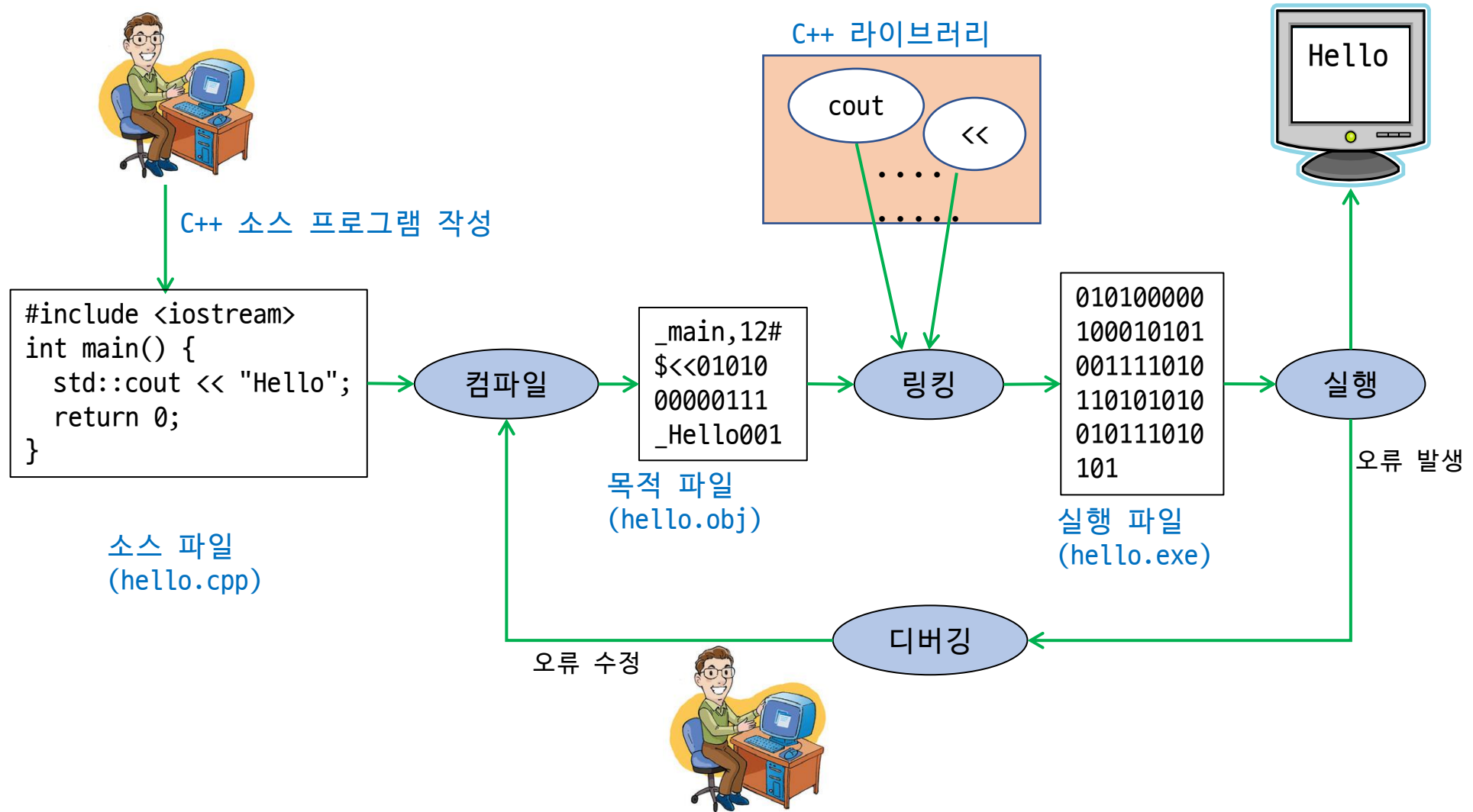
- ▶ 제네릭 함수와 제네릭 클래스를 활용하여 프로그램을 작성하는 새로운 프로그래밍 패러다임
- ▶ 점점 중요성이 높아지고 있음.



■ C++ 언어의 아킬레스

- ▶ C++ 언어는 C언어와의 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





■ C++ 프로그램 작성 및 컴파일

▶ 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

C++ 언어

C++ 프로그램 작성 및 컴파일

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

	PROC		COMDAT
; 3	: int main() {		
00000	55	push ebp	
00001	8b ec	mov ebp, esp	
00003	81 ec c0 00 00		
	00		
00009	53	sub esp, 192	; 000000c0H
0000a	56	push ebx	
0000b	57	push esi	
0000c	8d bd 40 ff ff	push edi	
	ff		
00012	b9 30 00 00 00	lea edi, DWORD PTR [ebp-192]	
00017	b8 cc cc cc cc	mov ecx, 48	; 00000030H
0001c	f3 ab	mov eax, -858993460	; ccccccccH
		rep stosd	
; 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		
	00		
0003c	3b ec	add esp, 192	; 000000c0H
0003e	e8 00 00 00 00	cmp ebp, esp	
00043	8b e5	call __RTC_CheckEsp	
00045	5d	mov esp, ebp	
00046	c3	pop ebp	
		ret 0	
_main	ENDP		

int main() { 라인을 컴파일한 기계어 코드

어셈블리어 코드

■ 링킹

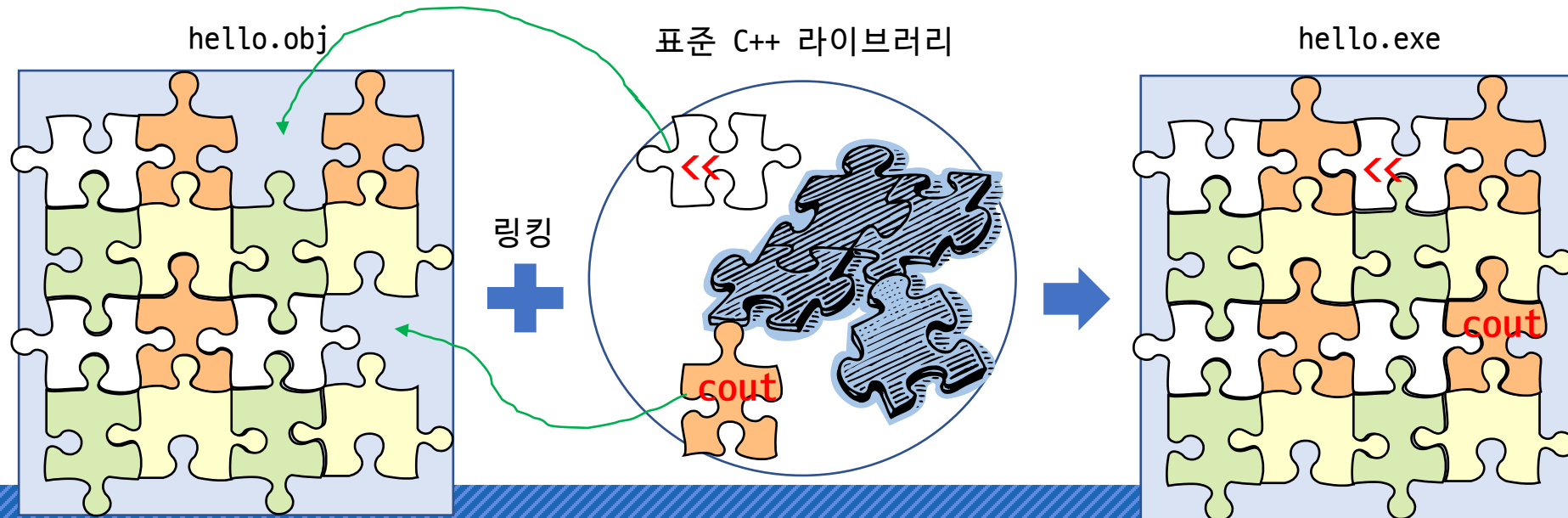
▶ 링킹

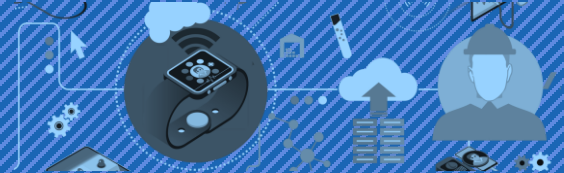
- ▶ 목적 파일끼리 합쳐 실행 파일을 만드는 과정

- ▶ 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

- ▶ 실행 파일은 독립적으로 바로 실행 가능
- ▶ 실행 중 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
 - ▶ 소스레벨 디버깅
 - ▶ C++ 소스를 한 라인씩 실행하고 변수 값의 변화를 보면서 오류 발견
 - ▶ Visual Studio는 소스 레벨 디버깅 지원



■ C++ 표준 라이브러리

- ▶ C++ 표준 라이브러리는 3개의 그룹으로 구분
 - ▶ C 라이브러리
 - ▶ 기존 C 표준 라이브러리를 수용, C++에서 사용할 수 있게 함수들
 - ▶ 이름이 c로 시작하는 헤더 파일에 선언됨.
 - ▶ C++ 입출력 라이브러리
 - ▶ 콘솔 및 파일 입출력을 위한 라이브러리
 - ▶ C++ STL 라이브러리
 - ▶ 제네릭 프로그래밍을 지원하기 위한 템플릿 라이브러리

C++ 표준 라이브러리

▶ C++ 표준 라이브러리는 3개의 그룹으로 구분

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

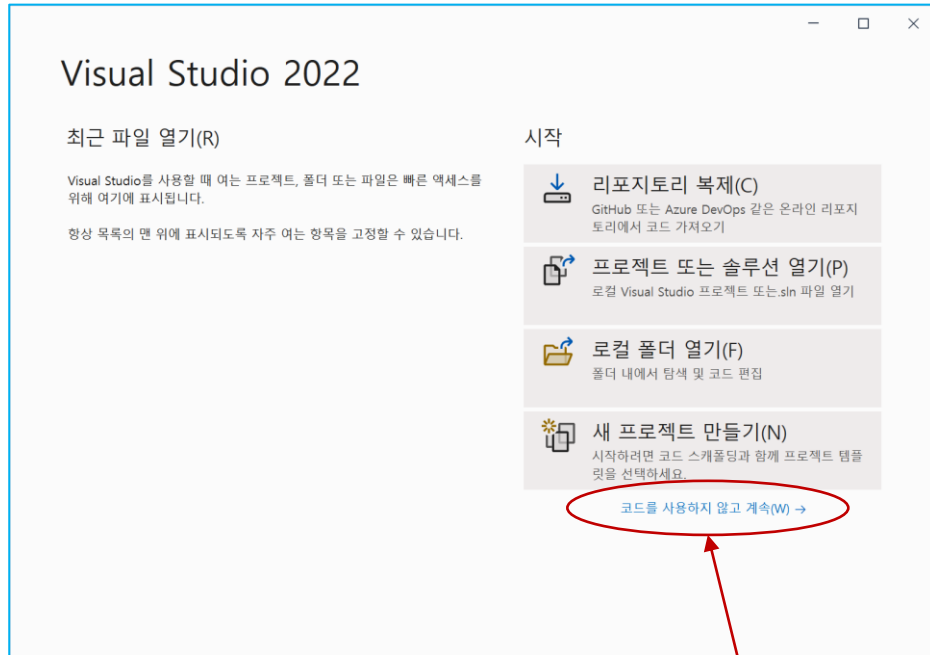
STL 라이브러리

C++ 입출력
라이브러리

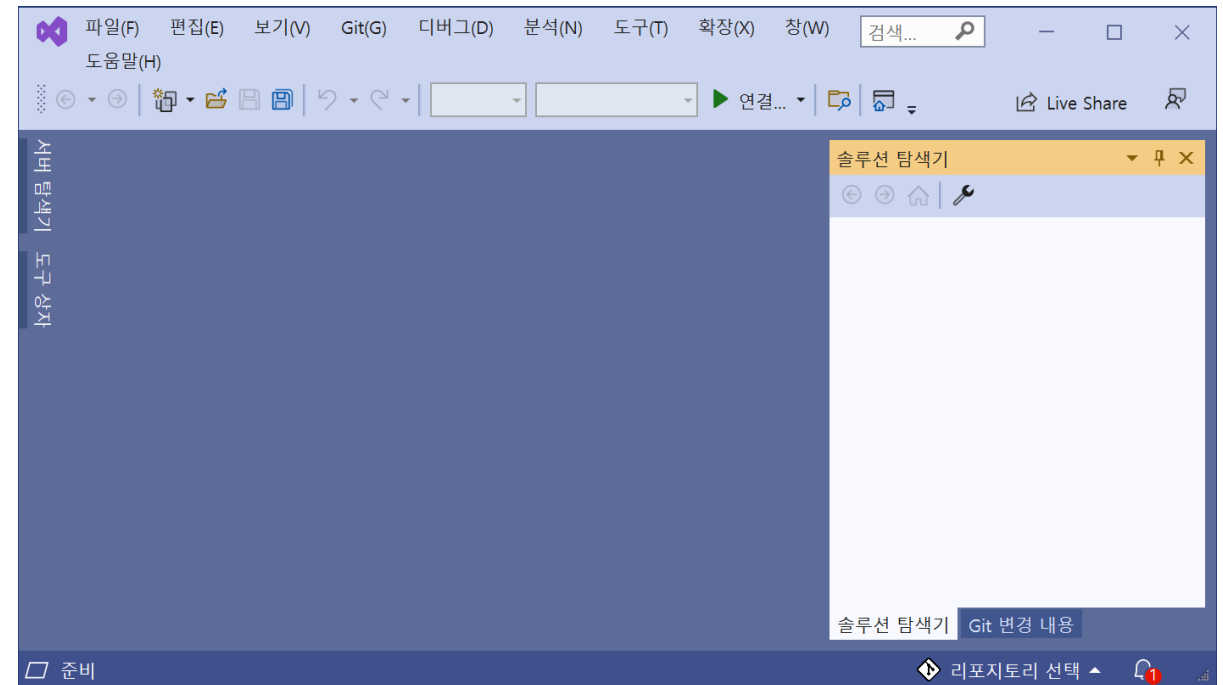
*〈new〉 헤더 파일은 STL에 포함되지 않는 기타 기능을 구현함

C++ 언어

C++ 의 시작 : Hello

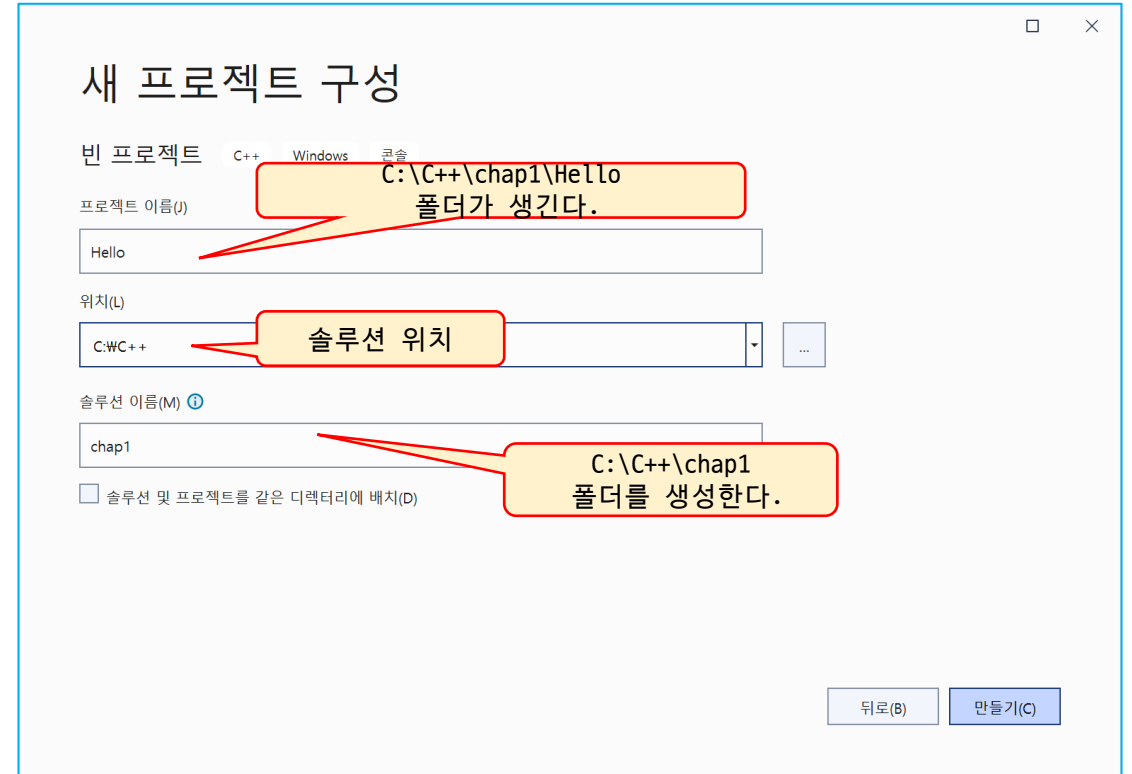


클릭하면
다음 슬라이드로



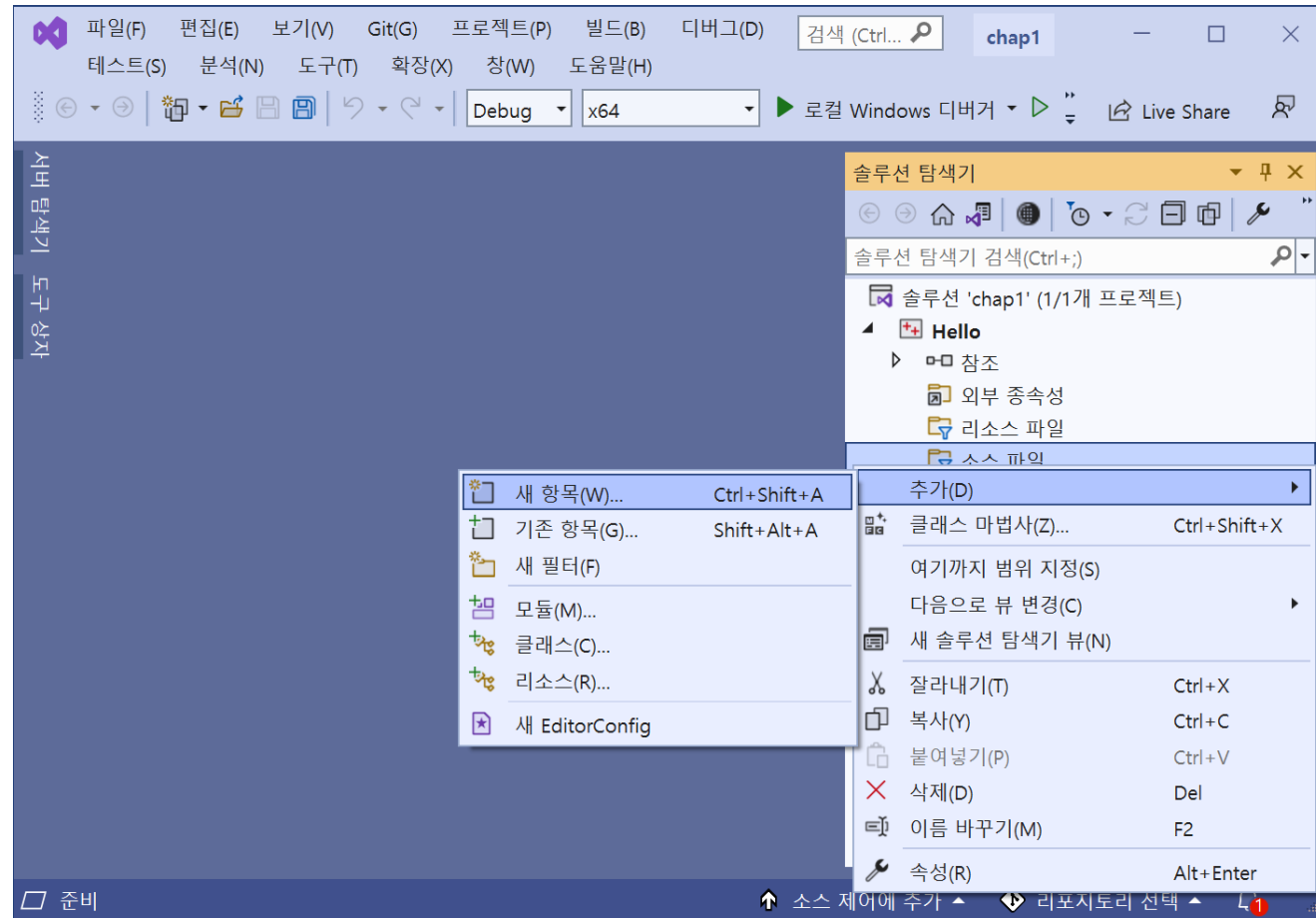
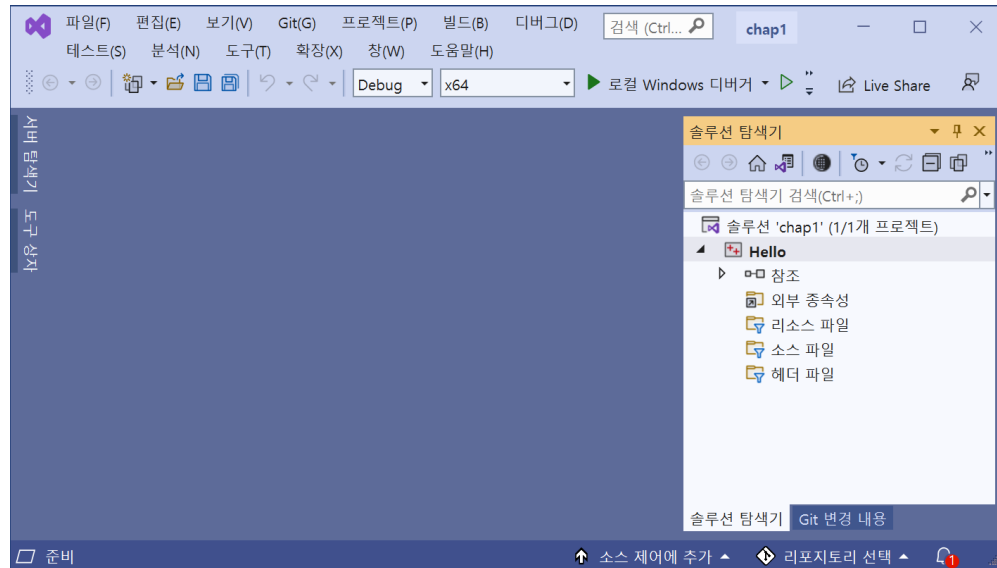
C++ 언어

C++ 의 시작 : Hello

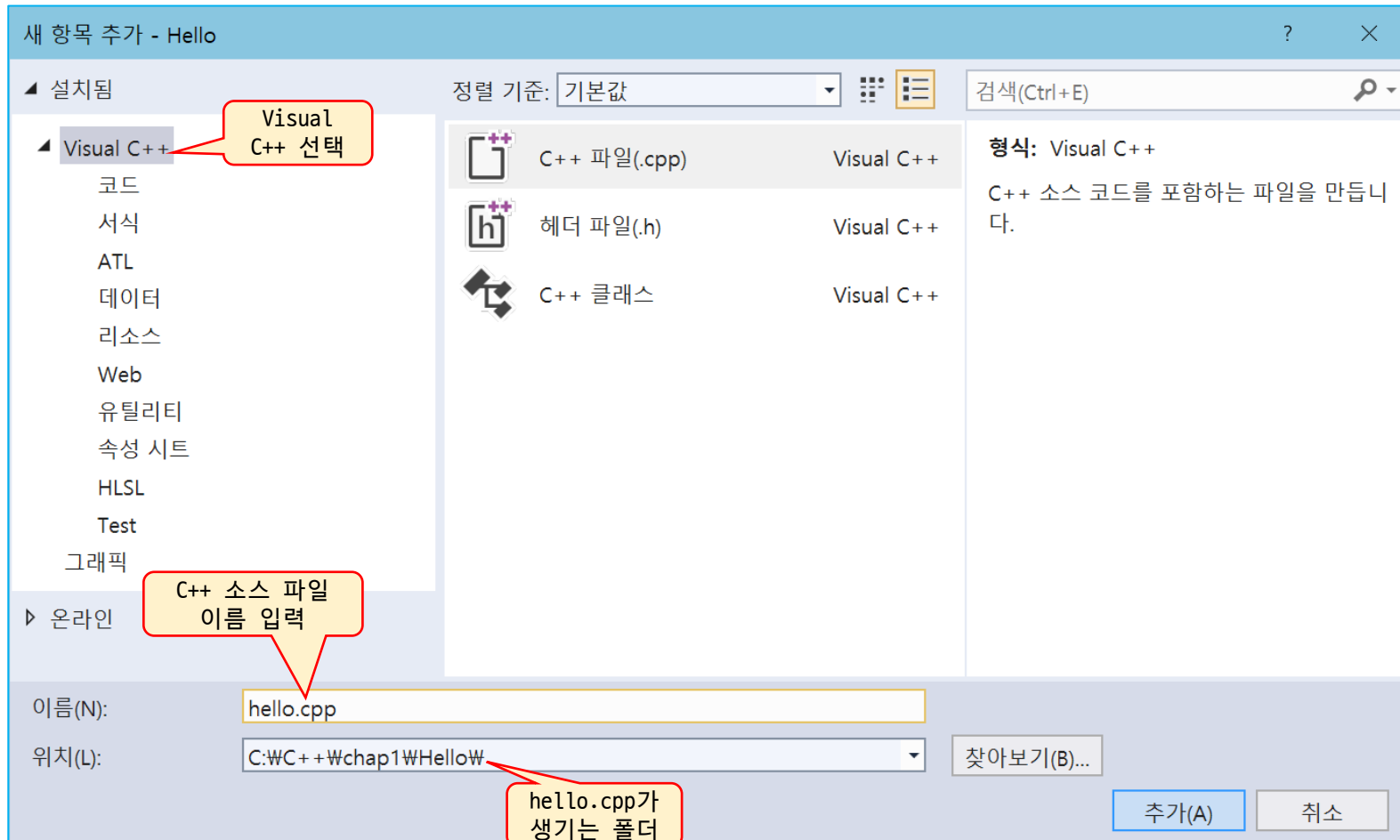


C++ 언어

C++ 의 시작 : Hello

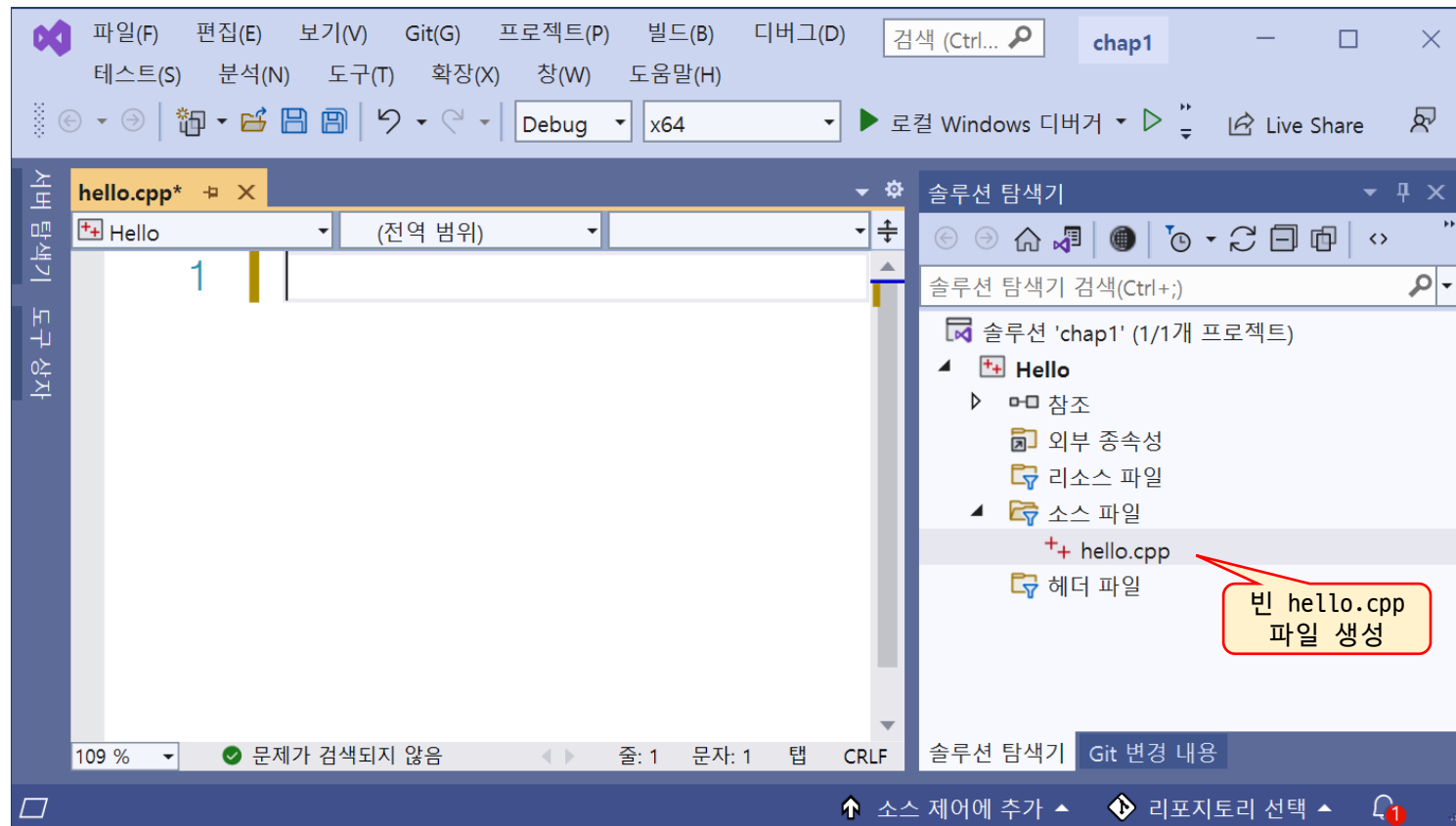


C++ 의 시작 : Hello



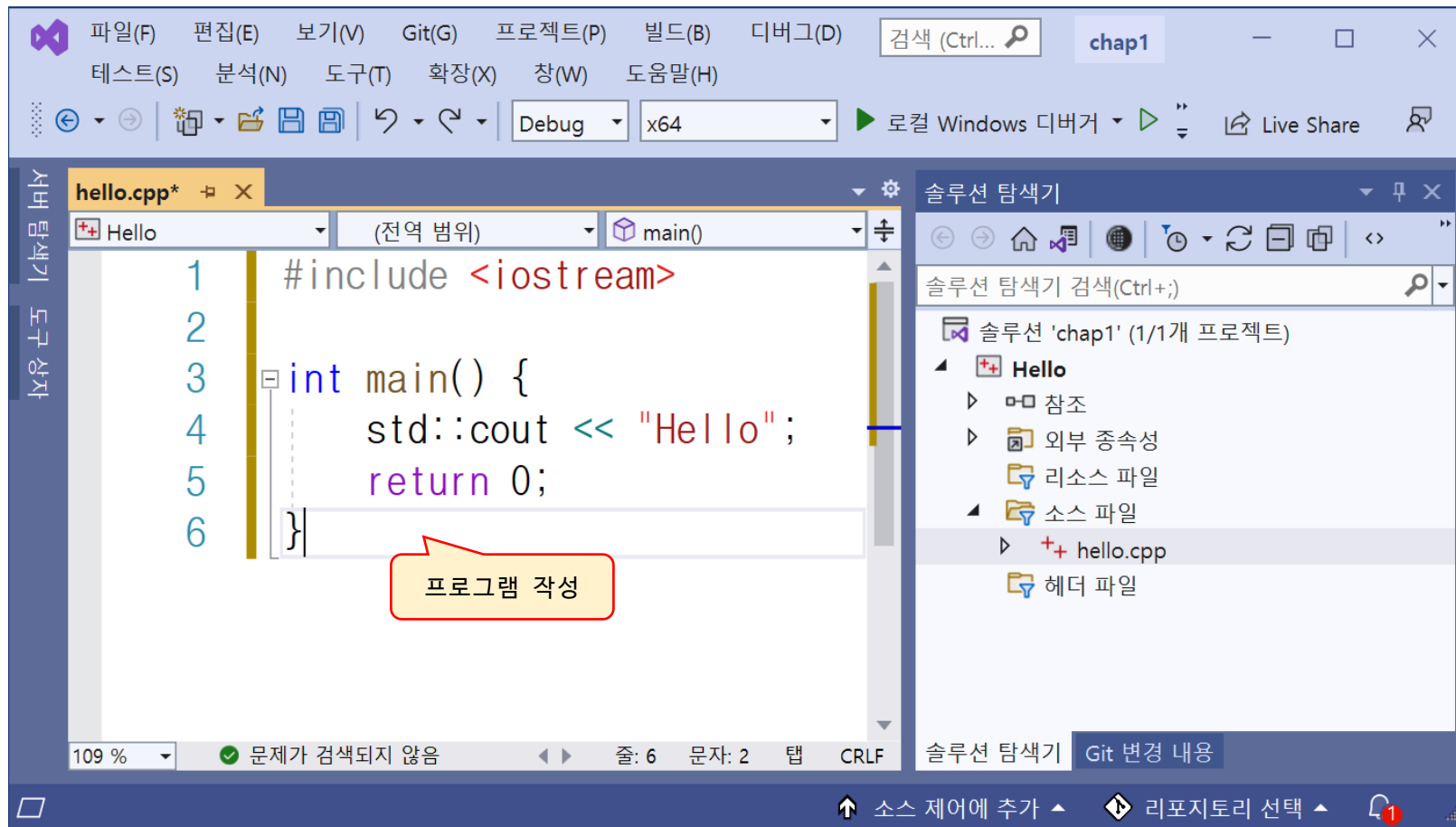
C++ 언어

C++ 의 시작 : Hello



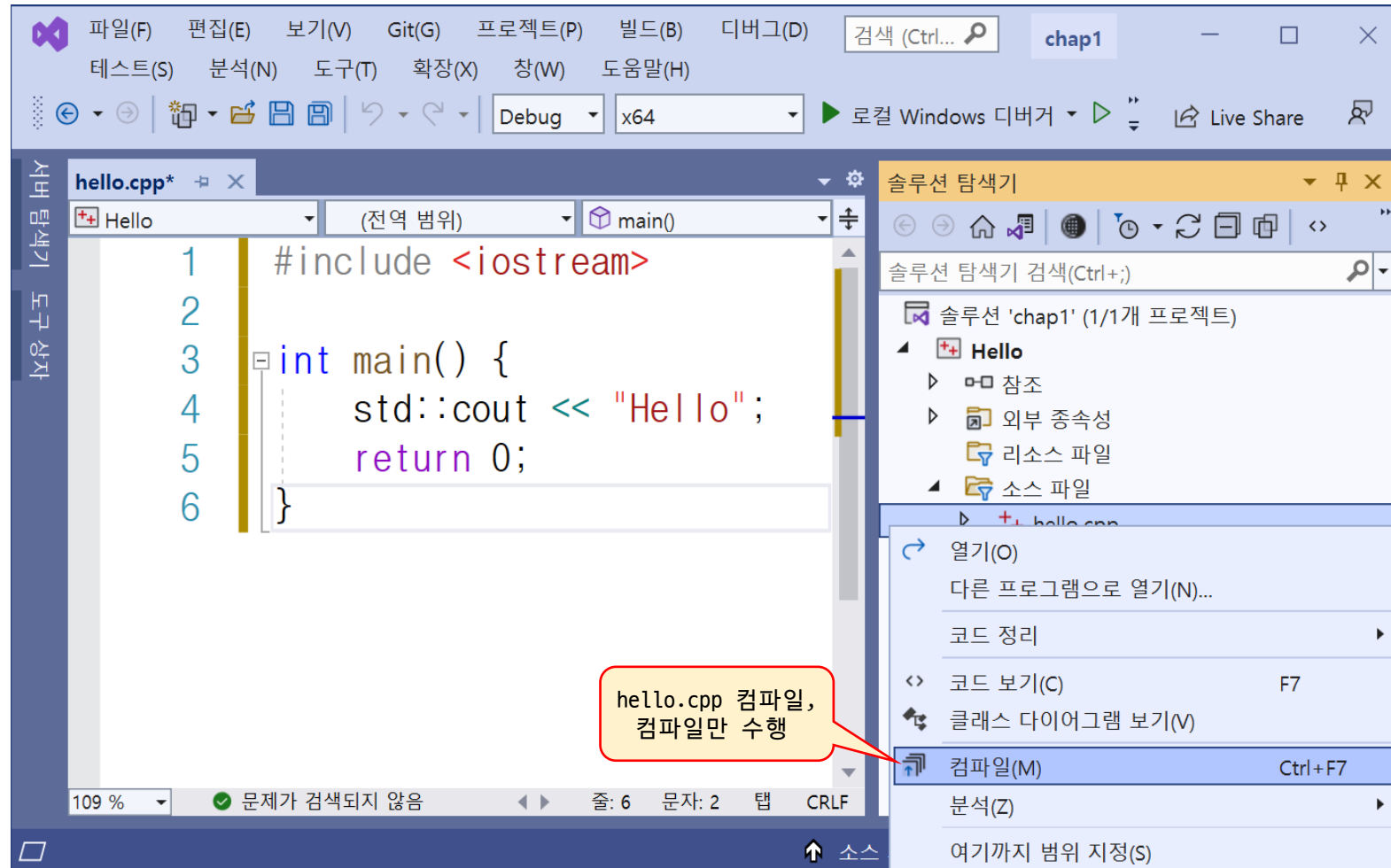
C++ 언어

C++ 의 시작 : Hello



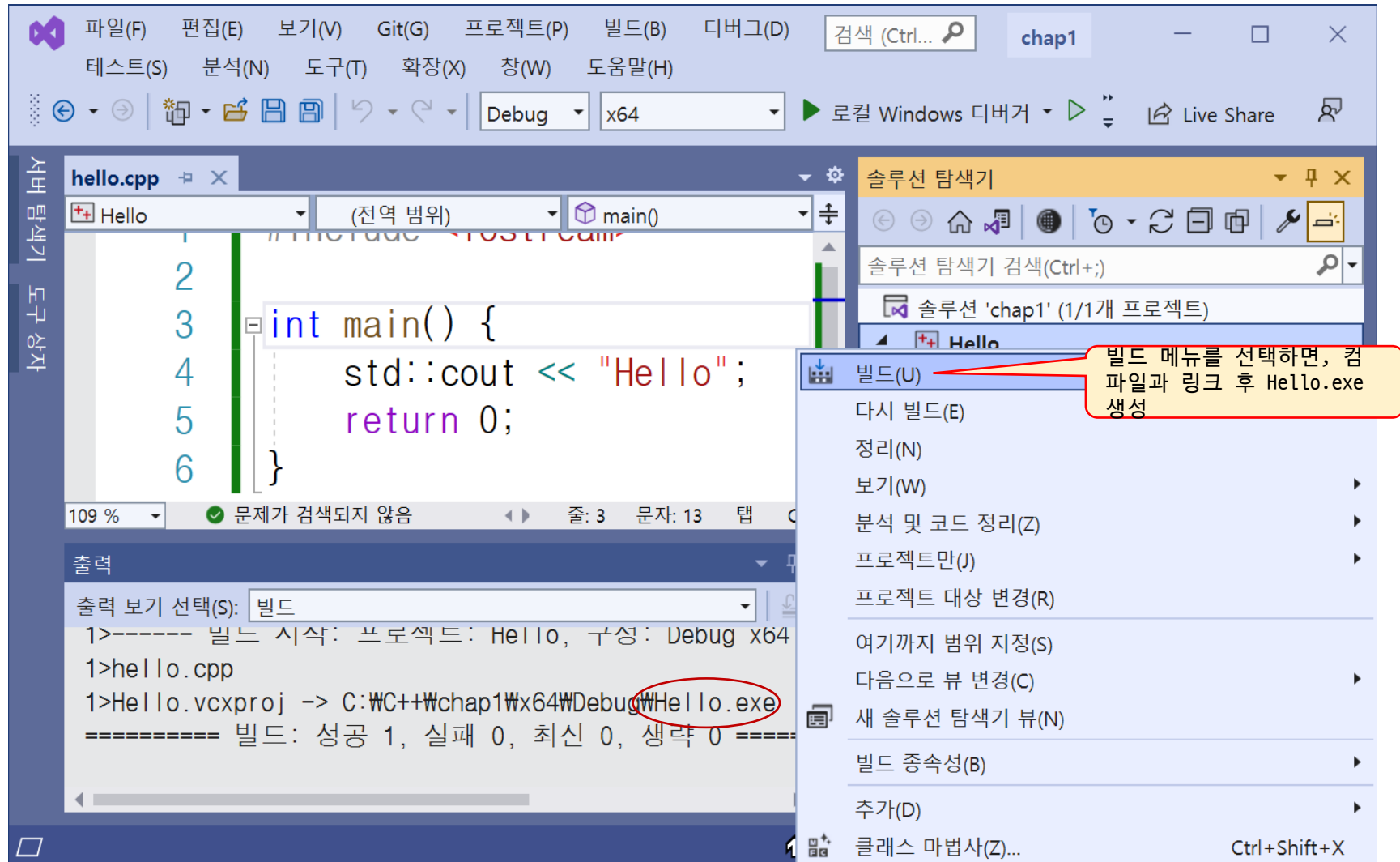
C++ 언어

C++ 의 시작 : Hello



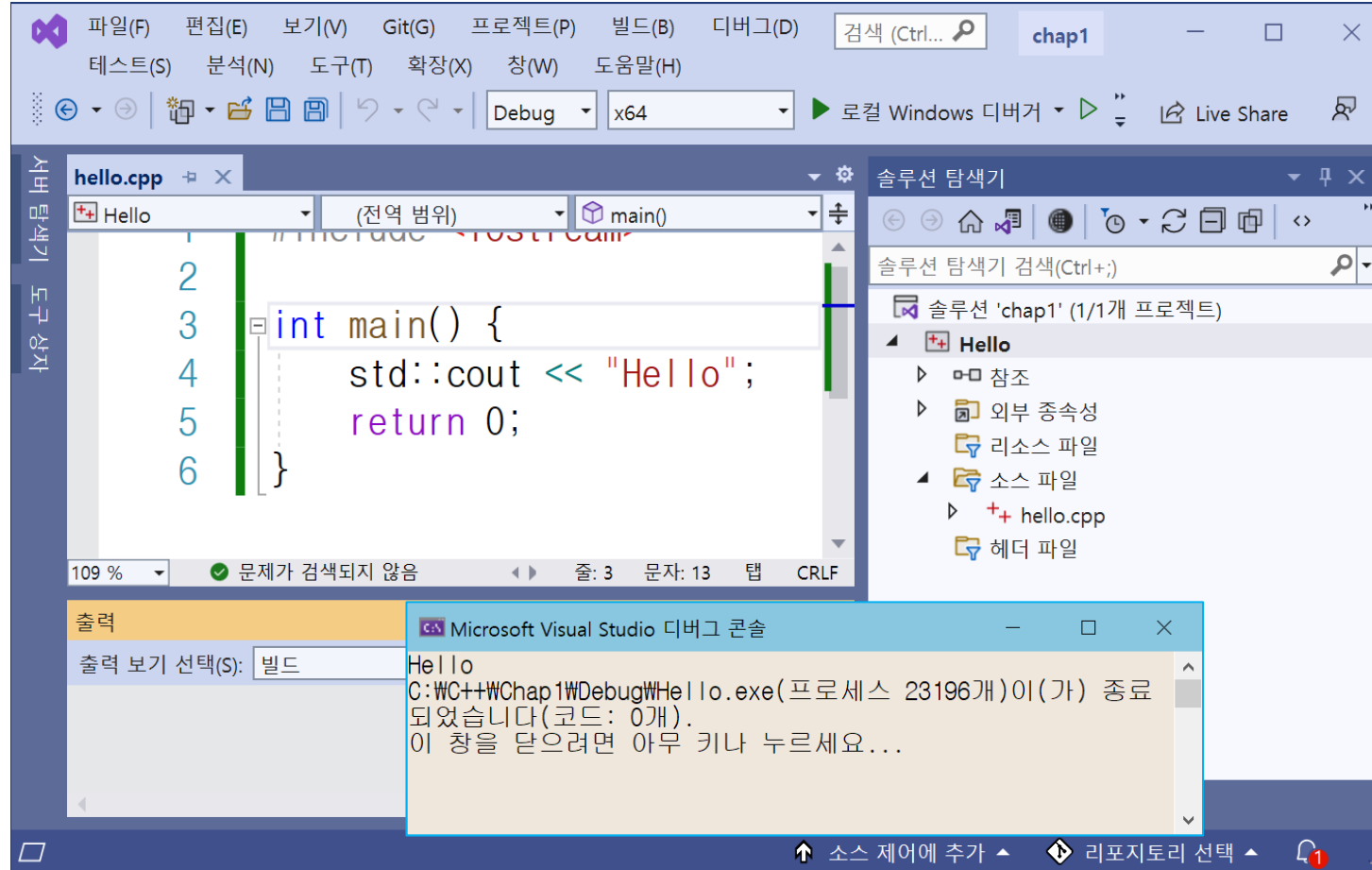
C++ 언어

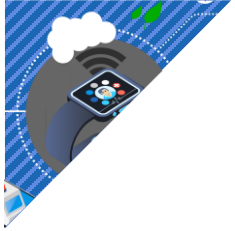
C++ 의 시작 : Hello



C++ 언어

C++ 의 시작 : Hello





C++ 프로그래밍의 기본

C++ 프로그래밍의 기본



C++ 프로그래밍의 기본



C++ 코드 분석을 위한 기본 프로그램

```
/*
    소스: SimpleC++.cpp
    cout과 << 연산자를 이용하여 화면에 출력한다.
*/

#include <iostream> // cout과 << 연산자 포함

// C++ 프로그램은 main() 함수에서부터 실행을 시작한다.
int main() {
    std::cout << "Hello\n"; // 화면에 Hello를 출력하고 다음 줄로 넘어감
    std::cout << "첫 번째 맛보기입니다.";
    return 0; // main() 함수가 종료하면 프로그램이 종료됨
}
```

```
Hello
첫 번째 맛보기입니다.
```




주석과 main() 함수

▶ 주석

- ▶ 개발자가 자유롭게 붙인 특이 사항의 메모, 프로그램에 대한 설명
- ▶ 프로그램의 실행에 영향을 미치지 않음.
 - ▶ 여러 줄 주석 : `/* ... */`, 한 줄 주석 : `//`



주석과 main() 함수

▶ main() 함수

- ▶ C++ 프로그램의 실행을 시작하는 함수
 - ▶ main() 함수가 종료되면 C++ 프로그램 종료
- ▶ C++ 표준 main() 함수

```
int main() { // main()의 리턴 타입 int
    .....
    return 0; // 0이 아닌 다른 값으로 리턴 가능
}
```

```
void main() { // 표준 아님
    .....
}
```

```
int main() {
    .....
    // return 0; // 개발자의 편리를 위해 return 문 생략 가능
}
```

#include

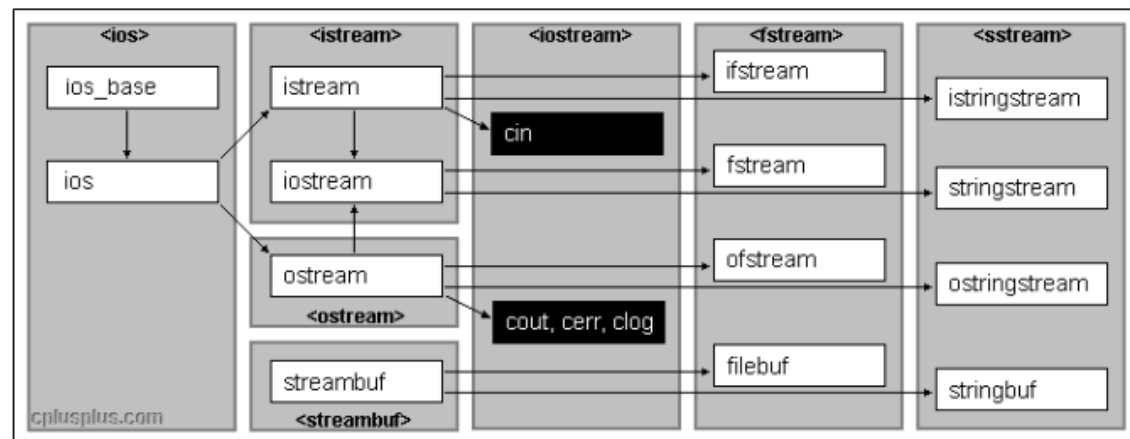
▶ #include <iostream>

- ▶ 전처리기(C++ Preprocessor)에 내리는 지시문
 - ▶ <iostream> 헤더 파일을 컴파일 전에 소스에 확장하도록 지시

▶ <iostream> 헤더파일

- ▶ 표준 입출력을 위한 클래스와 객체, 변수 등이 선언됨.
 - ▶ ios, istream, ostream, iostream 클래스 선언
 - ▶ ios, istream, ostream은 C++에서 직접사용 안함.
 - ▶ cout, cin, <<, >> 등 연산자 선언

```
#include <iostream>
....
std::cout << "Hello\n";
std::cout << "첫 번째 맛보기입니다.";
```





■ 화면출력

▶ cout과 << 연산자 이용

```
std::cout << "Hello\n"; // 화면에 Hello를 출력하고 다음 줄로 넘어감  
std::cout << "첫 번째 맛보기입니다.";
```

▶ cout 객체

- ▶ 스크린 출력 장치에 연결된 표준 C++ 출력 스트림 객체
- ▶ <iostream> 헤더 파일에 선언
- ▶ std 이름공간(namespace)에 선언 → std::cout으로 사용



■ 화면출력

▶ << 연산자

▶ 스트림 삽입 연산자(stream insertion operator)

- ▶ C++ 기본 산술 시프트 연산자(<< >>)가 스트림 삽입 연산자로 재정의 됨
- ▶ ostream 클래스에 구현됨.
- ▶ 오른쪽 피연산자를 왼쪽 스트림 객체에 삽입
- ▶ cout 객체에 연결된 화면에 출력

▶ 여러 개의 << 연산자로 여러 값 동시 출력

```
std::cout << "Hello\n" << "첫 번째 맛보기입니다.";
```



◀ 연산자 활용

▶ 문자열 및 기본 데이터 타입의 데이터 출력

- ▶ Boo, char, short, int, long, float, double 타입 값 출력

```
int n=3;  
char c='#';  
std::cout << c << 5.5 << '-' << n << "hello" << true;
```

- ▶ 연산식 뿐 아니라 함수 호출도 가능

```
std::cout << "n + 5 =" << n + 5;  
std::cout << f(); // 함수 f()의 리턴값을 출력한다.
```

▶ 다음줄로 넘어가기[ENTER]

- ▶ '\n'이나 endl 조작자 사용

```
std::cout << "Hello" << '\n';  
std::cout << "Hello" << std::endl;
```



연산자 활용

```
#include <iostream>

double area(int r); // 함수의 원형 선언

double area(int r) { // 함수 구현
    return 3.14*r*r; // 반지름 r의 원면적 리턴
}

int main() {
    int n=3;
    char c='#';
    std::cout << c << 5.5 << '-' << n << "hello" << true << std::endl;
    std::cout << "n + 5 = " << n + 5 << '\n';
    std::cout << "면적은 " << area(n); // 함수 area()의 리턴 값 출력
}
```

true는 1
로 출력됨

```
#5.5-3hello1
n + 5 = 8
면적은 28.26
```



printf() ???



C 언어에서 사용했던 printf를 더 이상 C++에서 사용하지 말기 바란다. printf()나 scanf() 등을 사용하여 구석기 시대로 회귀한다면, 더 이상 C++ 프로그래머로서의 미래는 없다.





이름 충돌 사례



우리 아파트에 여러 명의 마이클이 산다.
마이클을 부를 때, 1동::마이클, 2동::마이클로 부른다.



namespace의 개념

- ▶ 이름(identifier) 충돌이 발생하는 경우
 - ▶ 여러 명이 서로 나누어 프로젝트를 개발하는 경우
 - ▶ 오픈 소스 혹은 다른 사람이 작성한 소스나 목적파일을 가져와서 컴파일 하거나 링크하는 경우
 - ▶ 해결하는데 많은 시간과 노력이 필요함.
- ▶ namespace 키워드
 - ▶ 이름 충돌 해결
 - ▶ 2003년 새로운 C++ 표준에서 도입
 - ▶ 개발자가 자신만의 이름 공간을 생성할 수 있도록 함
 - ▶ 이름 공간 안에 선언된 이름은 다른 이름공간과 별도로 구분됨.



namespace의 개념

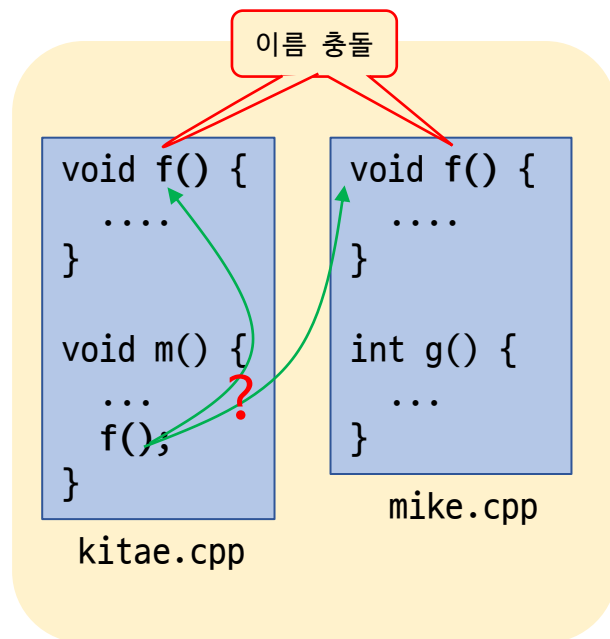
▶ 이름 공간 생성 및 사용

```
namespace foxliver { // foxliver 라는 이름 공간 생성
    ..... // 이 곳에 선언된 모든 이름은 foxliver 이름 공간에 생성된 이름
}
```

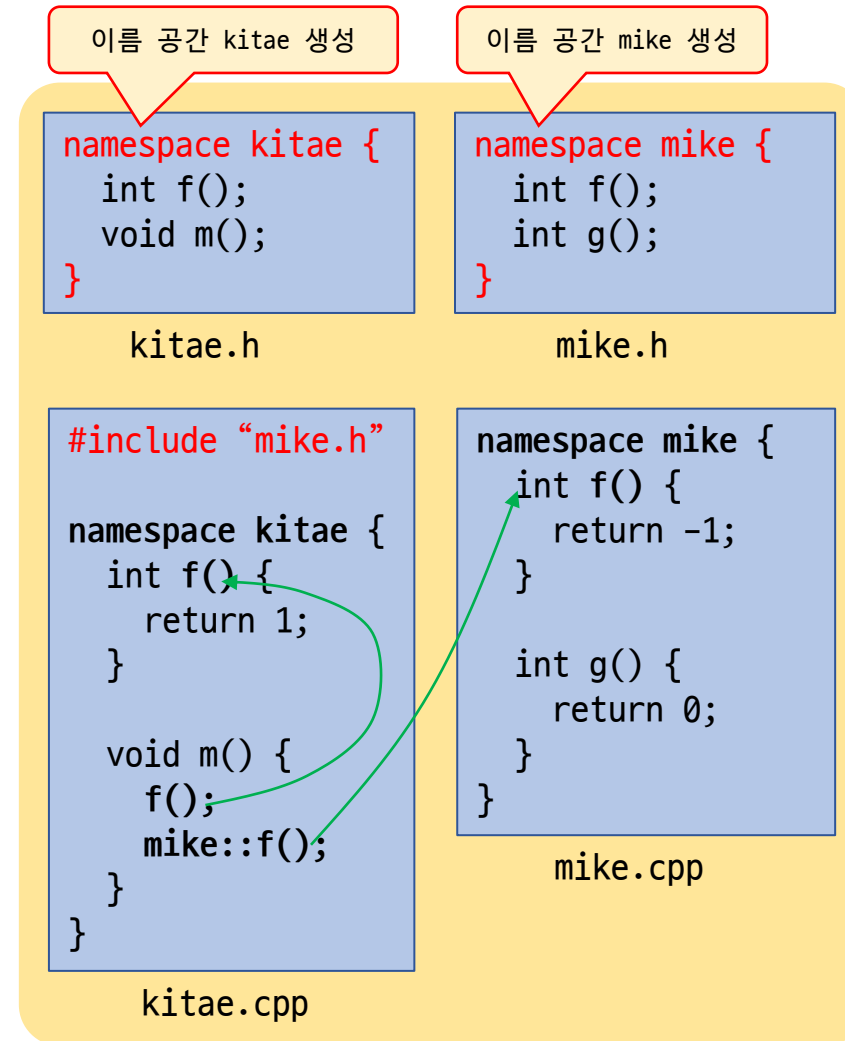
▶ 이름 공간 사용

- ▶ 이름공간:이름 → foxliver::내부에 구성된 요소

namespace의 개념



(a) kitae와 mike에 의해 작성된 소스를 합치면 f() 함수의 이름 충돌. 컴파일 오류 발생



(b) 이름 공간을 사용하여 f() 함수 이름의 충돌 문제 해결



std:: ???

▶ std(standard)

- ▶ C++ 표준에서 정의한 이름공간(namespace)중 하나
 - ▶ <iostream> 헤더 파일에 선언된 모든 이름 : std 이름 공간 안에 있음.
 - ▶ cout, cin, endl 등
- ▶ std 이름 공간에 선언된 이름을 접근하기 위해 std:: 접두어 사용
 - ▶ std::cout, std::cin, std::endl

▶ std:: 생략

- ▶ using 지시어 사용

std:: 생략

```
using std::cout; // cout에 대해서만 std:: 생략
.....
cout << "Hello" << std::endl; // std::cout에서 std:: 생략
```

```
using namespace std; // std 이름 공간에 선언된 모든 이름에 std::
생략
.....
cout << "Hello" << endl; // std:: 생략
```

std:: 생략

std:: 생략



■ #include <iostream>과 std

▶ 개요

- ▶ <iostream>이 통째로 std 이름 공간 내에 선언
 - ▶ <iostream> 헤더 파일을 사용하려면 다음 코드 필요

```
#include <iostream>
using namespace std;
```



■ #include <iostream>과 std

▶ 실습문제 1

- ▶ 교재 88페이지 문제 1

▶ 실습문제 2

- ▶ 교재 89페이지 문제 2



C++ 코드 분석을 위한 기본 프로그램(입력)

```
#include <iostream>
using namespace std;

int main() {
    cout << "너비를 입력하세요>>";

    int width;
    cin >> width; // 키보드로부터 너비를 읽어 width 변수에 저장

    cout << "높이를 입력하세요>>";

    int height;
    cin >> height; // 키보드로부터 높이를 읽어 height 변수에 저장

    int area = width*height; // 사각형의 면적 계산
    cout << "면적은 " << area << "\n"; // 면적을 출력하고 다음 줄로 넘어감
}
```

```
너비를 입력하세요>>3
높이를 입력하세요>>5
면적은 15
```




cin과 >> 연산자를 이용한 키 입력

▶ cin

- ▶ 표준 입력 장치인 키보드를 연결하는 C++ 입력 스트림 객체

▶ >> 연산자

- ▶ 스트림 추출 연산자(stream extraction operator)
 - ▶ C++ 산술 시프트 연산자(>>)가 <iostream> 헤더 파일에 스트림 추출 연산자로 재정의 됨
 - ▶ 입력 스트림에서 값을 읽어 변수에 저장
- ▶ 연속된 >> 연산자를 사용하여 여러 값 입력 가능

```
cout << "너비와 높이를 입력하세요>>";  
cin >> width >> height;  
cout << width << '\n' << height << '\n';
```

너비와 높이를 입력하세요>>23 36

23

36

width에
입력

height에
입력



■ #include <iostream>과 std

▶ 실습문제 3

- ▶ 교재 89페이지 문제 3

▶ 실습문제 4

- ▶ 교재 89페이지 문제 4



■ [ENTER] 키를 입력할 때 변수에 값 전달

▶ cin의 특징

- ▶ 입력 버퍼를 내장하고 있음.
- ▶ [ENTER]키가 입력될 때까지 입력된 키를 입력 버퍼에 저장
 - ▶ 도중에 [backspace]키를 입력하면 입력된 키 삭제

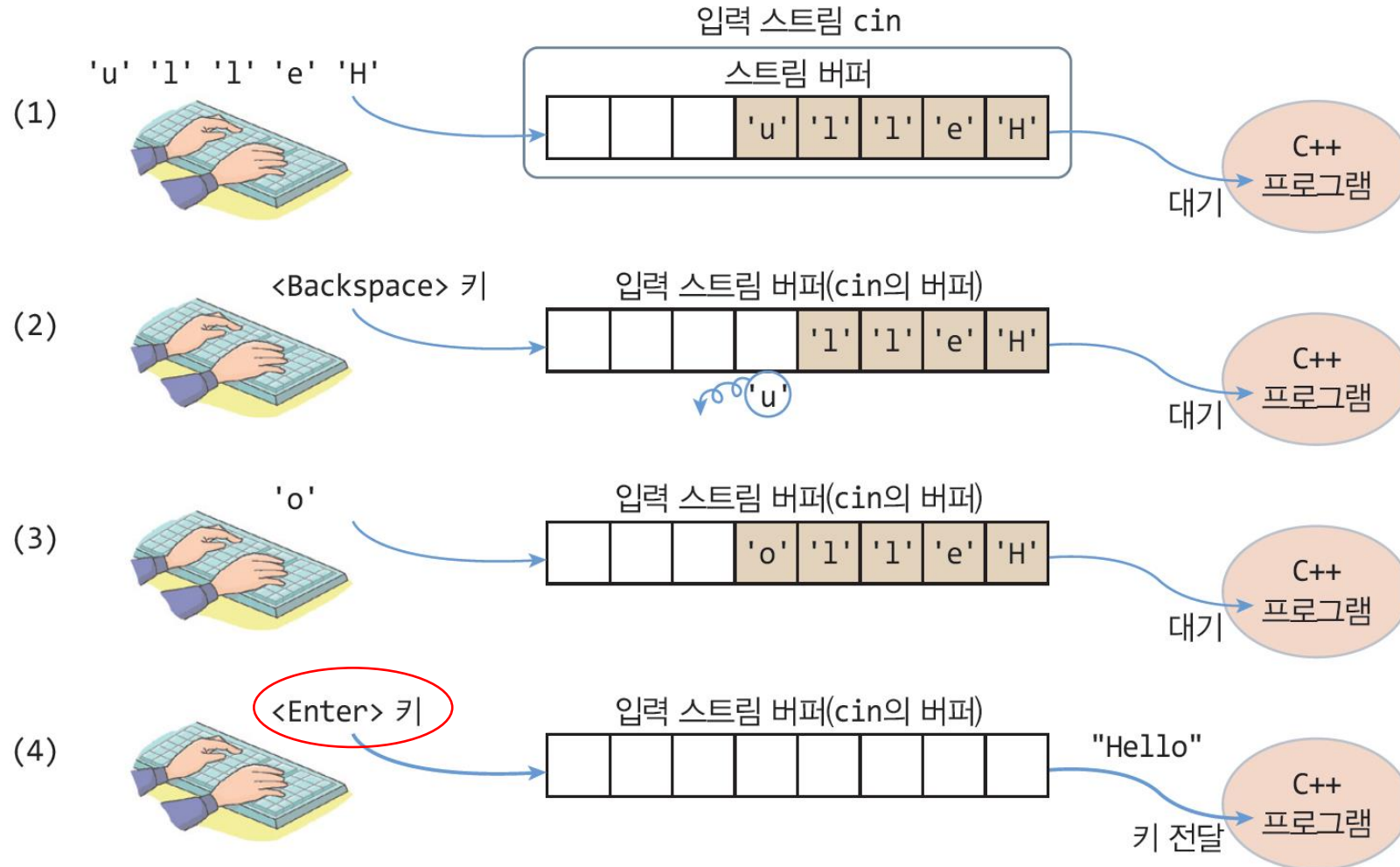
▶ >> 연산자

- ▶ [ENTER] 키가 입력되면 비로소 cin의 입력 버퍼에서 키 값을 읽어 변수에 전달

C++ 프로그래밍의 기본



■ [ENTER] 키를 입력할 때 변수에 값 전달





■ 실행문 중간에 변수 선언

▶ C++의 변수 선언

- ▶ 변수 선언은 아무 곳이나 가능

실행문 중간
에 변수 선언

```
int width;  
cin >> width; // 키보드로부터 너비를 읽는다.  
  
cout << "높이를 입력하세요>>";  
  
int height;  
cin >> height; // 키보드로부터 높이를 읽는다.  
  
// 너비와 높이로 구성되는 사각형의 면적을 계산한다.  
int area = width*height;  
cout << "면적은 " << area << "\n"; // 면적을 출력하고 한 줄 뺀다.
```

▶ C++ 변수 선언 방식의 장점

- ▶ 변수를 사용하기 직전에 선언함으로써 변수 이름에 대한 타이핑 오류 줄임

▶ C++ 변수 선언 방식의 단점

- ▶ 선언된 변수를 일괄적으로 보기 힘들
- ▶ 코드 사이에 있는 변수를 찾기 어려움.



타이핑 오류 가능성 해소

- ▶ 선언부에 모든 변수를 선언하는 경우, 타이핑 오류 가능

```
int time, timer;
...
timer = 5; // time에 5을 저장하려다 timer로 잘못 입력. 컴파일 오류 발생하지 않음
           // 그러나 잘못된 실행 결과 발생
....
timer = 3;
```

- ▶ 변수 사용 전에 변수를 선언하면, 타이핑 오류 사전 발견

컴파일 오류

```
int time;
timer = 5; // time에 5을 저장하려다 timer로 잘못 입력. 컴파일 오류 발생
....
int timer;
timer = 3;
```

C++ 프로그래밍의 기본



C++ 문자열

▶ C++의 문자열 표현 방식 : 2가지

▶ C-STRING 방식 : '\n'로 끝나는 문자 배열

C-STRING
문자열

```
char name1[6] = {'G', 'r', 'a', 'c', 'e', '\0'}; // name1은 문자열 "Grace"
```

단순 문자
배열

```
char name2[5] = {'G', 'r', 'a', 'c', 'e'}; // name2는 문자열이 아니고 단순 문자 배열
```

```
char name5[10] = "Grace";
```

name5[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]

'G'	'r'	'a'	'c'	'e'	'\0'	'\0'	'\0'	'\0'	'\0'
-----	-----	-----	-----	-----	------	------	------	------	------

"Grace" 문자열

'\0'로 초기화

▶ string 클래스 이용

- ▶ <string> 헤더 파일에 선언됨
- ▶ 다양한 멤버 함수 제공, 문자열 비교, 복사, 수정 등



■ C++ 문자열 다루기(C-스tring 방식)

▶ C 언어에서 사용한 함수 사용 가능

▶ strcmp(), strlen(), strcpy() 등

▶ #include

▶ <cstring> 또는 <string.h>

▶ 팁 : <cstring>은 C++ 표준 방식으로 이 방식을 더 추천함.

```
#include <cstring> 또는  
#include <string.h>  
...  
int n = strlen("hello");
```




cin을 이용한 문자열 입력

▶ 문자열 입력

```
char name[6]; // 5 개의 문자를 저장할 수 있는 char 배열  
cin >> name; // 키보드로부터 문자열을 읽어 name 배열에 저장한다.
```

Grace

키 입력

name [0] [1] [2] [3] [4] [5]

'G'	'r'	'a'	'c'	'e'	'\0'
-----	-----	-----	-----	-----	------

“Grace” 문자열



문자열 입력 받고 출력

▶ 문자열 입/출력

```
#include <iostream>
using namespace std;

int main() {
    cout << "이름을 입력하세요>>";

    char name[11]; // 한글은 5개 글자, 영문은 10까지 저장할 수 있다.
    cin >> name; // 키보드로부터 문자열을 읽는다.

    cout << "이름은 " << name << "입니다\n"; // 이름을 출력한다.
}
```

이름을 입력하세요>>마이클
이름은 마이클입니다

빈 칸 없이 키 입력해야 함

이름을 입력하세요>>마 이 클
이름은 마입니다

빈 칸을 만나면 문자열 입력 종료



C-스트링을 이용한 응용

▶ 문자열 입/출력

```
#include <iostream>
#include <cstring>
using namespace std;

int main() {
    char password[11];
    cout << "프로그램을 종료하려면 암호를 입력하세요." << endl;
    while(true) {
        cout << "암호>>";
        cin >> password;
        if(strcmp(password, "C++") == 0) {
            cout << "프로그램을 정상 종료합니다." << endl;
            break;
        }
        else
            cout << "암호가 틀립니다~~" << endl;
    }
}
```

strcmp() 함수를 사용
하기 위한 헤더 파일

프로그램을 종료하려면 암호를 입력하세요.

암호>>Java

암호가 틀립니다~~

암호>>C

암호가 틀립니다~~

암호>>C++

프로그램을 정상 종료합니다.

빈 칸 없이 키 입력해야 함



■ #include <iostream>과 std

▶ 실습문제 5

- ▶ 교재 89페이지 문제 5

▶ 실습문제 6

- ▶ 교재 89페이지 문제 6



공백 포함 문자열 입력

▶ cin.getline()

- ▶ 공백이 포함된 문자열을 입력 받기 위한 방법
- ▶ cin.getline(char buf[], int size, char delimiterChar)
 - ▶ buf에 최대 size - 1개의 문자 입력. "size - 1"의 이유는 문자열의 끝은 항상 '\0'이기 때문에
 - ▶ delimiterChar를 만나면, 입력 중단. 끝에 '\0'이 붙음
 - ▶ delimiterChar의 디폴트 값은 '\n'(ENTER 키)

```
char address[100];  
cin.getline(address, 100, '\n');
```

최대 99개의 문자를 읽어 address 배열에 저장. 도중에 <Enter> 키를 만나면 입력 중단

사용자가 'Seoul Korea<Enter>'를 입력할 때,

address[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [99]

'S'	'e'	'o'	'u'	'l'	' '	'K'	'o'	'r'	'e'	'a'	'\0'
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	-----	-----

"Seoul Korea" 문자열



공백 포함 문자열 입력

▶ cin.getline()

```
#include <iostream>
using namespace std;

int main() {
    cout << "주소를 입력하세요>>";

    char address[100];
    cin.getline(address, 100, '\n'); // 키보드로부터 주소 읽기

    cout << "주소는 " << address << "입니다\n"; // 주소 출력
}
```

주소를 입력하세요>>컴퓨터시 프로그램구 C++동 스트링 1-1
주소는 컴퓨터시 프로그램구 C++동 스트링 1-1입니다

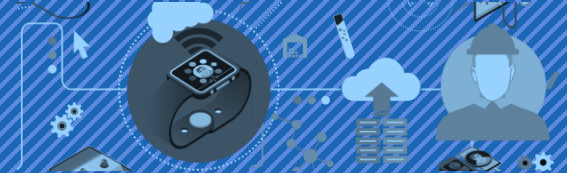
빈칸이 있어도 <Enter> 키가 입력
될 때까지 하나의 문자열로 인식



string 클래스 : 강추

▶ string 클래스

- ▶ C++에서 강력 추천
- ▶ C++ 표준 클래스
- ▶ 문자열의 크기에 따른 제약 없음
 - ▶ string 클래스가 스스로 문자열 크기에 맞게 내부 버퍼 조절
- ▶ 문자열 복사, 비교, 수정 등을 위한 다양한 함수와 연산자 제공
- ▶ 객체 지향적
- ▶ `<string>` 헤더 파일에 선언
 - ▶ `#include <string>` → 주의! "cstring.h"와는 다른 녀석임.
- ▶ C-스트링보다 다루기 쉬움.



string 클래스 : 문자열 입력 및 응용

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string song("Falling in love with you"); // 문자열 song
    string elvis("Elvis Presley"); // 문자열 elvis
    string singer; // 문자열 singer

    cout << song + "를 부른 가수는"; // + 로 문자열 연결
    cout << "(힌트 : 첫글자는 " << elvis[0] << ")?" << endl; // [] 연산자 사용

    getline(cin, singer); // 문자열 입력
    if(singer == elvis) // 문자열 비교
        cout << "맞았습니다.";
    else
        cout << "틀렸습니다. " + elvis + "입니다." << endl; // +로 문자열 연결
}
```

string 클래스를 사용하기 위한 헤더 파일

빈칸을 포함하는
문자열 입력 가능

getline()은 string 타입
의 문자열을 입력 받기
위해 제공되는 전역 함수

Falling in love with you를 부른 가수는(힌트 : 첫글자는 E)?Elvis Pride
틀렸습니다. Elvis Presley입니다.

빈칸 포함



■ #include <iostream>과 std

▶ 실습문제 5

- ▶ 교재 89페이지 문제 5

▶ 실습문제 6

- ▶ 교재 89페이지 문제 6

▶ 실습문제 7

- ▶ 교재 90페이지 문제 7

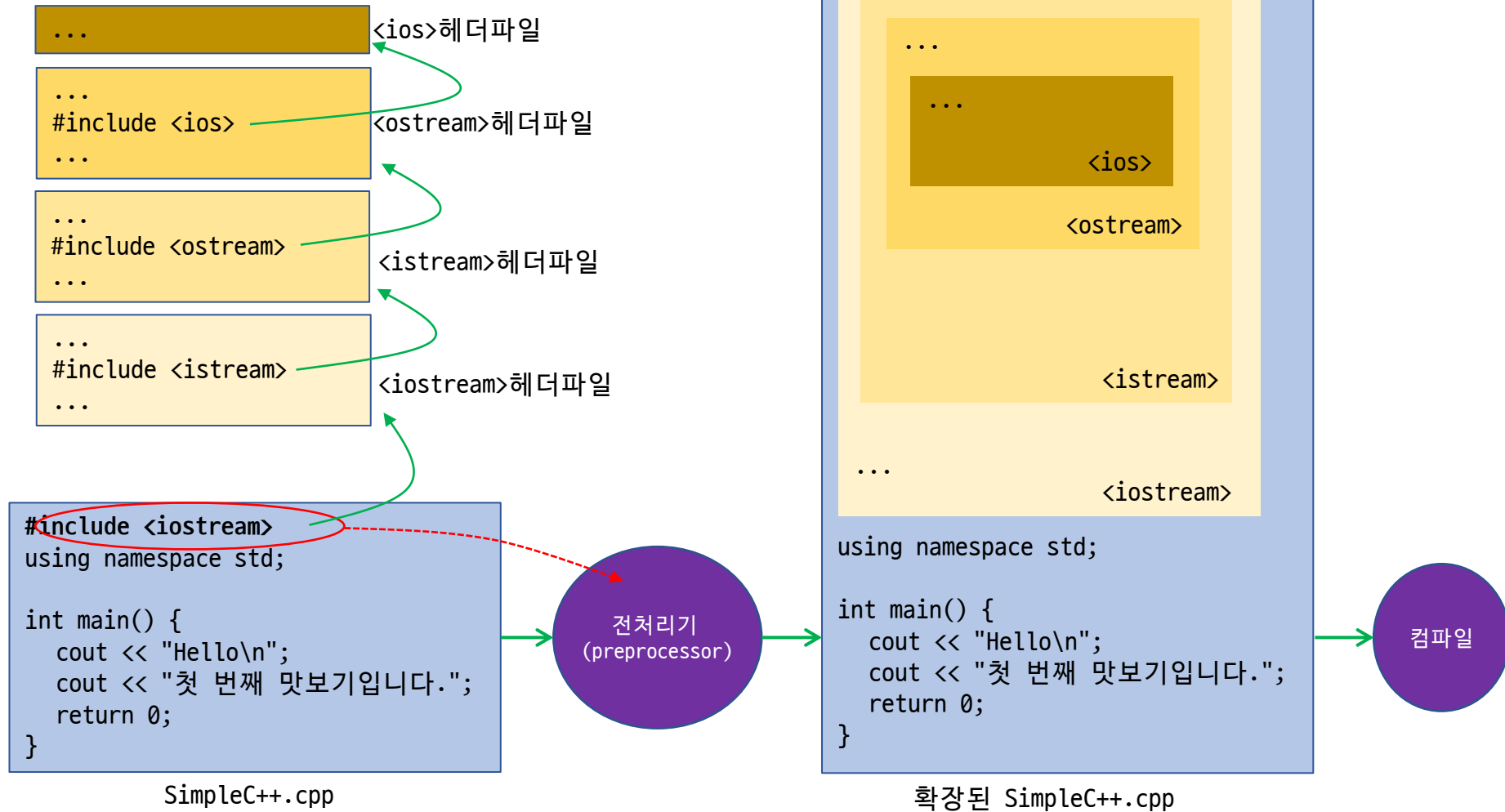
▶ 실습문제 8

- ▶ 교재 90페이지 문제8

C++ 프로그래밍의 기본



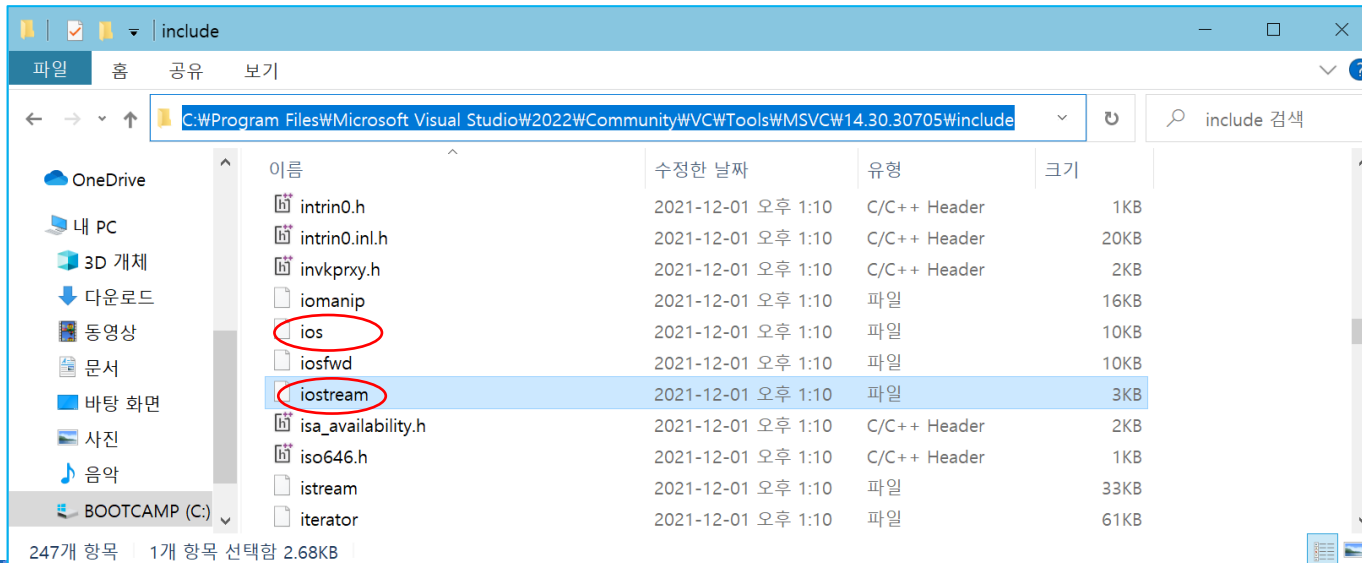
#include <iostream>과 전처리기



■ <iostream> 헤더는 어디에?

▶ <iostream>의 헤더

- ▶ <iostream> 파일은 확장자 없는 텍스트 파일
- ▶ 컴파일러가 설치된 폴더 아래 include 폴더에 존재
 - ▶ 설치되는 버전에 따라 경로명이 다를 수 있음에 주의
 - ▶ `C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.30.30705\include`





표준 C++ 헤더의 특이점

▶ 표준 C++에서 헤더의 특이점

- ▶ 헤더 파일 확장자가 없고, std 이름 공간 적시
- ▶ `#include <iostream>`
- ▶ `using namespace std;`

▶ 헤더 파일의 확장자 비교

언어	헤더 파일 확장자	사례	설명
C	.h	<code><string.h></code>	C/C++ 프로그램에서 사용 가능
C++	확장자 없음	<code><cstring></code>	<code>using namespace std;</code> 와 함께 사용해야 함



#include의 특징

▶ #include <헤더파일>

- ▶ "헤더파일"을 찾는 위치
 - ▶ 컴파일러가 설치된 폴더에서 찾으라는 지시
 - ▶ 예) #include <iostream>은 iostream 파일을 컴파일러가 설치된 폴더에서 찾도록 지시

▶ #include "헤더파일"

- ▶ "헤더파일"을 찾는 위치
 - ▶ 개발자의 프로젝트 폴더나
 - ▶ 개발자가 컴파일 옵션으로 지정한 include 폴더에서 찾도록 지시



■ 헤더파일?

▶ Q1) <cstring> 파일에 strcpy() 함수의 코드가 있는가?

▶ strcpy() 함수의 코드가 들어 있다. → X

▶ strcpy() 함수의 원형이 들어 있다. → O

▶ Q2) 그렇다면, strcpy() 함수의 코드는 어디에 있는가?

▶ 함수의 코드는 컴파일된 바이너리 코드로, Visual studio가 설치된 lib 폴더에 libcmt.lib 파일에 들어 있고, 링크 시에 strcpy() 함수의 코드가 exe에 들어간다.

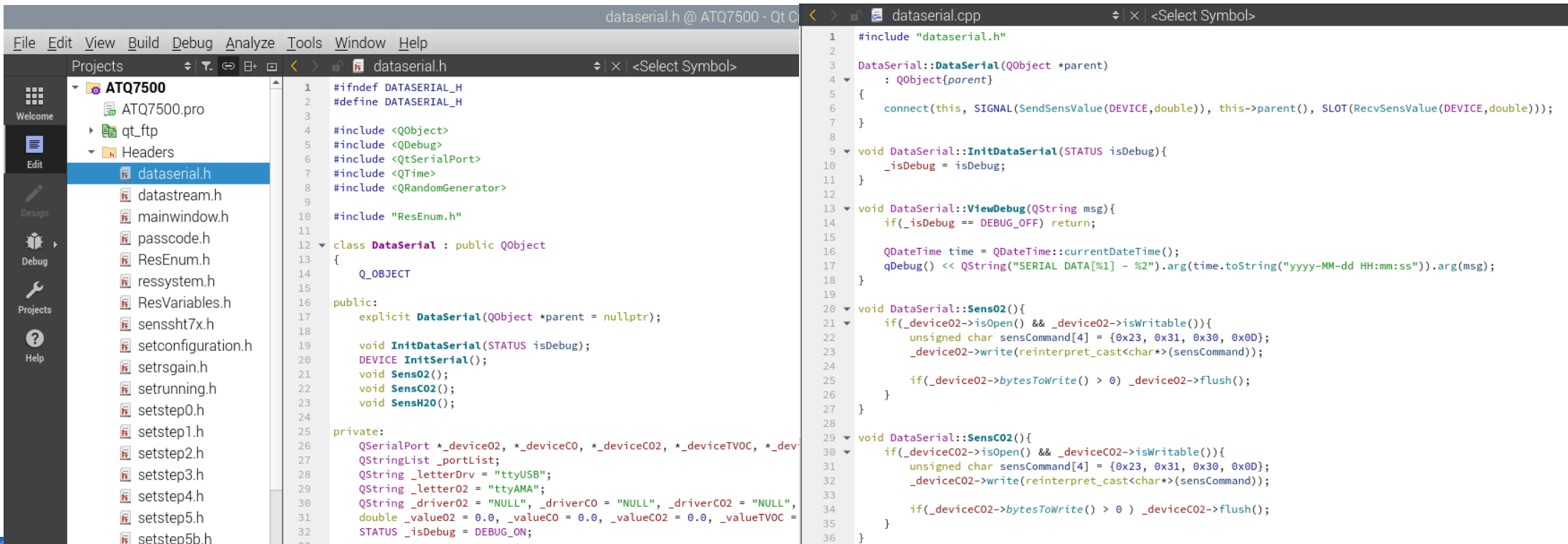
▶ Q3) 그러면, 왜 헤더를 사용하는가?

▶ 사용자 프로그램에서 strcpy() 함수를 호출하는 구문이 정확한지 확인하기 위해 컴파일러에 의해 필요

헤더파일의 추가내용

▶ 헤더 파일은 왜 사용하는가?

- ▶ 개발자마다 성향은 다르다, 교수(나)를 기준으로, 해당 namespace 또는 class에서 사용되는 모든 변수, 함수 들은 보통 헤더파일에 집어 넣고, *.cpp에서 구현한다.(단, 앞서 이야기 했듯이 개발자마다 성향이 다르므로 참고)



The screenshot displays the Qt Creator IDE with two files open: `dataserial.h` and `dataserial.cpp`.

dataserial.h (Header File):

```
1 #ifndef DATASERIAL_H
2 #define DATASERIAL_H
3
4 #include <QObject>
5 #include <QDebug>
6 #include <QtSerialPort>
7 #include <QTime>
8 #include <QRandomGenerator>
9
10 #include "ResEnum.h"
11
12 class DataSerial : public QObject
13 {
14     Q_OBJECT
15
16 public:
17     explicit DataSerial(QObject *parent = nullptr);
18
19     void InitDataSerial(STATUS isDebug);
20     DEVICE InitSerial();
21     void Sens02();
22     void SensC02();
23     void SensH20();
24
25 private:
26     QSerialPort *_device02, *_deviceC0, *_deviceC02, *_deviceTV0C, *_dev
27     QStringList _portList;
28     QString _letterDrv = "ttyUSB";
29     QString _letter02 = "ttyAMA";
30     QString _driver02 = "NULL", _driverC0 = "NULL", _driverC02 = "NULL",
31     double _value02 = 0.0, _valueC0 = 0.0, _valueC02 = 0.0, _valueTV0C =
32     STATUS _isDebug = DEBUG_ON;
```

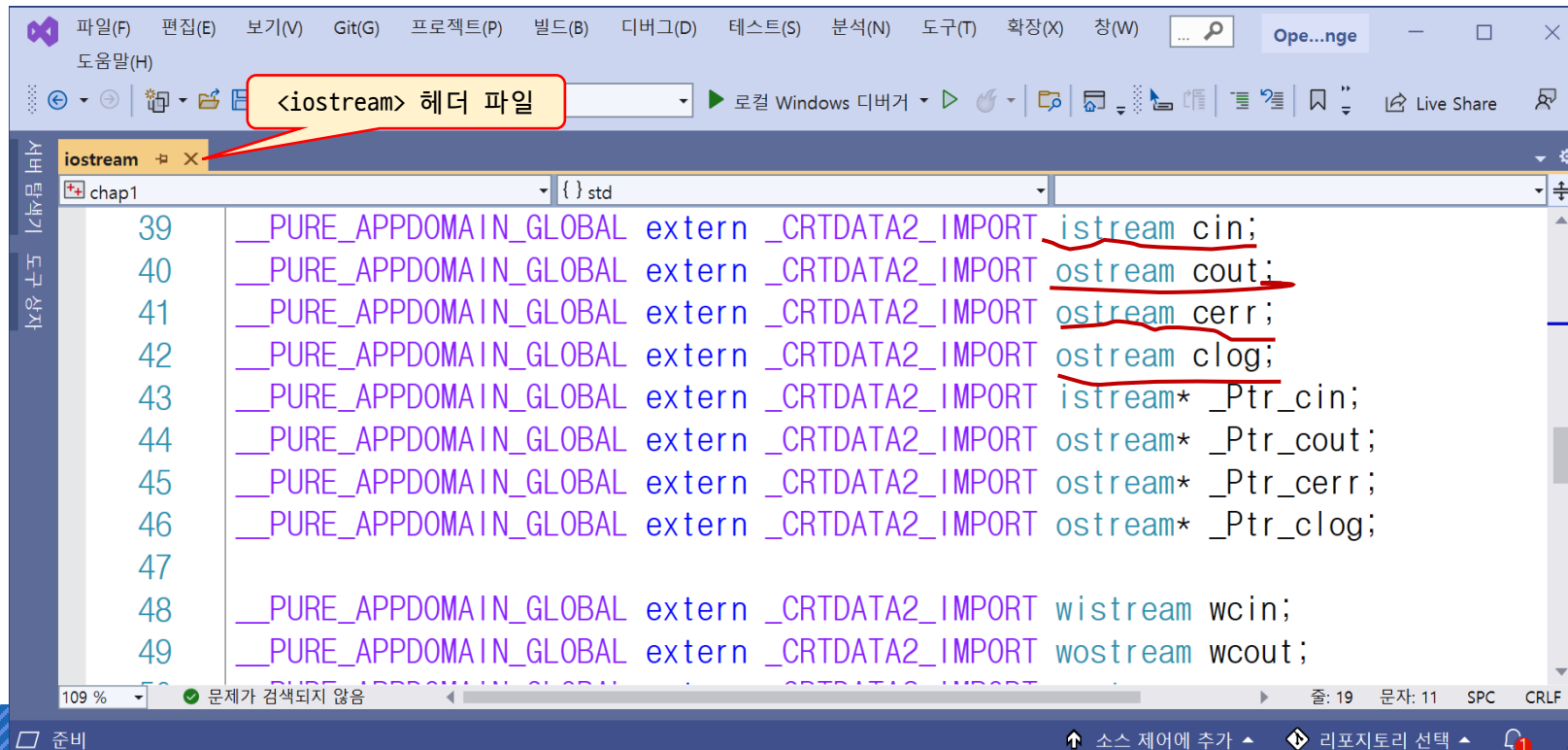
dataserial.cpp (Source File):

```
1 #include "dataserial.h"
2
3 DataSerial::DataSerial(QObject *parent)
4 : QObject{parent}
5 {
6     connect(this, SIGNAL(SendSensValue(DEVICE,double)), this->parent(), SLOT(RecvSensValue(DEVICE,double)));
7 }
8
9 void DataSerial::InitDataSerial(STATUS isDebug){
10     _isDebug = isDebug;
11 }
12
13 void DataSerial::ViewDebug(QString msg){
14     if(_isDebug == DEBUG_OFF) return;
15
16     QDateTime time = QDateTime::currentDateTime();
17     qDebug() << QString("SERIAL DATA[%1] - %2").arg(time.toString("yyyy-MM-dd HH:mm:ss")).arg(msg);
18 }
19
20 void DataSerial::Sens02(){
21     if(!_device02->isOpen() && !_device02->isWritable()){
22         unsigned char sensCommand[4] = {0x23, 0x31, 0x30, 0x00};
23         _device02->write(reinterpret_cast<char*>(sensCommand));
24
25         if(!_device02->bytesToWrite() > 0) _device02->flush();
26     }
27 }
28
29 void DataSerial::SensC02(){
30     if(!_deviceC02->isOpen() && !_deviceC02->isWritable()){
31         unsigned char sensCommand[4] = {0x23, 0x31, 0x30, 0x00};
32         _deviceC02->write(reinterpret_cast<char*>(sensCommand));
33
34         if(!_deviceC02->bytesToWrite() > 0) _deviceC02->flush();
35     }
36 }
```

cin과 cout은 어디에???

▶ <iostream> 헤더에 선언된 객체이다.

- ▶ 따라서, #include <iostream>을 한 프로그램에는 자동으로 cin과 cout이 전역 변수로 선언한 결과가 된다.
- ▶ 프로그램에서 cin과 cout을 바로 사용할 수 있다.



```
39 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT istream cin;
40 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream cout;
41 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream cerr;
42 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream clog;
43 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT istream* _Ptr_cin;
44 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream* _Ptr_cout;
45 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream* _Ptr_cerr;
46 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT ostream* _Ptr_clog;
47
48 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT wistream wcin;
49 __PURE_APPDOMAIN_GLOBAL extern _CRTDATA2_IMPORT wostream wcout;
```




■ #include <iostream>과 std

▶ 실습문제 9 ~ 16 문제