

스트림

1

- 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

현재의 표준 C++ 입출력 라이브러리

2

- 다양한 크기의 다국어 문자를 수용하기 위해, 입출력 라이브러리가 템플릿으로 작성됨

스트림 입출력
기본 템플릿 클래스

basic_ios

입출력 스트림
템플릿 클래스

basic_istream

basic_ostream

basic_iostream

파일 입출력
템플릿 클래스

basic_ifstream

basic_ofstream

basic_fstream

새 표준 입출력 라이브러리 - 템플릿으로 작성

ios

istream

ostream

iostream

ifstream

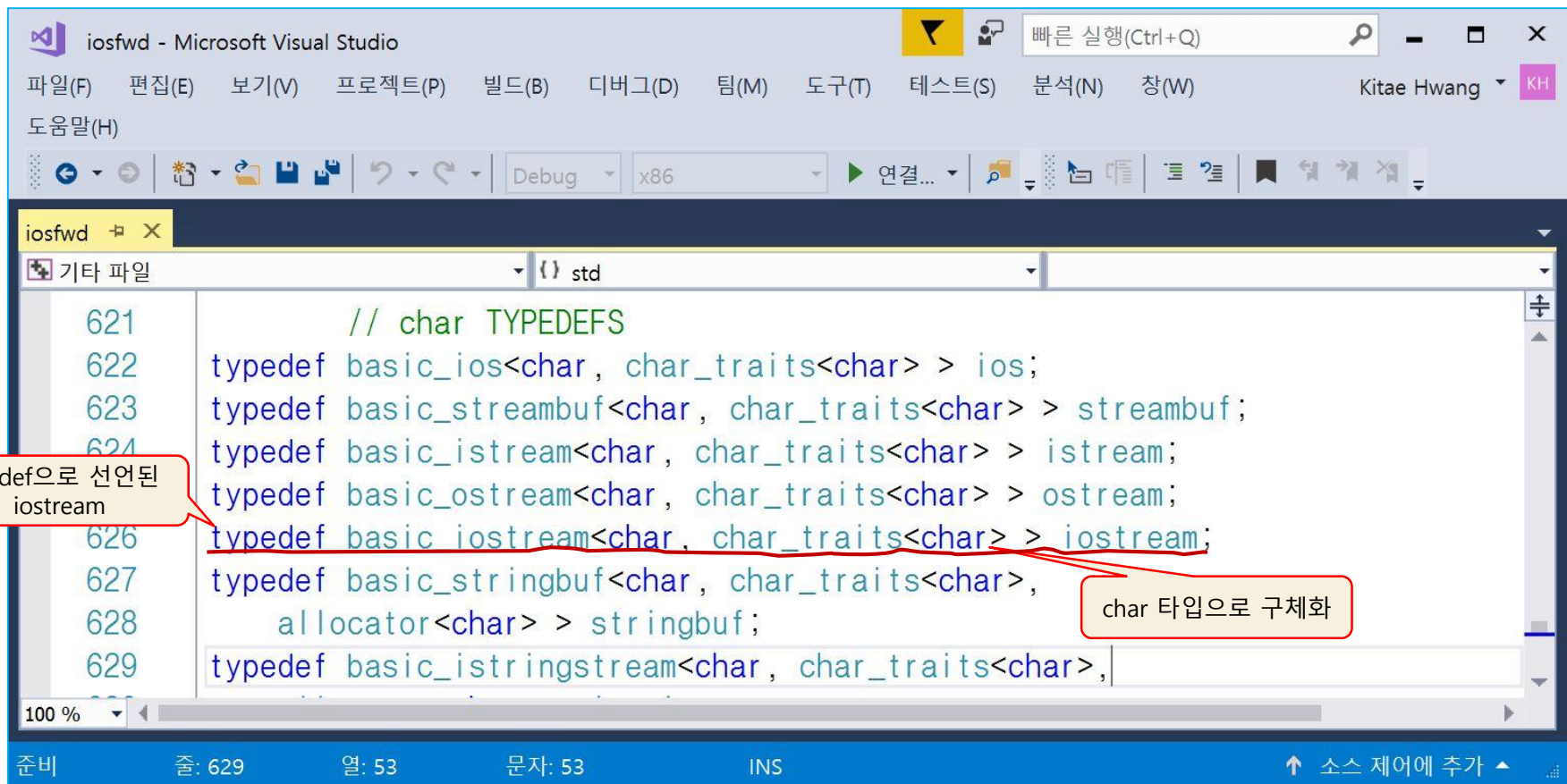
ofstream

fstream

템플릿에 char 타입으로 구체화한 클래스
- 구 표준의 이름 그대로 사용할 수 있음

typedef로 선언된 ios, istream, ostream, iostream 클래스

3



입출력 클래스 소개

4

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

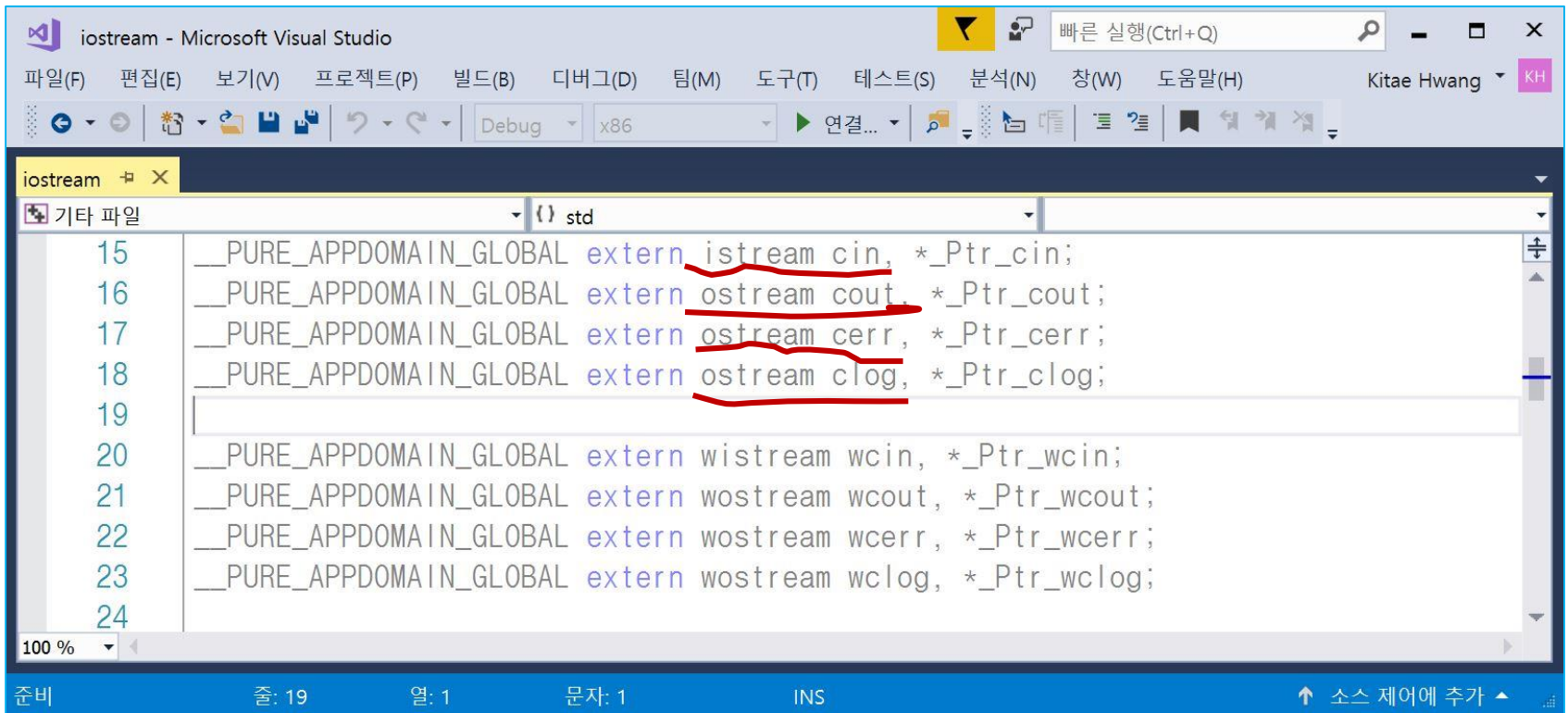
C++ 표준 입출력 스트림 객체

5

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

<iostream>에 선언된 스트림 객체들

6



```
iostream - Microsoft Visual Studio
빠른 실행(Ctrl+Q)

파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 도구(T) 테스트(S) 분석(N) 창(W) 도움말(H) Kitae Hwang KH

Debug x86 연결...

iostream
기타 파일 {} std
15 __PURE_APPDOMAIN_GLOBAL extern istream cin, *_Ptr_cin;
16 __PURE_APPDOMAIN_GLOBAL extern ostream cout, *_Ptr_cout;
17 __PURE_APPDOMAIN_GLOBAL extern ostream cerr, *_Ptr_cerr;
18 __PURE_APPDOMAIN_GLOBAL extern ostream clog, *_Ptr_clog;
19
20 __PURE_APPDOMAIN_GLOBAL extern wistream wcin, *_Ptr_wcin;
21 __PURE_APPDOMAIN_GLOBAL extern wostream wcout, *_Ptr_wcout;
22 __PURE_APPDOMAIN_GLOBAL extern wostream wcerr, *_Ptr_wcerr;
23 __PURE_APPDOMAIN_GLOBAL extern wostream wclog, *_Ptr_wclog;
24
100 %
준비 줄: 19 열: 1 문자: 1 INS ↑ 소스 제어에 추가 ▲
```

삽입 연산자(<<)

7

□ 삽입 연산자(<<)

- insertion operator, 삽입자라고도 부름

 - << 연산자는 C++의 기본 연산자 : 정수 시프트 연산자

- ostream 클래스에 중복 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

cout의 스트림 버퍼

a

a123

cout << 'a' << 123;

① cout.<<('a') 호출

② cout의 operator<<(char) 함수 실행

```
ostream& operator << (char c) {
    ... 현재 스트림 버퍼에 변수 c 값 삽입
    ... 버퍼가 차면 장치에 출력
    return *this;
}
```

cout에 대한 참조 리턴

cout << 123;

③ cout.<<(123) 호출

④ cout의 operator<<(int) 함수 실행

```
ostream& operator << (int n) {
    ... 현재 스트림 버퍼에 정수 n 삽입
    ... 버퍼가 차면 장치에 출력
    return *this;
}
```


사용자 삽입 연산자 만들기

9

- 개발자가 작성한 클래스의 객체를 << 연산자로 출력

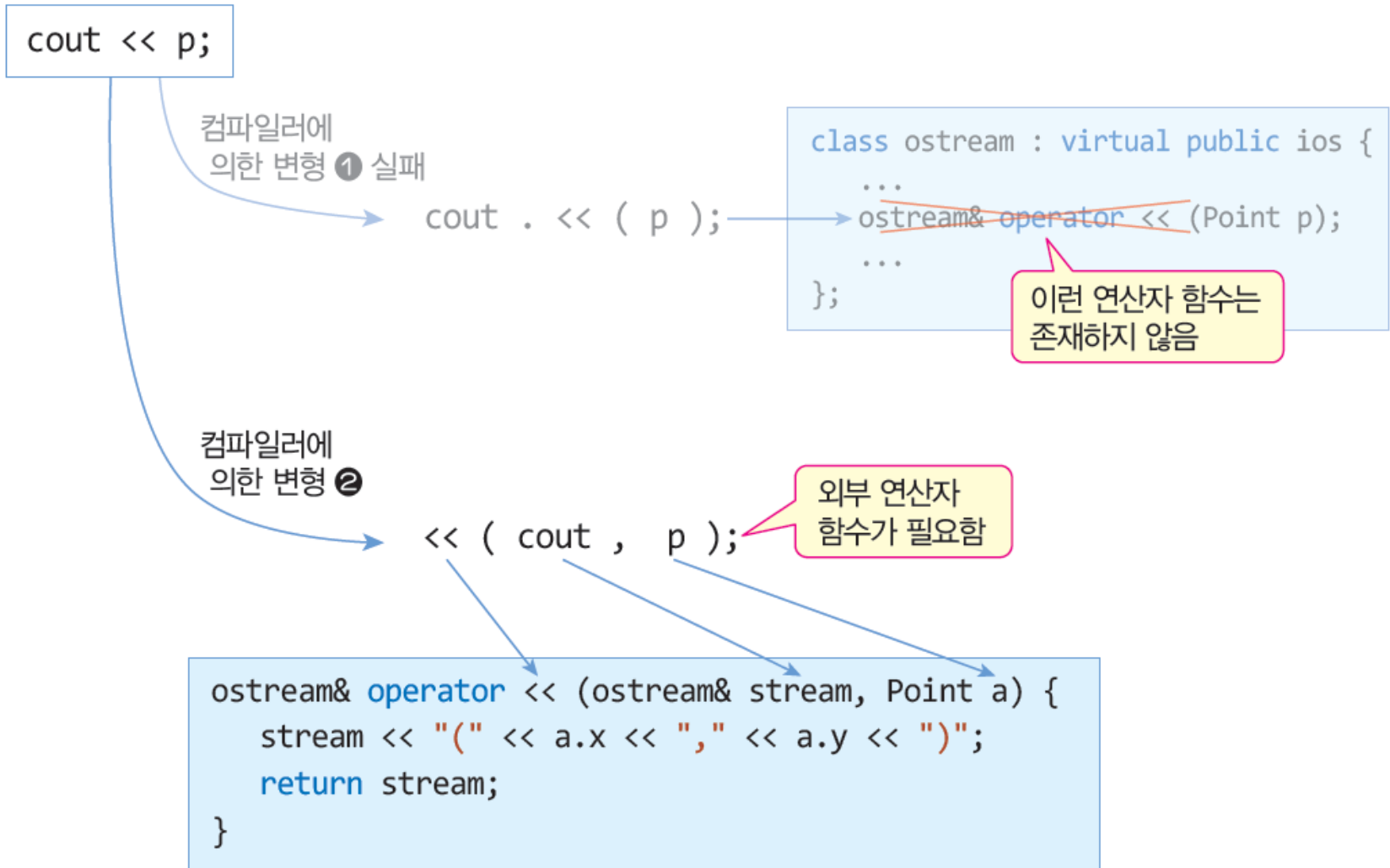
다음 Point 클래스에 대해 `cout << p;`가 가능하도록 << 연산자를 작성하라.

```
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



Point 객체를 스트림에 출력하는 << 연산자

11

```
#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(>>)

12

□ 추출 연산자(>>)

▣ extraction operator

- >> 연산자는 C++의 기본 연산자 : 정수 시프트 연산자

▣ ostream 클래스에 중복 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  
    .....  
};
```

▣ 추출 연산자의 실행 과정

- 삽입 연산자의 실행 과정과 유사하므로 생략

사용자 추출 연산자 만들기

13

- 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 실행

cout << p 실행

cin >> p;를 위한 >> 연산자 만들기

cin >> p;

컴파일러에
의한 시도 ❶

cin . >> (p);

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

이런 연산자 함수
는 istream에 존재
하지 않음

컴파일러에
의한 시도 ❷

>> (cin , p);

아래의 외부
연산자함수를
필요로 함

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

Point 객체를 입력 받는 >> 연산자 작성

```
#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); // friend 선언
```

```
    friend istream& operator >> (istream& ins, 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

조작자 실행 과정

```
cout << endl;
```

컴파일러에 의해 변형

```
cout . << ( endl );
```

호출

endl 함수 주소 전달

```
class ostream : virtual public ios {  
public:  
    ostream& operator << (ostream& (* _f)(ostream&));  
    ...  
};
```

```
ostream& ostream::operator << (ostream& (* _f)(ostream&)) {  
    (*_f)(*this); // *this는 cout  
    return *this; // cout의 참조 리턴  
}
```

endl(cout) 호출

cout 전달

```
ostream& endl(ostream& outs) {  
    outs.put('\n'); // 개행 문자 삽입  
    outs.flush(); // 버퍼 강제 출력  
    return outs; // 출력 스트림의 참조 리턴  
}
```