01. c++ 프로그래밍의 기본

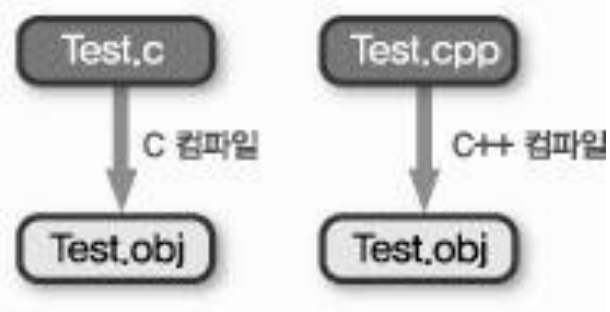
1.1 c++ 기본요소와 화면 출력

1.2 namespace

1.3 입출력

소스파일의 확장자 cpp

- 소스 파일의 확장자
  - C 프로그램의 소스 파일 : *.c 파일
  - C++ 프로그램의 소스 파일 : *.cpp 파일



main() 함수

- main() 함수
  - C++ 프로그램의 실행을 시작하는 함수
    - main() 함수가 종료하면 C++ 프로그램 종료
  - main() 함수의 C++ 표준 모양

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

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

  - main()에서 return문 생략 가능

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

#include <iostream>

- #include <iostream>
  - 전처리기(C++ Preprocessor)에게 내리는 지시
    - <iostream> 헤더 파일을 컴파일 전에 소스에 확장하도록 지시

- <iostream> 헤더 파일
  - 표준 입출력을 위한 클래스와 객체, 변수 등이 선언됨
    - iostream 클래스 선언
      - iostream : 문자 단위로 입출력을 동시에 할 수 있는 스트림 클래스
    - cout, cin, <<, >> 등 연산자 선언

```
#include <iostream>
....
   std::cout << "Hello\n";
   std::cin >> width;
```

    - <iostream.h> 헤더파일
      - 구버전의 C++ 표준에서 사용하던 헤더파일
      - 구버전 컴파일러에서 사용

stream

- 스트림(stream)
  - 데이터의 흐름, 혹은 데이타를 전송하는 소프트웨어 모듈
    - 흐르는 시내와 유사한 개념
  - **스트림의 양 끝에는 프로그램과 장치 연결**
    - 보낸 순서대로 데이터 전달
    - 입출력 기본 단위 : 바이트

- C++ 스트림 종류
  - 입력 스트림
    - 입력 장치, 네트워크, 파일로부터 데이터를 프로그램으로 전달하는 스트림
  - 출력 스트림
    - 프로그램에서 출력되는 데이터를 출력 장치, 네트워크, 파일로 전달하는 스트림

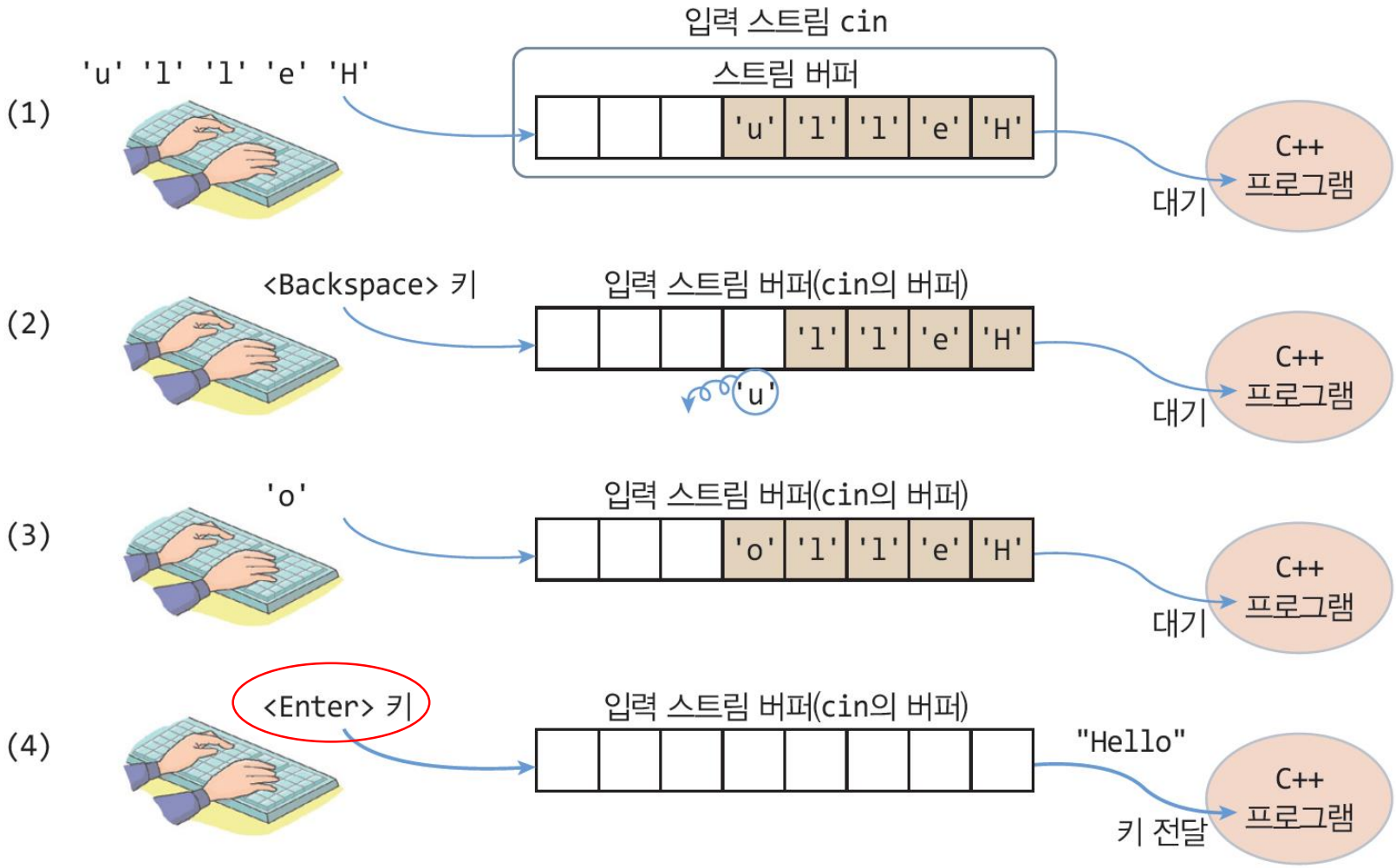
C++ 입출력 스트림 버퍼

- C++ 입출력 스트림은 버퍼를 가짐.

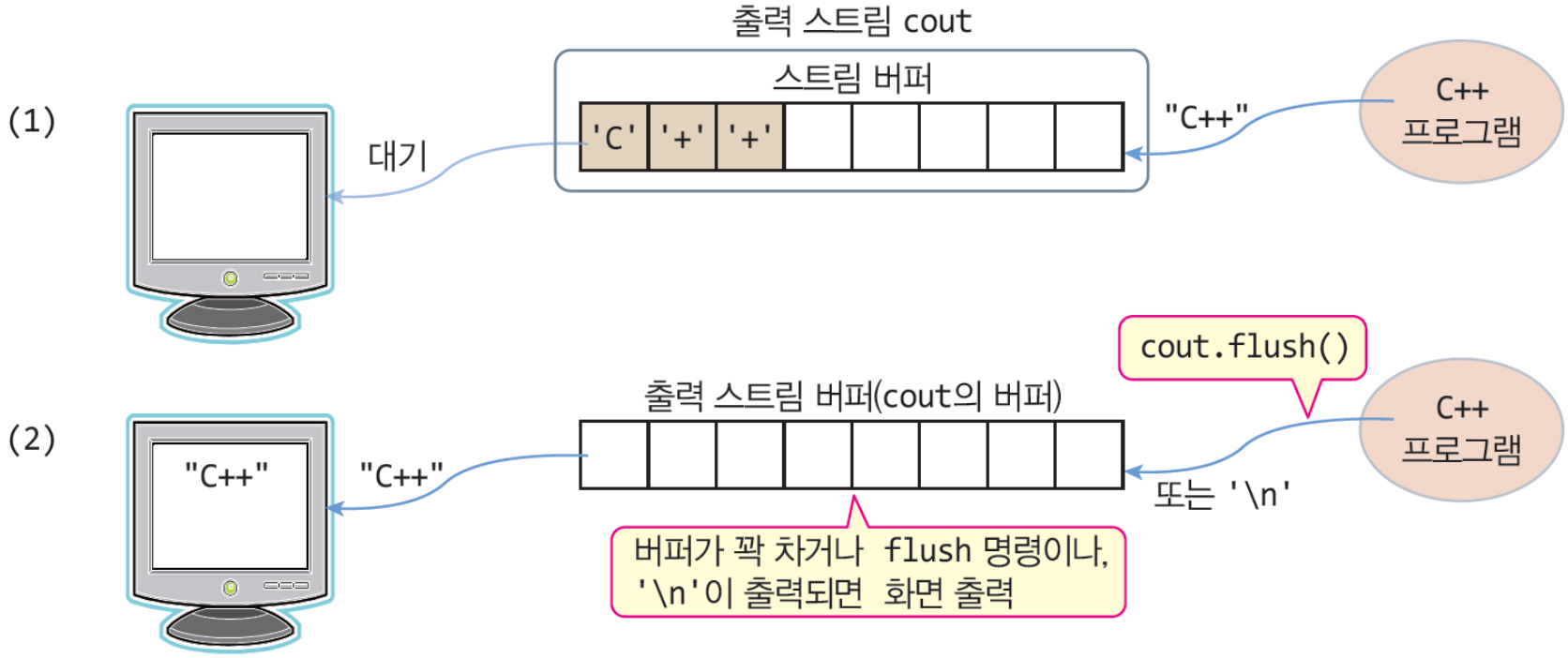
- 키 입력 스트림의 버퍼
  - 목적
    - 입력장치로부터 입력된 데이터를 프로그램으로 전달하기 전에 일시 저장
    - 키 입력 도중 수정 가능
      - <backspace> 키가 입력되면 이전에 입력된 키를 버퍼에서 지움
  - 프로그램은 사용자의 키 입력이 끝난 시점에서 읽음
    - <Enter> 키 : 키 입력의 끝을 의미
    - <Enter> 키가 입력된 시점부터 키 입력 버퍼에서 프로그램이 읽기 시작

- 스크린 출력 스트림 버퍼
  - 목적
    - 프로그램에서 출력된 데이터를 출력 장치로 보내기 전에 일시 저장
    - 출력 장치를 반복적으로 사용하는 비효율성 개선
  - 버퍼가 꽉 차거나 강제 출력 명령 시에 출력 장치에 출력

키 입력 스트림과 버퍼의 역할



스크린 출력 스트림과 버퍼의 역할



C++ 표준은 스트림 입출력만 지원

- 2가지 형태의 입출력 방식
  - **스트림 입출력 방식(stream I/O)**
    - 스트림 버퍼를 이용한 입출력 방식
    - 입력된 키는 버퍼에 저장
      - <Enter>키가 입력되면 프로그램이 버퍼에서 읽어가는 방식
    - 출력되는 데이터는 일차적으로 스트림 버퍼에 저장
      - 버퍼가 꽉 차거나, '\n'을 만나거나, 강제 출력 명령의 경우에만 버퍼가 출력 장치에 출력
  - 저 수준 입출력 방식(raw level console I/O)
    - 키가 입력되는 즉시 프로그램에게 키 값 전달
      - <backspace>키 그 자체도 프로그램에게 바로 전달
      - 게임 등 키 입력이 즉각적으로 필요한 곳에 사용
    - 프로그램이 출력하는 즉시 출력 장치에 출력
    - 컴파일러마다 다른 라이브러리나 API 지원 : C++ 프로그램의 호환성 낮음

- C++ 표준은 스트림 방식만 지원
  - 스트림 입출력은 모든 표준 C++ 컴파일러에 의해 컴파일 됨.
  - 높은 호환성

출력객체인 cout을 이용한 화면 출력

- C++에서의 출력 - 출력객체인 **cout** 이용
  - "cout << 값" 형식으로 출력
    - << 연산자 다음에 기본형의 변수를 지정하면 자동으로 데이터 형에 맞추어 출력.
    - 정수, 실수, 문자열 출력 가능

```
std::cout << "Hello\n"
```

- cout 객체
  - **스크린** 출력 장치에 연결된 **표준** C++ 출력 스트림 객체
  - <iostream> 헤더 파일에 선언. std 이름 공간에 선언: **std::cout**으로 사용

- << 연산자
  - 스트림 삽입 연산자(stream insertion operator)
    - C++ 기본 산술 시프트 연산자(<<)가 스트림 삽입 연산자로 재정의됨
    - ostream 클래스에 구현됨. 오른쪽 피연산자를 왼쪽 스트림 객체에 삽입
    - cout 객체에 연결된 화면에 출력
  - 여러 개의 << 연산자로 여러 값 출력

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

출력객체인 cout을 이용한 화면 출력

- 문자열 및 기본 타입의 데이터 출력
  - bool, char, short, int, long, float, double 타입 값 출력

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

#5.5-3hello1

  - 연산식뿐 아니라 함수 호출도 가능

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

- 출력 형식 조정

```
std::cout << 123;                    // 123 출력
std:: cout.width(10);                //출력할 내용의 폭 지정
                                      //10칸 확보, 오른쪽 정렬, 남는 공간은 빈 공백
std:: cout << 123;                   // _______123 출력 : 앞 쪽에 빈칸 7개 출력

std:: cout.setf(ios_base::left);  //왼쪽 정렬
std:: cout << 123                    // 123_______ 출력 : 뒤쪽에 빈칸 7개 출력
```

출력객체인 cout을 이용한 화면 출력

- 다음 줄로 넘어가기
  - '\n`
    - << 연산자가 '\n' 문자를 cout의 스트림 버퍼에 단순히 삽입
  - endl 조작자
    - <iostream>헤더파일에 있는 함수. << 연산자가 호출
    - '\n'을 cout의 스트림 버퍼에 넣고,
      cout에게 현재 스트림 버퍼에 있는 데이터를 즉각 장치에 출력하도록 지시

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

cout과 <<를 이용한 화면 출력

```
#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 1로 출력됨

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

namespace 개념

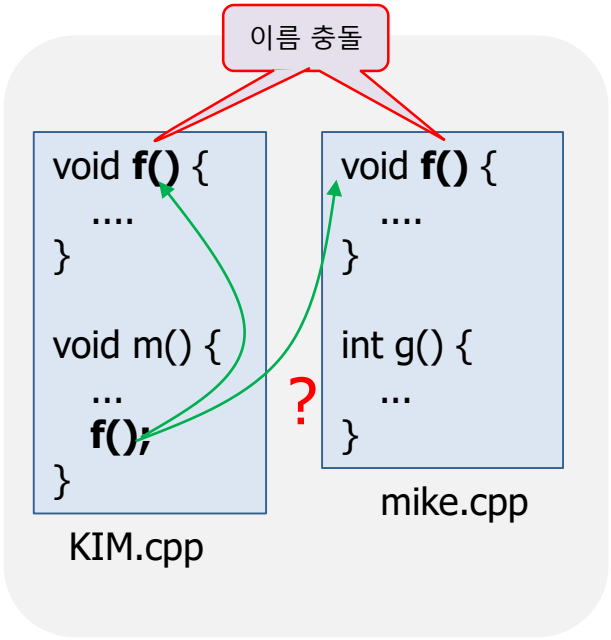
- 이름(identifier) 충돌이 발생하는 경우
  - 여러 명이 서로 나누어 프로젝트를 개발하는 경우
  - 오픈 소스 혹은 다른 사람이 작성한 소스나 목적 파일을 가져와서 컴파일 하거나 링크하는 경우
  - 해결하는데 많은 시간과 노력이 필요

- namespace 키워드
  - 이름 충돌 해결
    - 2003년 새로운 ANSI C++ 표준에서 도입
  - 개발자가 자신만의 이름 공간을 생성할 수 있도록 함
    - 이름 공간 안에 선언된 이름은 다른 이름 공간과 별도 구분

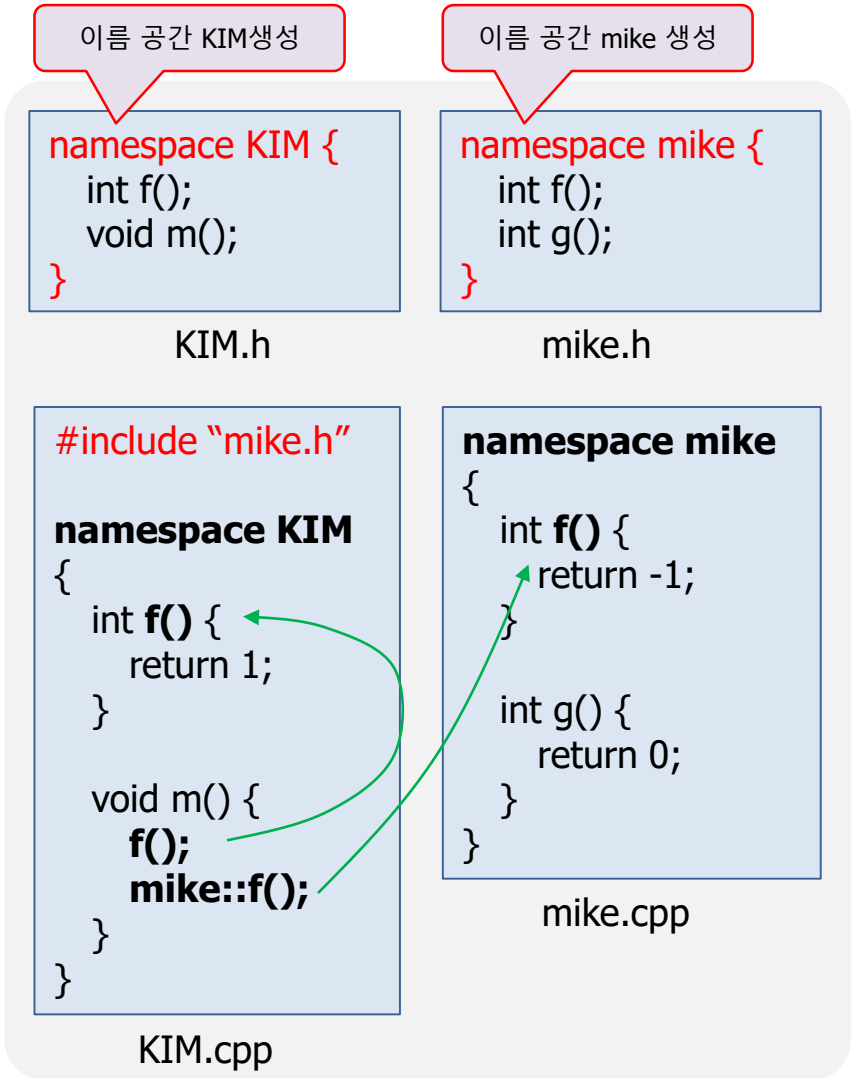
```
namespace KIM { // KIM 이라는 이름 공간 생성
   ..... // 이 곳에 선언된 모든 이름은 KIM 이름 공간에 생성된 이름
}
```

  - 이름 공간 사용
    - 이름 공간 :: 이름

namespace의 사용



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



(b) 이름 공간을 사용하여 f() 함수 이름의 충돌 문제 해결

std:: 란?

- **std**
  - ANSI C++ 표준에서 정의한 **이름 공간(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

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

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

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

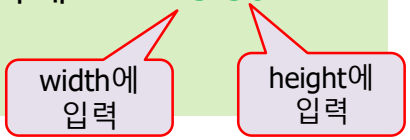
```
#include <iostream>
using namespace std;
```

cin을 이용한 키 입력

- C++에서의 입력 - 입력 객체인 **cin** 이용
  - "cin >> 변수" 형식으로 입력
    - >> 연산자 다음에 기본형의 변수를 지정하면 자동으로 다양한 데이터 형에 맞추어 입력 처리
      - 입력받을 값의 형식 지정할 필요 없음 → 변수의 데이터 형에 따라 입력

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

```
너비와 높이를 입력하세요>>23 36
23
36
```



- >> 연산자
  - 스트림 추출 연산자(stream extraction operator)
    - C++ 산술 시프트 연산자(>>)가 <iostream> 헤더 파일에 스트림 추출 연산자로 재정의됨
    - 입력 스트림에서 값을 읽어 변수에 저장
  - 연속된 >> 연산자를 사용하여 여러 값 입력
    - 단, 문자열을 입력할 때는 빈칸 없이 입력해야 한다.
      - >> 연산자는 빈칸이 입력될 때까지를 하나의 입력으로 처리하므로 빈칸을 포함한 문자열을 입력 받으려면 별도의 방법을 사용해야 한다.

cin과 >>를 이용한 키 입력

- C++ 프로그램에서 키 입력 받기

```
#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
}
```

'\n'도 가능

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

<Enter> 키를 칠 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

cin으로부터의 키 입력 오류

- 입력 오류가 발생하면?
  - 입력받을 데이터보다 더 큰 데이터나, 자료형이 다른 데이터를 입력받을 경우
    - 입력버퍼에 쓸데없는 데이터가 남게 됨.
    - (예) int형의 변수 a에 문자 'A'를 입력하면 문자 'A'가 버퍼에 남음

- 해결책
  - cin객체 내부상태 초기화(입력오류 flag 초기화), 입력 버퍼 초기화

cin으로부터의 키 입력 오류

- **cin.fail()**
  - cin 오류시 1을 반환하고 아니면 0을 반환

- **cin.clear()**
  - cin 객체의 내부 상태 flag를 초기화시켜 cin 관련 기능이 정상 작동하도록 함.
  - 오류 사실 제거

- **cin.ignore(최대확인글자수, '문자')** : 입력 버퍼 지우기
  - 최대 수만큼 버퍼에 있는 문자를 지우고, 혹시 '문자'가 나타나면 '문자'까지 지우고 스톱

```
cin >> val;

if (cin.fail() == 1) {   // I/O 오류가 발생하면
    cin.clear(); // 모든 오류비트 clear
    cin.ignore(256, '\n');   // 최대 256개의 문자 무시가능, 개행문자를 만나면 추출을 멈춤
}
```

변수의 정의 위치

- C++의 변수 선언
  - **C++에서는 변수의 정의는 변수가 사용되기 전에만 하면 됨.**

실행문 중간에 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 띈다.
```

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;
```

변수의 정의 위치에 따른 장단점

- C++ 변수 선언 방식의 장점
  - C에서와 같이 변수 선언부과 실행 문 사이를 왔다 갔다 하는 번거로움 해소
  - 변수를 사용하기 직전 선언함으로써 변수 이름에 대한 타이핑 오류 줄임

- C++ 변수 선언 방식의 단점
  - 선언된 변수를 일괄적으로 보기 힘듦
  - 코드 사이에 있는 변수 찾기 어려움

변수의 정의 예

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

int main()
{
    // 변수의 정의
    int a, b;

    a = 100; // 변수 a에 숫자 대입

    // 이곳에 아주 많은 코드가 있다고 가정..
    // 코드, 코드, 코드…

    b = a; // b에 a의 값을 대입

    return 0;
}
```

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

int main()
{
    // 변수의 정의 1
    int a;

    a = 100; // 변수 a에 숫자 대입

    // 이곳에 아주 많은 코드가 있다고 가정
    // 코드, 코드, 코드…

    // 변수의 정의2
    int b;
    b = a; // b에 a의 값을 대입

    return 0;
}
```

C++ 문자열

- C++의 문자열 표현 방식 : 2가지
  - **C-스트링 방식** – '\0'로 끝나는 문자 배열
    - 문자열의 저장 : 배열 또는 동적으로 메모리 할당

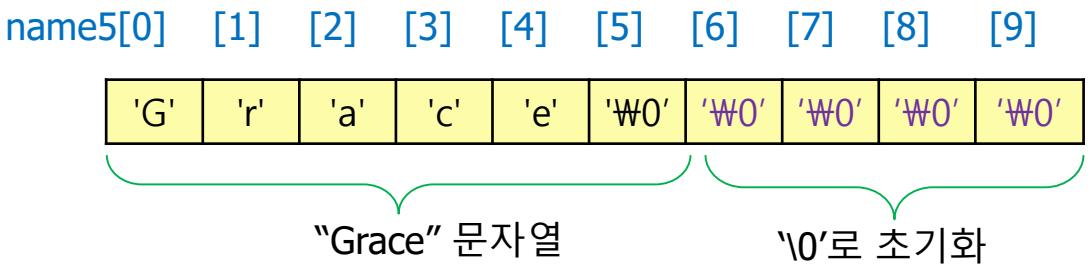
C-스트링 문자열

```
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";
```



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

C 언어에서 사용한 문자열 라이브러리

- C-스트링으로 문자열 다루기
  - C 언어에서 사용한 함수 사용 가능
    - strcmp(), strlen(), strcpy() 등
  - <cstring>이나 <string.h> 헤더 파일 include

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

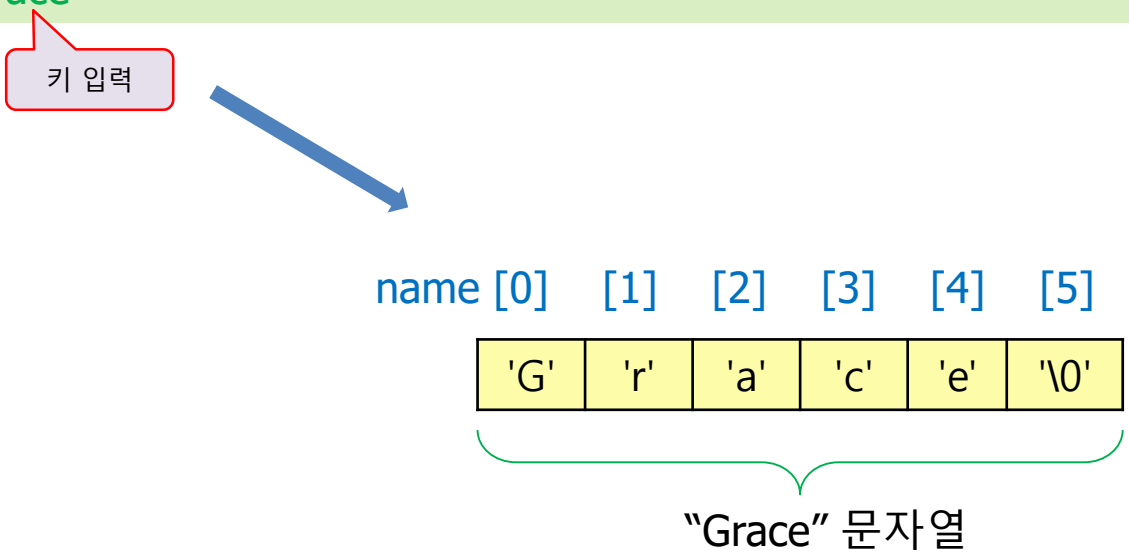
  - <cstring> 헤더 파일을 사용하는 것이 바람직함
    - <cstring>이 C++ 표준 방식

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



키보드에서 문자열 입력 받고 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

cin.getline()을 이용한 문자열 입력

- **공백이 낀 문자열**을 입력 받는 방법

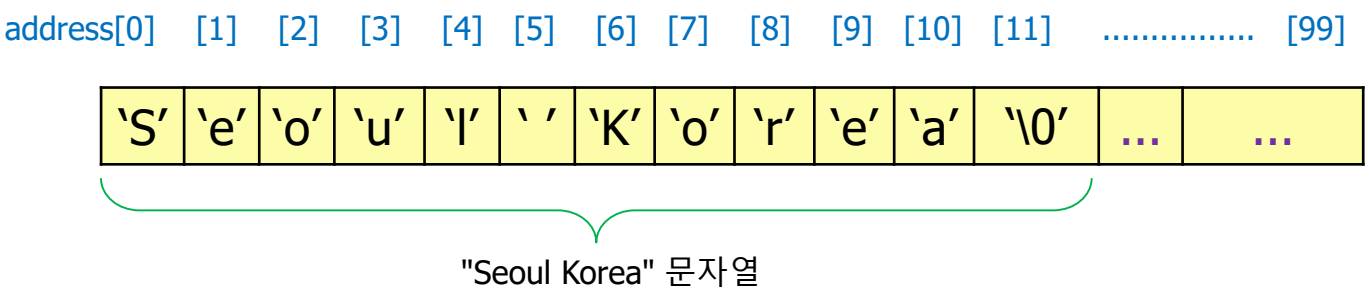
cin.getline(char buf[], int size, char delimitChar)

- buf에 최대 size-1개의 문자 입력. 끝에 '\0' 붙임
- delimitChar를 만나면 입력 중단. 끝에 '\0' 붙임
  - delimitChar의 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,



cin.getline()을 이용한 문자열 입력

- getline()함수 : 한줄(개행문자까지 모두) 단위로 읽어 들임.
  - 기본적으로 한 줄을 읽어 들임. 여기서, 한줄의 구분은 enter키(개행문자, 새줄문자, \n)까지를 한 줄로 인식함.
  - 입력된 문자열은 마지막에 있는 개행문자(엔터키, \n)는 저장이 되지 않음. 대신 이 자리에 '\0'라고 하는 NULL문자가 저장되어 문자열이 완성됨.

```
#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 입력될 때까지 하나의 문자열로 인식

```
string str1, str2;

getline(cin, str1);
getline(cin, str2, 'i');

cout << "str1 = " << str1 << "\n";
cout << "str1 = " << str1 << "\n";
```

```
John Smith 엔터 John Smith
str1= John Smith
str2= John Sm
```

C++에서 문자열을 다루는 string 클래스

- 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> 필요
  - C-스트링보다 다루기 쉬움

string 클래스를 이용한 문자열 입력 및 다루기

```
#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] << ")?";  // [] 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

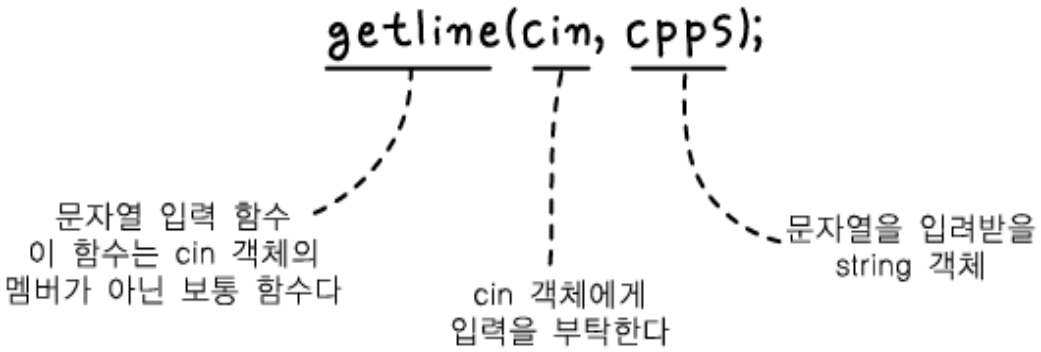
getline()은 string 타입의 문자열을 입력 받기 위해 제공되는 전역 함수

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

빈칸 포함

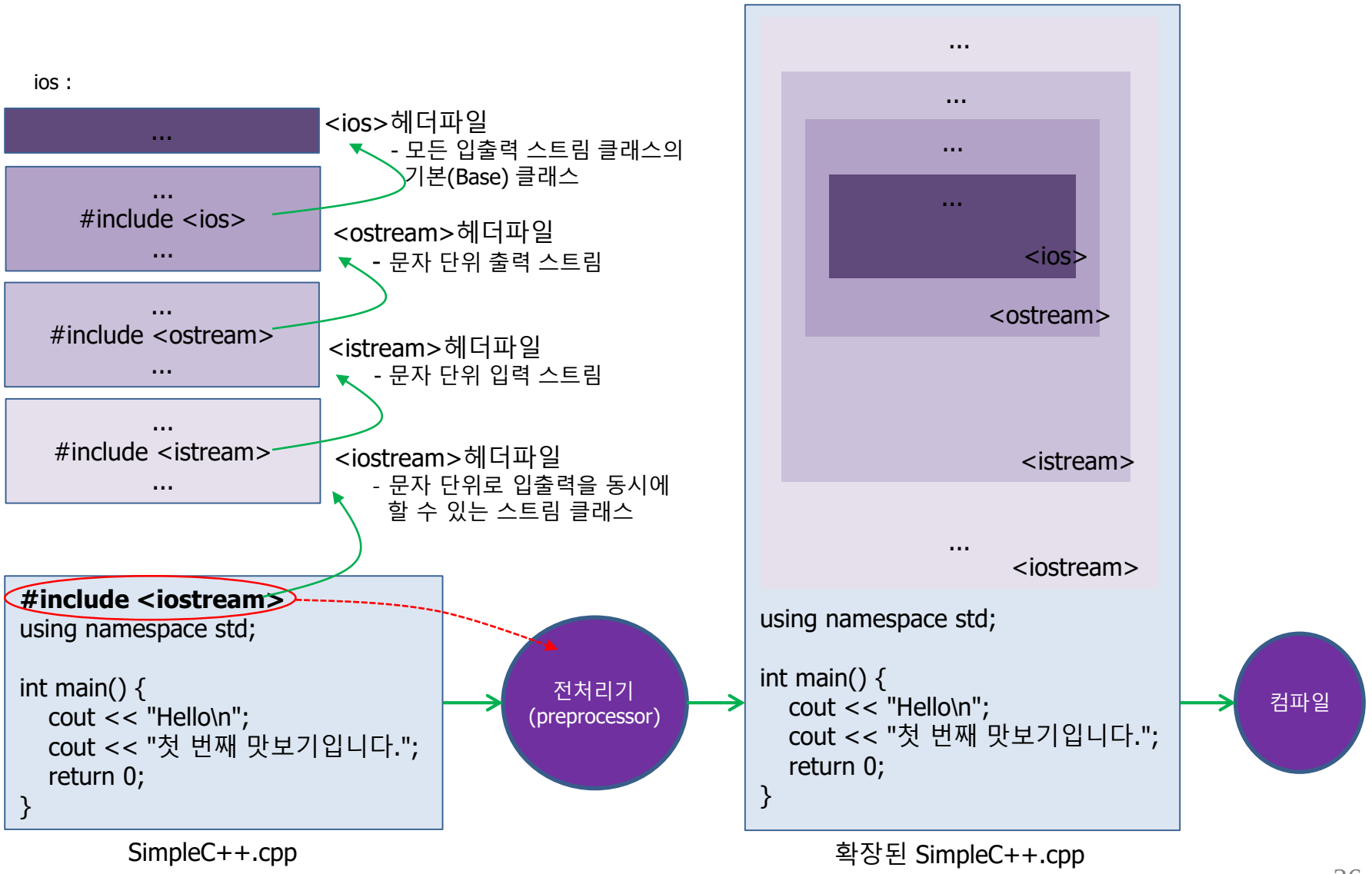
getline() 함수 사용법

- getline() 함수를 C++ 스타일의 문자열과 함께 사용하는 방법



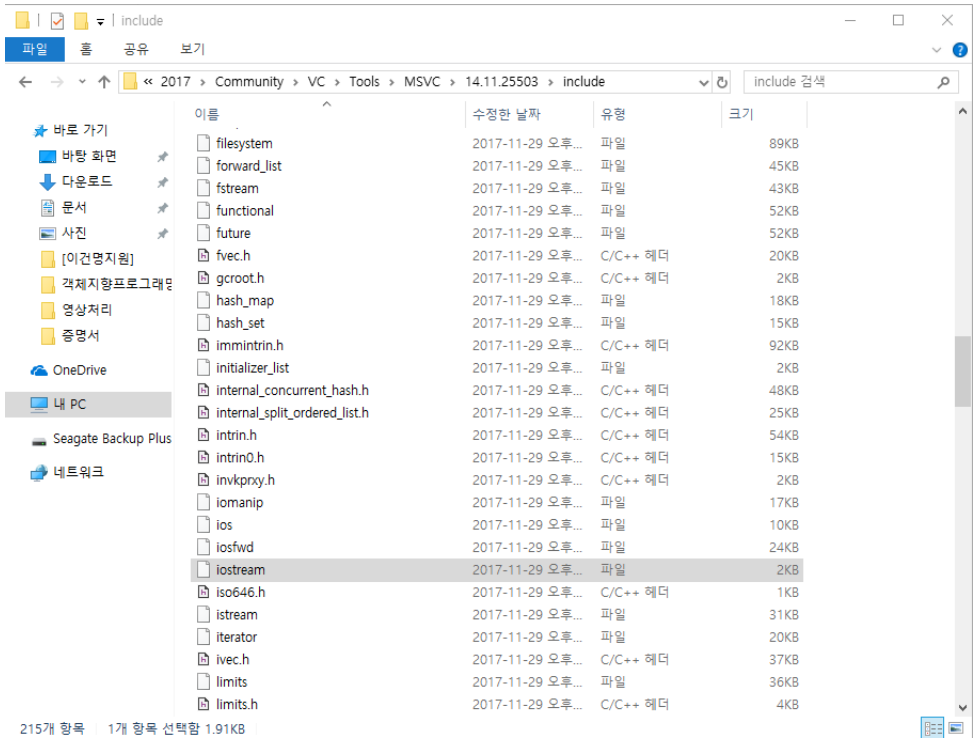
- getline() 함수를 C 스타일의 문자열과 함께 사용하는 방법

#include <iostream>와 전처리기



<iostream> 헤더 파일은 어디에?

- <iostream> 파일은 텍스트 파일
- 컴파일러가 설치된 폴더 아래 include 폴더에 존재
  - C:\Program Files(x86) \Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.25.286103\include



<iostream>과 <iostream.h>의 차이

- 구 표준의 C++ `#include <iostream.h>`
  - 구 버전 컴파일러(Visual Studio 6.0)에서 사용

- 2003년 표준 이후 버전의 C++

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

- 헤더 파일의 확장자 비교

언어	헤더 파일 확장자	사례	설명
C	.h	〈string.h〉	C/C++ 프로그램에서 사용 가능
구 버전 C++ 표준	.h	〈string.h〉	구 버전의 C++ 컴파일러에서 사용
2003년 이후 신 C++ 표준	확장자 없음	〈cstring〉	using namespace std;와 함께 사용해야 함

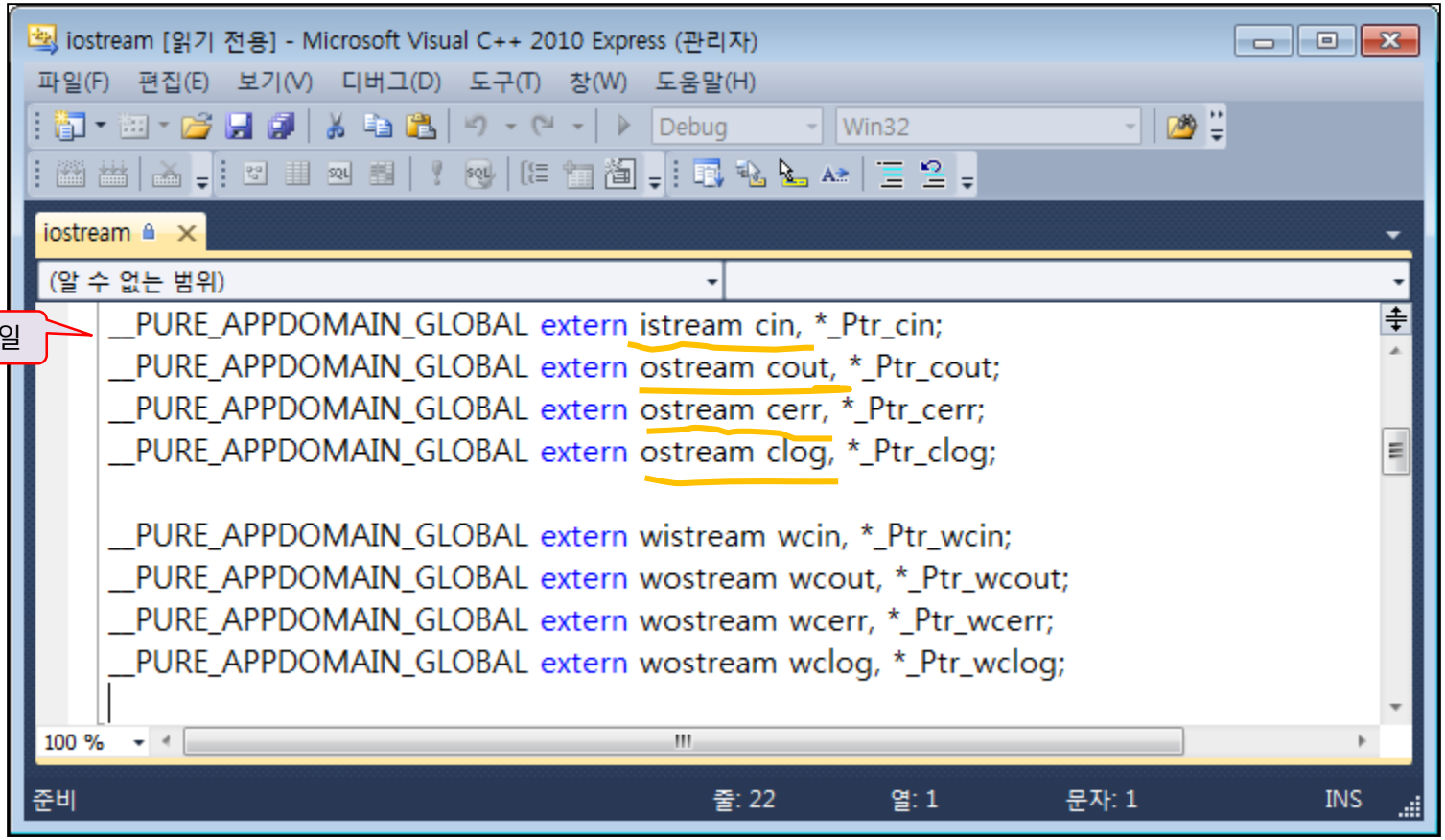
#include <헤더파일>와 #include "헤더파일"

- #include <헤더파일>
  - '헤더파일'을 찾는 위치 : 컴파일러가 설치된 폴더
  - #include <iostream>
    - <iostream> 헤더 파일은 컴파일러가 제공

- #include "헤더파일"
  - '헤더파일'을 찾는 위치 : 현재 디렉토리
  - 직접 만든 헤더를 사용할 때

cin과 cout은 어디에 선언되어 있는가?

- cout이나 cin은 모두 <iostream>에 선언된 객체



다음 수업

- 포인터와 레퍼런스
  - 1_ 포인터
  - 2_ 레퍼런스(참조)
  - 3_ 함수에서의 인자 전달