



# 고급프로그래밍

## 입출력

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory  
Kwangwoon University  
(02-940-5625, mhjeong@kw.ac.kr)

# Schedule

주차	주제		과제	퀴즈
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++	C++ 프로그래밍의 기본 (3), 클래스와 객체 (4)	1	1
3		객체생성과 사용 (5)	2	2
4		함수와 참조 (6, 3/26), 복사 생성자와 함수중복(7)	3	3
5		static, friend, 연산자중복 (8, 4/2), 연산자중복 상속(9)	4	
6		상속 (10, 4/9), 가상함수와 추상클래스 (11)		4
7		템플릿과 STL (12, 4/16), 입출력(13)	5	
8		중간고사		
9		파일 입출력, 예외처리 및 C 사용, 람다식		
10		멀티스레딩		
11		멀티스레딩, 고급문법		
12		고급문법		
13	병렬프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15		기말고사		

# 오늘의 학습내용

- 표준 입출력
- 포맷 입출력

표준 입출력

# 스트림

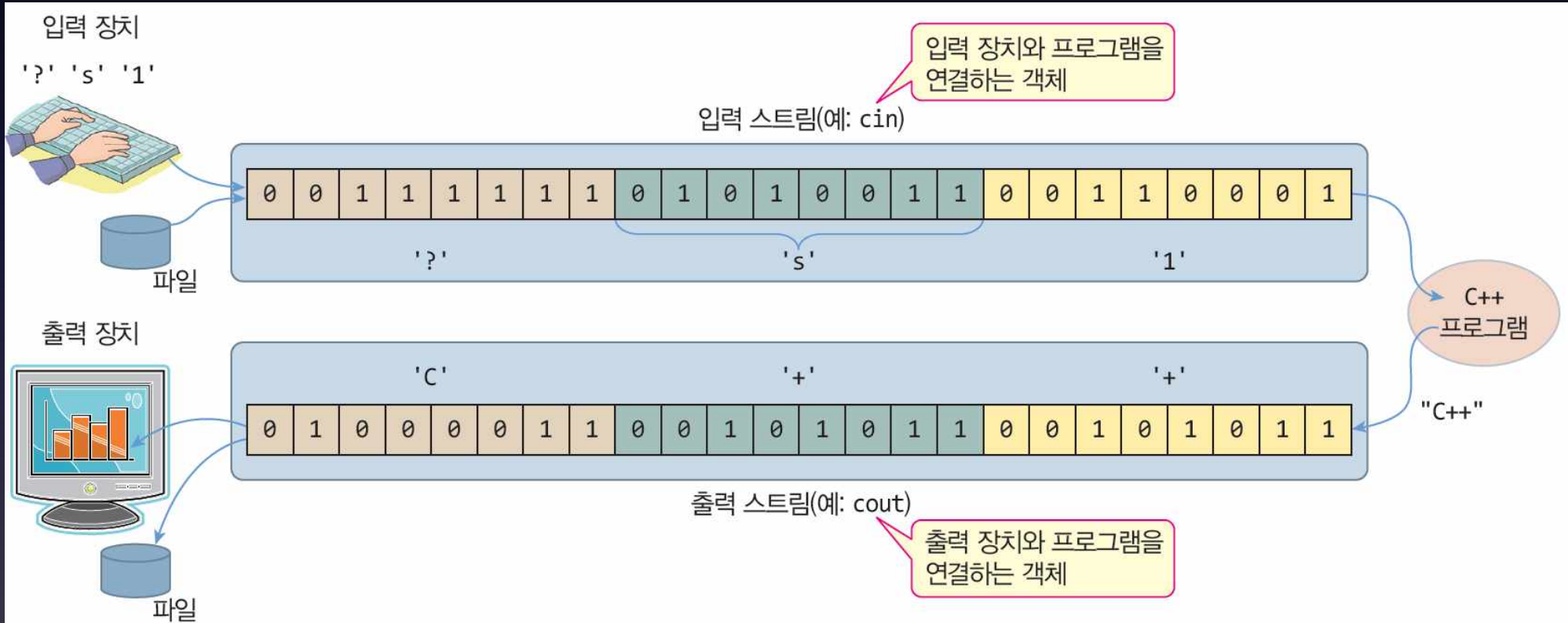
## ■ 스트림(stream)

- 데이터의 흐름, 혹은 데이터를 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



# 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

# 입출력 방식

## ■ 입출력 방식 2가지

### - 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 컴파일됨
- 높은 호환성



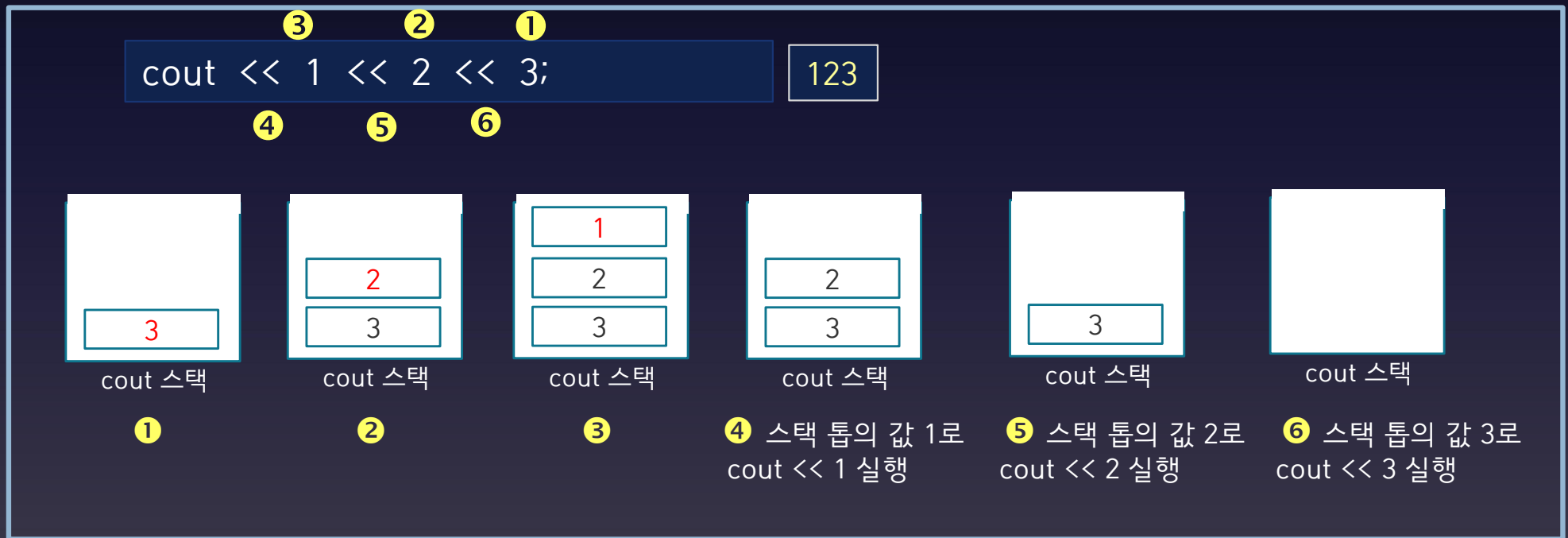
# 입출력 클래스 소개

클래스	설명
ios	모든 입출력 스트림 클래스들의 기본(Base) 클래스. 스트림 입출력에 필요한 공통 함수와 상수, 멤버 변수 선언
istream, ostream, iostream	istream은 문자 단위 입력 스트림. ostream은 문자 단위 출력 스트림. iostream은 문자 단위로 입출력을 동시에 할 수 있는 스트림 클래스
ifstream, ofstream, fstream	파일에서 읽고 쓰는 기능을 가진 파일 입출력 스트림 클래스. 파일에서 읽을 때는 ifstream 클래스를, 파일에 쓸 때는 ofstream 클래스를, 읽고 쓰기를 동시에 할 때 fstream 클래스 이용

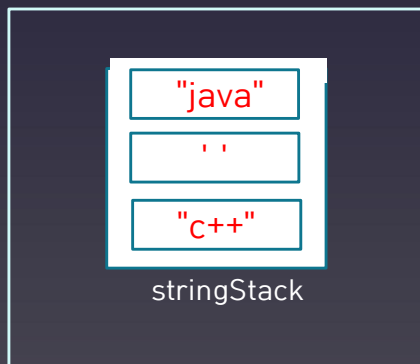
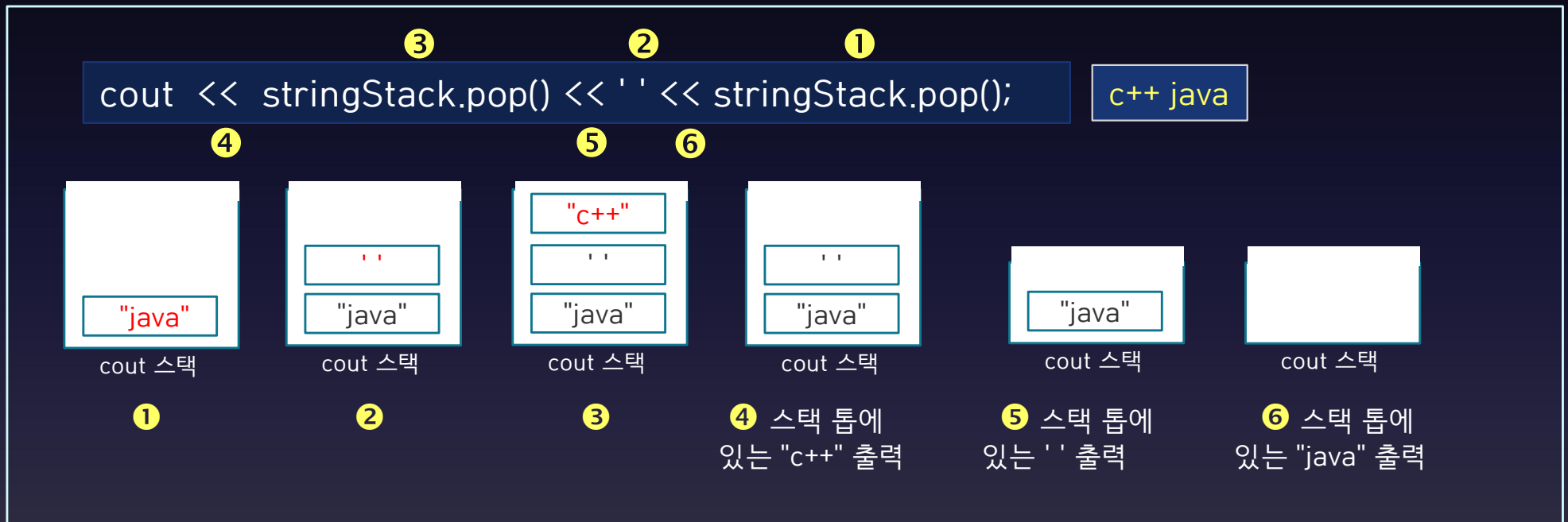
# 표준 입출력 스트림 객체

- C++ 프로그램이 실행될 때 자동으로 생겨나는 스트림
  - cin
    - istream타입의 스트림 객체로서 키보드 장치와 연결
  - cout
    - ostream타입의 스트림 객체로서 스크린 장치와 연결
  - cerr
    - ostream타입의 스트림 객체로서 스크린 장치와 연결
    - 오류 메시지를 출력할 목적
    - 스트림 내부 버퍼 거치지 않고 출력
  - clog
    - ostream타입의 스트림 객체로서 스크린 장치와 연결
    - 오류 메시지를 출력할 목적
    - 스트림 내부에 버퍼 거쳐 출력

# cout << a << b << c;의 실행 순서



# cout << a << b << c;의 실행 순서



# ostream 멤버 함수

- `ostream& put(char ch)`  
ch의 문자를 스트림에 출력
- `ostream& write(char* str, int n)`  
str 배열에 있는 n개의 문자를 스트림에 출력
- `ostream& flush()`  
스트림 버퍼에 있는 내용을 스트림에 강제 출력함

# 예제

- ostream의 put(), write() 멤버 함수를 이용하여 문자를 화면에 출력하는 사례를 보여준다.

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

int main() {
    // "Hi!"를 출력하고 다음 줄로 넘어간다.
    cout.put('H');
    cout.put('i');
    cout.put(33);
    cout.put('\n');

    // "C++ "을 출력한다.
    cout.put('C').put('+').put('+').put(' ');

    char str[]="I love programming";
    cout.write(str, 6); // str 배열의 6 개의 문자 "I love"를 스트림에 출력
}
```

ASCII 코드 33은 '!' 문자임

put() 메소드를 연결하여 사용할 수 있다.

```
Hi!
C++ I love
```

# istream 멤버 함수

## ■ int get()

- 입력 스트림에서 문자를 읽어 반환, 오류나 EOF를 만나면 반환

## ■ istream& get(char& ch)

- 입력 스트림에서 문자를 읽어 ch에 저장, 현재 입력 스트림 객체(\*this)의 참조 반환
- 오류나 EOF를 만나면 스트림 내부의 오류 플래그(failbit) 설정

- int get()을 이용하여 한 라인의 문자들을 읽는 코드

```
int ch;
while((ch = cin.get()) != EOF) { // EOF 는 -1
    cout.put(ch); // 읽은 문자 출력
    if(ch == '\n')
        break; // <Enter> 키가 입력되면 읽기 중단
}
```

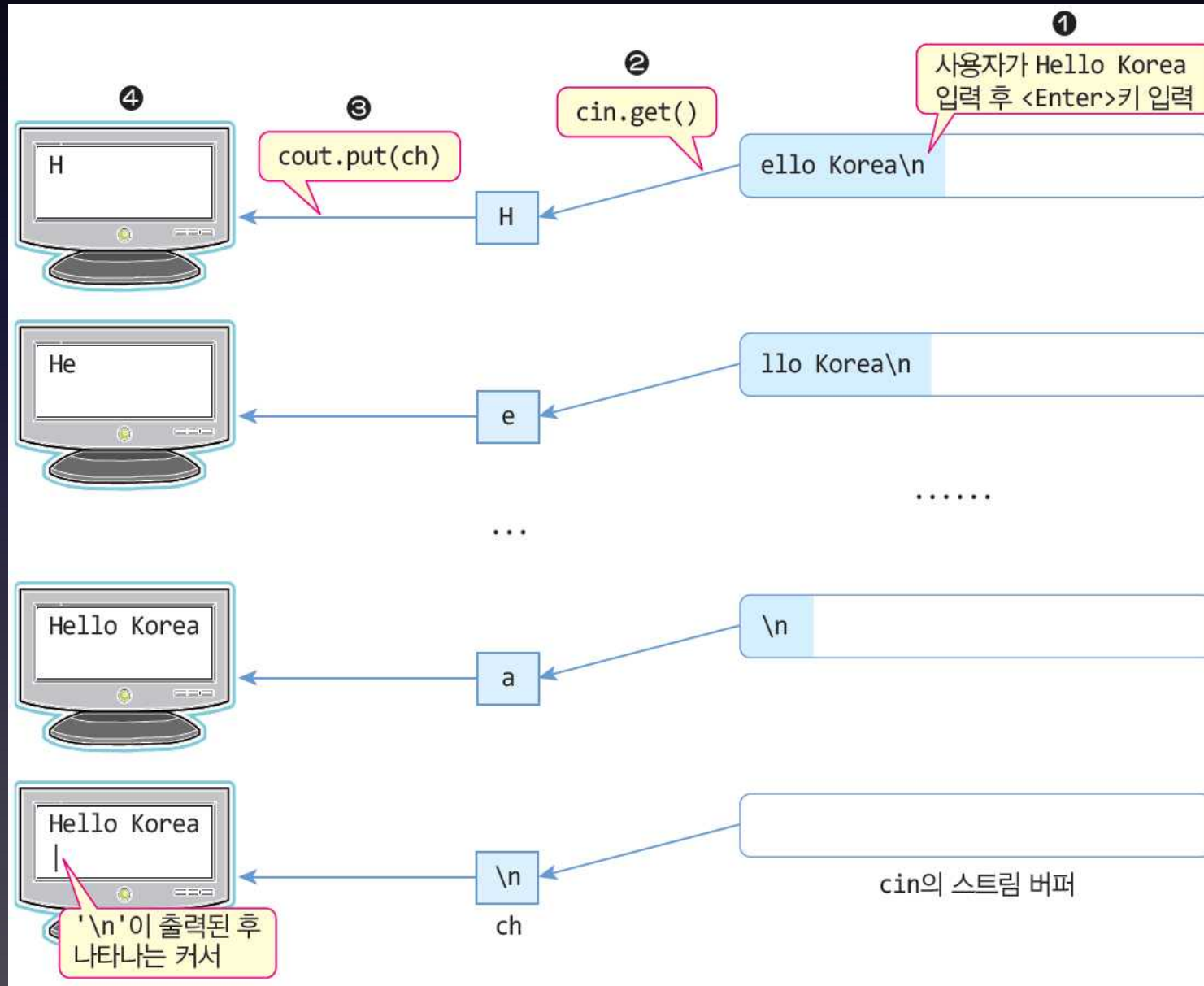
입력 스트림의 끝인지 비교

- istream& get(char& ch)을 이용하여 한 라인의 문자들을 읽는 코드

```
char ch;
while(true) {
    cin.get(ch); // 입력된 키를 ch에 저장하여 리턴
    if(cin.eof()) break; // EOF를 만나면 읽기 종료
    cout.put(ch); // ch의 문자 출력
    if(ch == '\n')
        break; // <Enter> 키가 입력되면 읽기 중단
}
```

입력 스트림의 끝인지 비교

# ch = cin.get()의 실행 사례





# 예제 - get(), get(char&)

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

void get1() {
    cout << "cin.get()로 <Enter> 키까지 입력 받고 출력합니다>>";
    int ch; // EOF와의 비교를 위해 int 타입으로 선언
    while((ch = cin.get()) != EOF) { // 문자 읽기. EOF는 -1
        cout.put(ch); // 읽은 문자 출력
        if(ch == '\n')
            break; // <Enter> 키가 입력되면 읽기 중단
    }
}

void get2() {
    cout << "cin.get(char&)로 <Enter> 키까지 입력 받고 출력합니다>>";
    char ch;
    while(true) {
        cin.get(ch); // 문자 읽기
        if(cin.eof()) break; // EOF를 만나면 읽기 종료
        cout.put(ch); // ch의 문자 출력
        if(ch == '\n')
            break; // <Enter> 키가 입력되면 읽기 중단
    }
}

int main() {
    get1(); // cin.get()을 이용하는 사례
    get2(); // cin.get(char&)을 이용하는 사례
}
```

```
cin.get()로 <Enter> 키까지 입력 받고 출력합니다>>Do you love C++?
Do you love C++?
cin.get(char&)로 <Enter> 키까지 입력 받고 출력합니다>>Yes, I do.
Yes, I do.
```

# 문자열 입력

## ■ `istream& get(char* s, int n)`

- 입력 스트림으로부터  $n-1$ 개의 문자를 읽어 배열 `s`에 저장하고 마지막에 `'\0'` 문자 삽입

```
char str[10];  
cin.get(str, 10); // 최대 9개의 문자를 읽고 끝에 '\0'를 붙여 str 배열에 저장  
cout << str; // str을 화면에 출력
```

- 입력 도중에 `'\n'`을 만나면 `'\0'`을 삽입하고 반환하고 스트림 버퍼에 `'\n'`을 남김
  - 다시 `get()`으로 문자열 읽기를 시도하면 입력 스트림에 남은 `'\n'`키를 읽게 되어 무한 루프에 빠짐
  - `cin.get()`이나 `cin.ignore(1);`를 통해 문자 1개(`'\n'`)를 스트림에서 읽어 버려야 함.

# 예제 – get(char\*, int)

- “exit”이 입력되면 프로그램을 종료하도록 작성하라.

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

int main() {
    char cmd[80];
    cout << "cin.get(char*, int)로 문자열을 읽습니다." << endl;
    while(true) {
        cout << "종료하려면 exit을 입력하세요 >> ";
        cin.get(cmd, 80); // 79개까지의 영어 문자 읽음.
        if(strcmp(cmd, "exit") == 0) {
            cout << "프로그램을 종료합니다....";
            return 0;
        }
        else
            cin.ignore(1); // 버퍼에 남아 있는 <Enter> 키 ('\n') 제거
    }
}
```

'\n'은 입력 스트림  
버퍼에 남겨둠

38개 까지의 한글 무  
자 읽을 수 있음

이 부분을 제거하면  
무한 루프에 빠짐

입력 버퍼에 남아  
있는 '\n' 제거

```
cin.get(char*, int)로 문자열을 읽습니다.
종료하려면 exit을 입력하세요 >> exi
종료하려면 exit을 입력하세요 >> exiT
종료하려면 exit을 입력하세요 >> exito
종료하려면 exit을 입력하세요 >> exit
프로그램을 종료합니다....
```

# 한 줄 읽기

```
istream& get(char* s, int n, char delim='\n')
```

입력 스트림으로부터 최대  $n-1$ 개의 문자를 읽어 배열  $s$ 에 저장하고 마지막에  $\backslash 0$  문자 삽입. 입력  
도중  $delim$ 에 지정된 구분 문자를 만나면 지금까지 읽은 문자를 배열  $s$ 에 저장하고 리턴

```
istream& getline(char* s, int n, char delim='\n')
```

`get()`과 동일. 하지만  $delim$ 에 지정된 구분 문자를 스트림에서 제거

```
char line[80];  
cin.getline(line, 80);
```

```
cin.getline(line, 80);
```

C++ programming language.

C++ programming language.\n

<Enter>키를 입력하면  
읽기 시작됨

.....

.....

C++ programming language.\0

cin의 스트림 버퍼

\n이 없음에 주목

char line[80]

# 예제 - getline()

- istream의 getline()을 이용하여 빈 칸을 포함하는 한 줄을 읽고 다시 그대로 출력하는 프로그램을 작성하라.

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

int main() {
    char line[80];
    cout << "cin.getline() 함수로 라인을 읽습니다." << endl;
    cout << "exit를 입력하면 루프가 끝납니다." << endl;

    int no = 1; // 라인 번호
    while(true) {
        cout << "라인 " << no << " >> ";
        cin.getline(line, 80); // 79개까지의 문자 읽음
        if(strcmp(line, "exit") == 0)
            break;
        cout << "echo --> ";
        cout << line << endl; // 읽은 라인을 화면에 출력
        no++; // 라인 번호 증가
    }
}
```

'\n'은 line에 삽입하지 않고,  
스트림 버퍼에서 제거

```
cin.getline() 함수로 라인을 읽습니다.
exit를 입력하면 루프가 끝납니다.
라인 1 >> It's now or never.
echo --> It's now or never.
라인 2 >> Come hold me tight.
echo --> Come hold me tight.
라인 3 >> Kiss me my darling, be mine tonight.
echo --> Kiss me my darling, be mine tonight.
라인 4 >> 엘비스 프레슬리 노래입니다.
echo --> 엘비스 프레슬리 노래입니다.
라인 5 >> exit
```

# 입력 문자 건너 띄기와 문자 개수

```
istream& ignore(int n=1, int delim=EOF)
```

입력 스트림에서 n개 문자 제거. 도중에 delim 문자를 만나면 delim 문자를 제거하고 리턴

```
int gcount()
```

최근에 입력 스트림에서 읽은 바이트 수(문자의 개수) 리턴. <Enter> 키도 개수에 포함

## - 입력 스트림에서 문자 건너뛰기

```
cin.ignore(10); // 입력 스트림에 입력된 문자 중 10개 제거
```

```
cin.ignore(10, ';'); // 입력 스트림에서 10개의 문자 제거. 제거 도중 ';'을 만나면 종료
```

## - 최근에 읽은 문자 개수 리턴

```
char line[80];
```


```
cin.getline(line, 80);
```

```
int n = cin.gcount(); // 최근의 실행한 getline() 함수에서 읽은 문자의 개수 리턴
```

# 실습

- `int cin.get()` 을 이용하여 키보드로부터 한 라인을 읽고 'a'가 몇 개인지 출력하는 프로그램을 완성하시오.

```
#include <iostream>
using namespace std;
int main()
{
    int count = 0, ch;

    while((ch=cin.get()) != EOF)
    {
        
    }
    cout << "a 문자는 총 " << count << "개 입니다." << endl;
}
```

포맷 입출력



# ios 클래스에 정의된 포맷 플래그

플래그	값	의미
ios::skipws	0x0001	입력시 공백 문자(스페이스, 탭, 개행문자)를 무시
ios::unitbuf	0x0002	출력 스트림에 들어오는 데이터를 버퍼링하지 않고 바로 출력
ios::uppercase	0x0004	16진수의 A~F, 지수 표현의 E를 대문자로 출력
ios::showbase	0x0008	16진수이면 0x를, 8진수이면 0을 숫자 앞에 붙여 출력
ios::showpoint	0x0010	실수 값에 대해, 정수 부분과 더불어 소수점 이하의 끝자리들을 0으로 출력
ios::showpos	0x0020	양수에 대해 + 기호 출력
ios::left	0x0040	필드를 왼쪽 맞춤(left-align) 형식으로 출력
ios::right	0x0080	필드를 오른쪽 맞춤(right-align) 형식으로 출력
ios::internal	0x0100	부호는 왼쪽 맞춤으로 숫자는 오른쪽 맞춤으로 출력
ios::dec	0x0200	10진수로 출력. 디폴트 설정
ios::oct	0x0400	8진수로 출력
ios::hex	0x0800	16진수로 출력
ios::scientific	0x1000	실수에 대해 과학 산술용 규칙에 따라 출력
ios::fixed	0x2000	실수에 대해 소수점 형태로 출력
ios::boolalpha	0x4000	설정되면, 논리값 true를 "true"로, false를 "false"로 출력하고, 설정되지 않으면, 정수 1과 0으로 출력

# 포맷 플래그를 세팅하는 멤버 함수

`Long setf(Long flags)`

`flags`를 스트림의 포맷 플래그로 설정하고 이전 플래그를 리턴한다.

`Long unsetf(Long flags)`

`flags`에 설정된 비트 값에 따라 스트림의 포맷 플래그를 해제하고 이전 플래그를 리턴한다.

```
cout.unsetf(ios::dec); // 10진수 해제  
cout.setf(ios::hex); // 16진수로 설정  
cout << 30 << endl; // 1e가 출력됨
```

```
cout.setf(ios::dec | ios::showpoint); // 10진수 표현과 동시에 실수에  
// 소숫점이하 나머지는 0으로 출력  
cout << 23.5 << endl; // 23.5000으로 출력
```

# 예제 – setf(), unsetf()

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

```
int main() {
```

30 출력

```
    cout << 30 << endl; // 10진수로 출력
```

1e 출력

```
    cout.unsetf(ios::dec); // 10진수 해제
    cout.setf(ios::hex); // 16진수로 설정
    cout << 30 << endl;
```

0x1e 출력

```
    cout.setf(ios::showbase); // 16진수로 설정
    cout << 30 << endl;
```

0X1E 출력

```
    cout.setf(ios::uppercase); // 16진수의 A~F는 대문자로 출력
    cout << 30 << endl;
```

23.5000 출력

```
    cout.setf(ios::dec | ios::showpoint); // 10진수 표현과 동시에
                                           // 소숫점 이하 나머지는 0으로 출력
    cout << 23.5 << endl;
```

2.350000E+001  
출력

```
    cout.setf(ios::scientific); // 실수를 과학산술용 표현으로 출력
    cout << 23.5 << endl;
```

+2.350000E+001  
출력

```
    cout.setf(ios::showpos); // 양수인 경우 + 부호도 함께 출력
    cout << 23.5;
```

```
}
```

```
30
1e
0x1e
0X1E
23.5000
2.350000E+001
+2.350000E+001
```

# 포맷 함수 활용

`int width(int minWidth)`

출력되는 필드의 최소 너비를 `minWidth`로 설정하고 이전에 설정된 너비 값 리턴

`char fill(char cFill)`

필드의 빈칸을 `cFill` 문자로 채우도록 지정하고 이전 문자 값 리턴

`int precision(int np)`

출력되는 수의 유효 숫자 자리수를 `np`개로 설정. 정수 부분과 소수점 이하의 수의 자리를 모두 포함하고 소수점(.)은 제외

너비설정

```
cout.width(10); // 다음에 출력되는 "hello"를 10 칸으로 지정
cout << "Hello" << endl;
cout.width(5); // 다음에 출력되는 정수 12를 5 칸으로 지정
cout << 12 << endl;
```

Hello  
12

```
cout << '%';
cout.width(10); // 다음에 출력되는 "Korea/"만 10 칸으로 지정
cout << "Korea/" << "Seoul/" << "City" << endl;
```

% Korea/Seoul/City

빈칸 채우기

```
cout.fill('^');
cout.width(10);
cout << "Hello" << endl;
```

^^^^Hello

유효숫자 자리수

```
cout.precision(5);
cout << 11./3.;
```

3.6667

# 예제 – width(), fill(), precision()

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

void showWidth() {
    cout.width(10); // 다음에 출력되는 "hello"를 10 칸으로 지정
    cout << "Hello" << endl;
    cout.width(5); // 다음에 출력되는 정수 12를 5 칸으로 지정
    cout << 12 << endl;

    cout << '%';
    cout.width(10); // 다음에 출력되는 "Korea/"만 10 칸으로 지정
    cout << "Korea/" << "Seoul/" << "City" << endl;
}

int main() {
    showWidth(); cout << endl;

    cout.fill('^');
    showWidth(); cout << endl;

    cout.precision(5);
    cout << 11./3. << endl;
}
```

```
    Hello
    12
%   Korea/Seoul/City

^^^^Hello
^^^12
%^^^^Korea/Seoul/City

3.6667
```

# 조작자

## ■ 조작자

- manipulator, 스트림 조작자(stream manipulator)
- 조작자는 함수
  - C++ 표준 라이브러리에 구현된 조작자 : 입출력 포맷 지정 목적
  - 개발자 만의 조작자 작성 가능 : 다양한 목적
  - 매개 변수 없는 조작자와 매개 변수를 가진 조작자로 구분
- 조작자는 항상 << 나 >> 연산자와 함께 사용됨

## ■ 매개 변수 없는 조작자

```
cout << hex << showbase << 30 << endl;  
cout << dec << showpos << 100 << endl;
```

```
0x1e  
+100
```

## ■ 매개 변수 있는 조작자

- #include <iomanip> 필요

```
cout << setw(10) << setfill('^') << "Hello" << endl;
```

```
^^^^Hello
```

# 매개 변수 없는 조작자

조작자	I/O	용도
endl	0	스트림 버퍼를 모두 출력하고 다음 줄로 넘어감
oct	0	정수 필드를 8진수 기반으로 출력
dec	0	정수 필드를 10진수 기반으로 출력
hex	0	정수 필드를 16진수 기반으로 출력
left	0	왼쪽 맞춤으로 출력
right	0	오른쪽 맞춤으로 출력
fixed	0	실수 필드를 고정 소수점 방식으로 출력
scientific	0	실수 필드를 과학 산술용 방식으로 출력
flush	0	스트림 버퍼 강제 출력
showbase	0	16진수의 경우 0x로, 8진수의 경우 0을 앞에 붙여서 출력
noshowbase	0	showbase 지정 취소
showpoint	0	실수 값에 대해, 정수 부분과 소수점 이하의 끝자리 이후 남은 공간을 0으로 출력
noshowpoint	0	showpoint 지정 취소
showpos	0	양수인 경우 + 부호를 붙여 출력
skipws	I	입력 스트림에서 공백 문자를 읽지 않고 건너뛴
noskipws	I	skipws 지정 취소
boolalpha	0	불린 값이 출력될 때, "true" 혹은 "false" 문자열로 출력

# 매개 변수를 가진 조작자

조작자	I/O	용도
<code>resetioflags(long flags)</code>	IO	flags에 지정된 플래그들 해제
<code>setbase(int base)</code>	0	base를 출력할 수의 진수로 지정
<code>setfill(char cFill)</code>	I	필드를 출력하고 남은 공간에 cFill 문자로 채움
<code>setioflags(long flags)</code>	IO	flags를 스트림 입출력 플래그로 설정
<code>setprecision(int np)</code>	0	출력되는 수의 유효 숫자 자리수를 np개로 설정. 소수점(.)은 별도로 카운트
<code>setw(int minWidth)</code>	0	필드의 최소 너비를 minWidth로 지정



# 예제 - 매개 변수 없는 조작자

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

int main() {
    cout << hex << showbase << 30 << endl;
    cout << dec << showpos << 100 << endl;
    cout << true << ' ' << false << endl;
    cout << boolalpha << true << ' ' << false << endl;
}
```

```
0x1e
+100
+1 +0
true false
```

boolalpha 조작자에 의해, "true", "false" 문자열로  
출력됨

# 예제 - 매개 변수를 가진 조작자

- 0에서 50까지 5의 배수를 10진수, 8진수, 16진수로 출력하라.

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

int main() {
    cout << showbase;

    // 타이틀을 출력한다.
    cout << setw(8) << "Number";
    cout << setw(10) << "Octal";
    cout << setw(10) << "Hexa" << endl;

    // 하나의 수를 십진수, 8진수, 16진수 형태로 한 줄에 출력한다.
    for(int i=0; i<50; i+=5) {
        cout << setw(8) << setfill('.') << dec << i; // 10진수
        cout << setw(10) << setfill(' ') << oct << i; // 8진수
        cout << setw(10) << setfill(' ') << hex << i << endl; // 16진수
    }
}
```

The diagram illustrates the output of the C++ program. It shows a table with three columns: Number, Octal, and Hexa. The rows represent the values 0, 5, 10, 15, 20, 25, 30, 35, 40, and 45. Annotations include 'setw(8)' pointing to the first column, 'setfill('.')' pointing to the first column, and 'showbase' pointing to the Hexa column. Brackets above the table indicate the width of each column: 8 for the first column, 10 for the second column, and 10 for the third column.

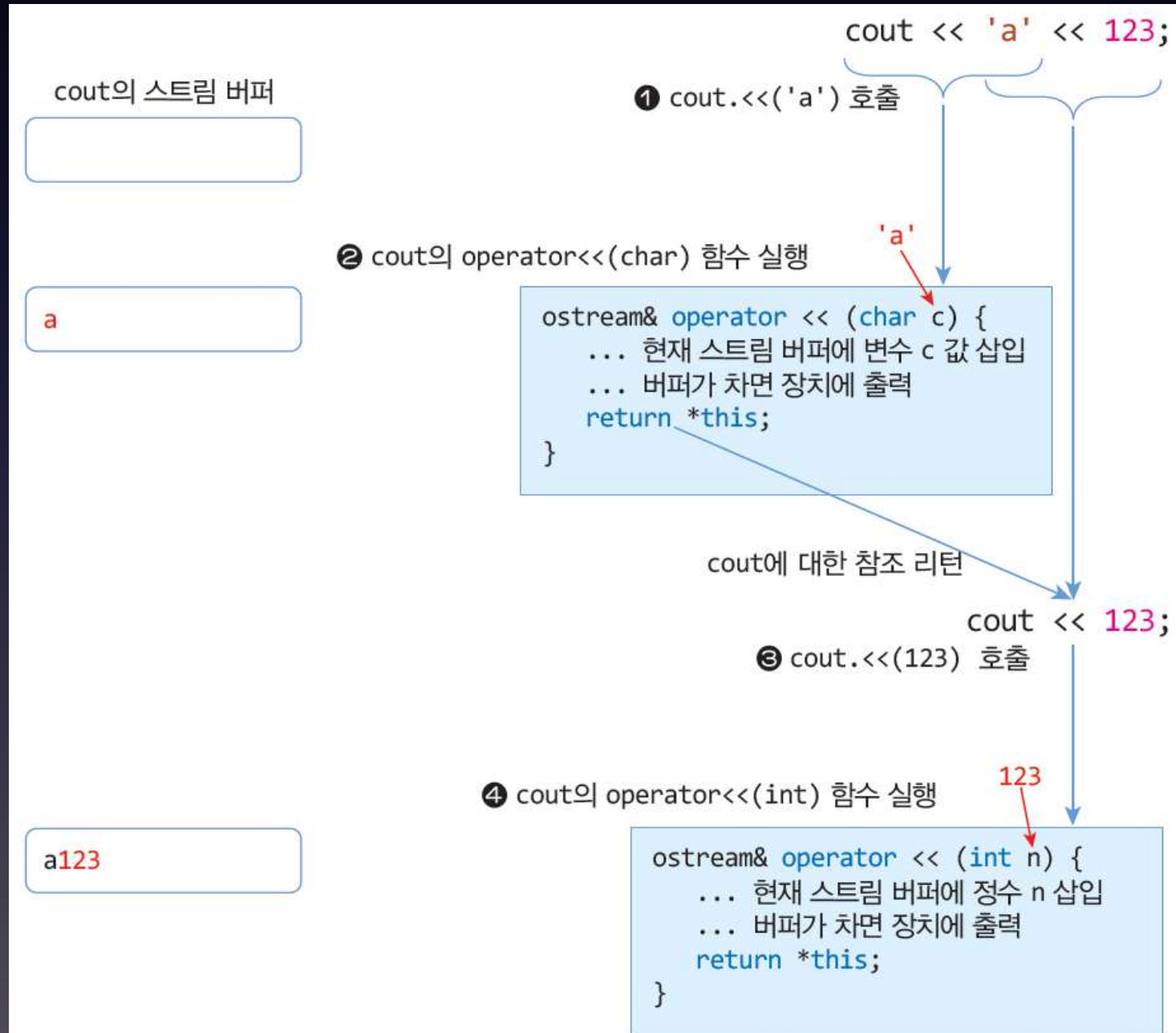
Number	Octal	Hexa
.....0	0	0
.....5	05	0x5
.....10	012	0xa
.....15	017	0xf
.....20	024	0x14
.....25	031	0x19
.....30	036	0x1e
.....35	043	0x23
.....40	050	0x28
.....45	055	0x2d

# 삽입 연산자(<<)

- 삽입 연산자(<<)
  - insertion operator, 삽입자라고도 부름
    - << 연산자는 C++의 기본 연산자 : 정수 시프트 연산자
  - ostream 클래스에 연산자 중복으로 작성되어 있음

```
class ostream : virtual public ios {  
    .....  
public :  
    ostream& operator<< (int n); // 정수를 출력하는 << 연산자  
    ostream& operator<< (char c); // 문자를 출력하는 << 연산자  
    ostream& operator<< (const char* s); // 문자열을 출력하는 << 연산자  
    .....  
};
```

# 삽입 연산자의 실행 과정



# 사용자 삽입 연산자 만들기

- 개발자가 작성한 클래스의 객체를 << 연산자로 출력

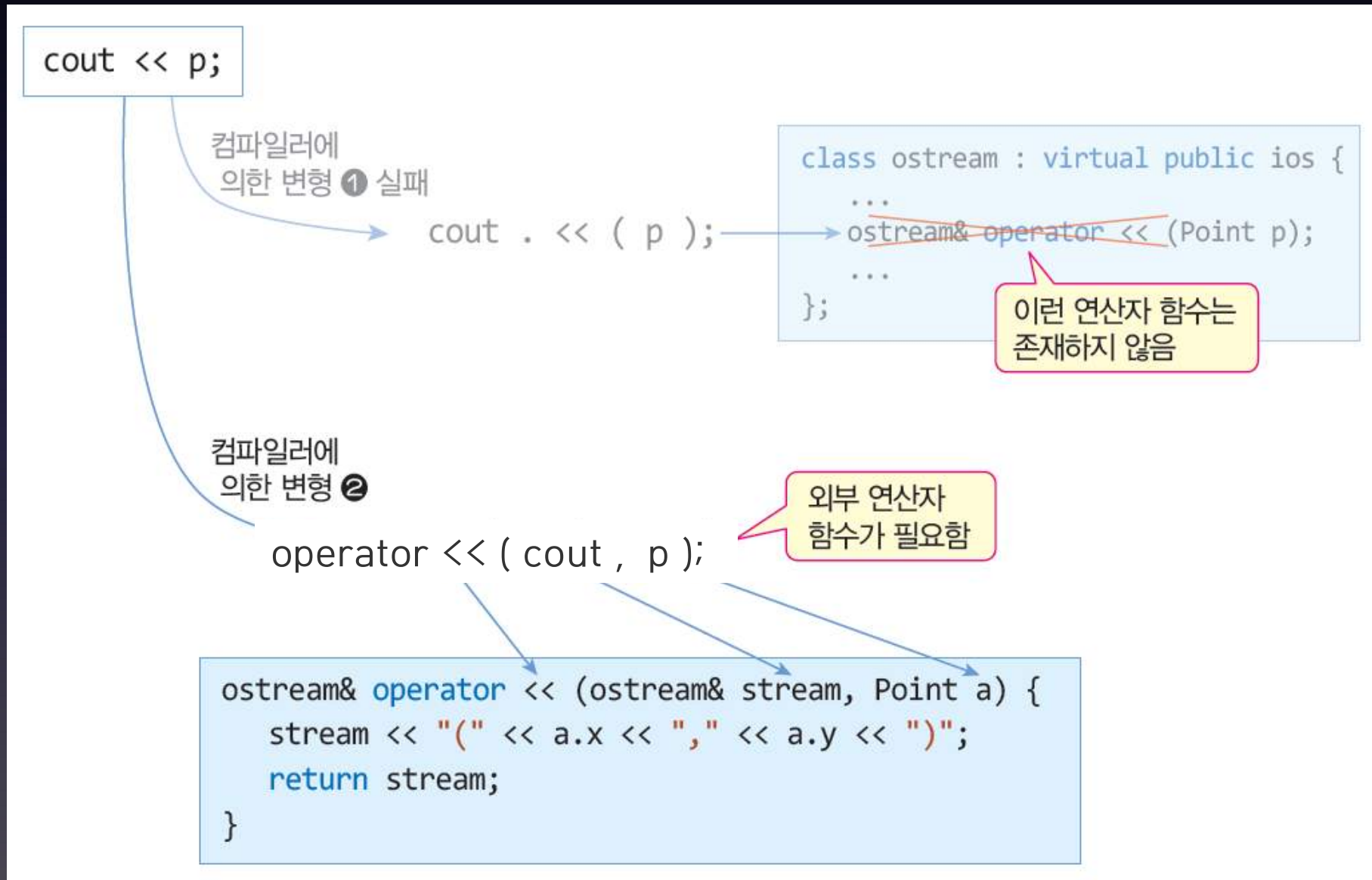
다음 Point 클래스에 대해 `cout << p;`가 가능하도록 << 연산자를 작성하라.

```
class Point {  
    int x, y;  
public:  
    Point(int x=0, int y=0) { this->x = x; this->y = y; }  
};
```

```
Point p(3,4);  
cout << p;
```

```
(3,4)
```

# cout << p;를 위한 << 연산자 만들기



# 예제 - Point 객체 출력

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

class Point { // 한 점을 표현하는 클래스
    int x, y; // private 멤버
public:
    Point(int x=0, int y=0) {
        this->x = x;
        this->y = y;
    }
    friend ostream& operator << (ostream& stream, Point a);
};

// << 연산자 함수
ostream& operator << (ostream& stream, Point a) {
    stream << "(" << a.x << "," << a.y << ")";
    return stream;
}

int main() {
    Point p(3,4); // Point 객체 생성
    cout << p << endl; // Point 객체 화면 출력

    Point q(1,100), r(2,200); // Point 객체 생성
    cout << q << r << endl; // Point 객체들 연속하여 화면 출력
}
```

private 필드 x, y를 접근하기 위해  
이 함수를 Point 클래스에  
friend로 선언함.

(3,4)  
(1,100)(2,200)

# 추출 연산자(>>)

- 추출 연산자(>>)
  - extraction operator
    - >> 연산자는 C++의 기본 연산자 : 정수 시프트 연산자
  - ostream 클래스에 연산자 중복으로 작성되어 있음

```
class istream : virtual public ios {  
    .....  
public :  
    istream& operator>> (int& n); // 정수를 입력하는 >> 연산자  
    istream& operator>> (char& c); // 문자를 입력하는 >> 연산자  
    istream& operator>> (const char* s); // 문자열을 입력하는 >> 연산자  
    .....  
};
```



# 사용자 추측 연산자 만들기

- 개발자가 작성한 클래스의 객체에 >> 연산자로 입력

다음 Point 클래스에 대해 cin >> p;가 가능하도록 >> 연산자를 작성하라.

```
class Point {  
    int x, y;  
public:  
    Point(int x=0, int y=0) { this->x = x; this->y = y; }  
};
```

```
Point p;  
cin >> p;  
cout << p;
```

```
x 좌표>>100  
y 좌표>>200  
(100,200)
```

# cin >> p;를 위한 >> 연산자 만들기

cin >> p;

컴파일러에  
의한 시도 ❶

cin.operator >> ( p );

```
class istream : virtual public ios {  
    ...  
    istream& operator >> (Point& p);  
    ...  
};
```

컴파일러에  
의한 시도 ❷

operator >> ( cin , p );

```
istream& operator >> (istream& stream, Point& a) {  
    ... // stream으로부터 입력 받는 코드  
    return stream;  
}
```

# 예제 - Point 객체 입출력

```
class Point { // 한 점을 표현하는 클래스
```

```
    int x, y; // private 멤버
```

```
public:
```

```
    Point(int x=0, int y=0) {
```

```
        this->x = x;
```

```
        this->y = y;
```

```
    }
```

```
    friend istream& operator >> (istream& ins, Point &a); // friend 선언
```

```
    friend ostream& operator << (ostream& stream, Point a); // friend 선언
```

```
};
```

```
istream& operator >> (istream& ins, Point &a) { // >> 연산자 함수
```

```
    cout << "x 좌표>>";
```

```
    ins >> a.x;
```

```
    cout << "y 좌표>>";
```

```
    ins >> a.y;
```

```
    return ins;
```

```
}
```

```
ostream& operator << (ostream& stream, Point a) { // << 연산자 함수
```

```
    stream << "(" << a.x << "," << a.y << ")";
```

```
    return stream;
```

```
}
```

```
int main() {
```

```
    Point p; // Point 객체 생성
```

```
    cin >> p; // >> 연산자 호출하여 x 좌표와 y 좌표를 키보드로 읽어 객체 p 완성
```

```
    cout << p; // << 연산자 호출하여 객체 p 출력
```

```
}
```

```
x 좌표>>100
```

```
y 좌표>>200
```

```
(100,200)
```

cin >> p 실행

cout << p 실행

# 사용자 정의 조작자 함수 원형

- 매개 변수 없는 조작자의 경우

```
istream& manipulatorFunction (istream& ins)
```

입력 스트림에 사용되는 조작자 원형

```
ostream& manipulatorFunction (ostream& outs)
```

출력 스트림에 사용되는 조작자 원형

# 예제 - 사용자 정의 조작자

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

ostream& fivestar(ostream& outs) {
    return outs << "*****";
}

ostream& rightarrow(ostream& outs) {
    return outs << "---->";
}

ostream& beep(ostream& outs) {
    return outs << '\a'; // 시스템 beep(뽕 소리) 발생
}

int main() {
    cout << "벨이 울립니다" << beep << endl;
    cout << "C" << rightarrow << "C++" << rightarrow << "Java" << endl;
    cout << "Visual" << fivestar << "C++" << endl;
}
```

```
벨이 울립니다
C---->C++---->Java
Visual*****C++
```

# 예제 - 사용자 정의 조작자

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

조작자 작성

```
istream& question(istream& ins) {
    cout << "거울아 거울아 누가 제일 예쁘니?";
    return ins;
}
```

```
int main() {
    string answer;
    cin >> question >> answer;
    cout << "세상에서 제일 예쁜 사람은 " << answer << "입니다." << endl;
}
```

조작자 사용

거울아 거울아 누가 제일 예쁘니?백설공주  
세상에서 제일 예쁜 사람은 백설공주입니다.

# 실습

- **Circle** 클래스의 객체를 입출력하는 다음 코드와 실행 결과를 참조하여 <<, >> 연산자를 작성하고 **Circle** 클래스를 완성하시오.

```
:  
class Circle {  
    string name;  
    int radius;  
public:  
    Circle(int r=1, string str=""){  
        radius = r; name = str;  
    }  
};  
  
int main() {  
    Circle c1, c2;  
    cin >> c1 >> c2;  
    cout << c1 << c2 << endl;  
}
```

```
Radius >> 5  
Name >> warfle  
Radius >> 20  
Name >> pizza  
(warfle of radius 5)(pizza of radius 20)
```

