

객체지향언어



11장 입출력 처리

학습목표

- 콘솔 입출력을 설명하고, 입출력 연산자를 사용할 수 있다.
- 콘솔 입출력에서 사용하는 함수와 조작자를 사용할 수 있다.
- 파일 입출력을 할 수 있다

C++의 스트림 입출력

■ 표준 입출력 방법으로 **cin**과 **cout**을 사용

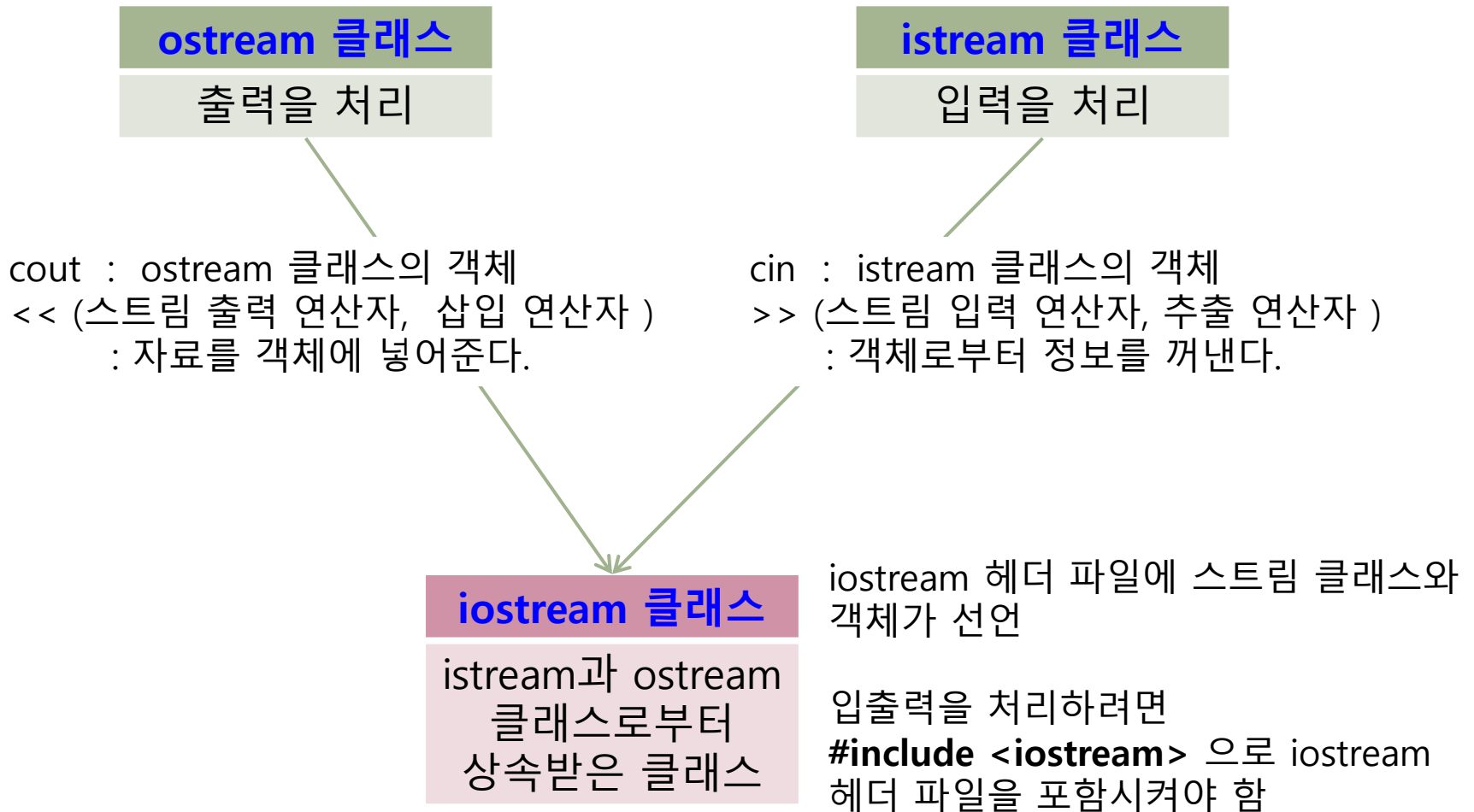
- cin과 cout은 형식을 한 번만 정해주면 계속 사용 가능
- 연산자 << 와 >>는 기본 자료형들을 인식하여 처리할 수 있도록 다중 정의되어 있어 편리함

■ 스트림(stream)

- cin이나 cout 객체를 통해서 입출력하는 자료의 흐름
- 출력 스트림(output stream)
 - 프로그램에서 화면, 프린터, 파일과 같은 프로그램 밖으로 흐르는 자료
- 입력 스트림(input stream)
 - 키보드, 파일, 외부장치와 같은 프로그램 밖에서 프로그램 안으로 흘러 들어오는 자료

C++의 스트림 입출력

■ iostream 헤더 파일



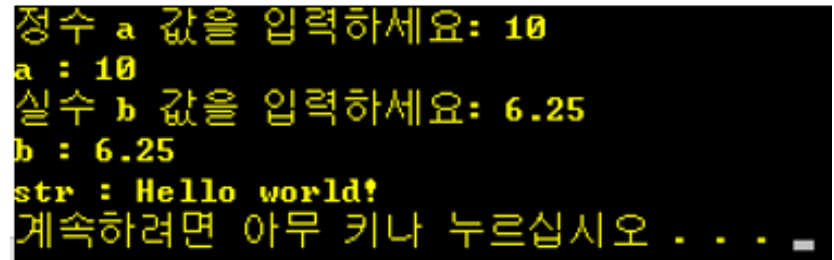
■ ex11_1.cpp (cin, cout을 이용한 입출력 예)

```
#include <iostream>
using namespace std;
int main()
{
    int a;
    double b;
    char *str = "Hello world!" ;

    cout << "정수 a 값을 입력하세요: " ;
    cin >> a;
    cout << "a : " << a << endl;

    cout << "실수 b 값을 입력하세요: " ;
    cin >> b;
    cout << "b : " << b << endl;
    cout << "str : " << str << endl;

    return 0;
}
```



A terminal window showing the execution of the C++ program. The output is in yellow text on a black background. It shows the program prompts for integer 'a' and double 'b', and then prints the values of 'a', 'b', and the string 'str'. The user has entered '10' for 'a' and '6.25' for 'b'. The string 'str' is printed as 'Hello world!'. The prompt '계속하려면 아무 키나 누르십시오 . . .' is visible at the bottom.

```
정수 a 값을 입력하세요: 10
a : 10
실수 b 값을 입력하세요: 6.25
b : 6.25
str : Hello world!
계속하려면 아무 키나 누르십시오 . . .
```

■ cout에서 사용하는 함수 (ostream 클래스의 함수)

- setf(fmtflags) 함수
 - 출력 형식을 지정할 수 있음
 - 원하는 설정들을 OR 연산자로 여러 개 지정 가능
- unsetf(fmtflags) 함수
 - 지정한 출력 형식 설정을 해제

[예시]

```
cout.setf(ios::hex);           // 16진수로 표시
cout.setf(ios::showbase);      // 16진수 출력에 0x 사용
cout.setf(ios::showpoint);     // 실수 출력에 소수점과 0을 표시

cout.unsetf(ios::hex);         // 16진수 표시를 해제
cout.setf(ios::dec);           // 10진수로 표시
```

■ cout에서 사용하는 함수

- setf() 함수에서 사용하는 출력 형식 지정 기호

출력 형식 지정 기호	기 능
ios::left	왼쪽으로 정렬한다.
ios::right	오른쪽으로 정렬한다.
ios::dec	10진수로 출력한다.
ios::hex	16진수로 출력한다.
ios::oct	8진수로 출력한다.
ios::uppercase	진수와 과학용 표기의 문자를 대문자로 한다.
ios::showbase	출력에 진법 표시 접두어(0, 0x)를 사용한다.
ios::showpoint	실수에서 뒤에 붙는 0과 소수점(.)을 출력한다.
ios::scientific	실수값을 과학용 표기(지수 형식)로 출력한다.
ios::fixed	실수값을 소수점 형식으로 출력한다.
ios::showpos	양수 앞에 + 부호를 출력한다.

■ ex11_2.cpp (필드의 출력 폭과 채울 문자 지정)

```
#include <iostream>
using namespace std;
int main()
{
    int a = 123;
    double b = 234.0;
    const double pi = 3.141592;
```

```
    cout << "a : " << a << endl;
```

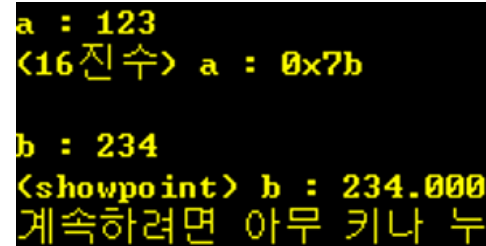
```
    cout.unsetf(ios::dec);
    cout.setf(ios::hex);
    cout.setf(ios::showbase);
```

```
    cout << "(16진수) a : " << a << endl << endl;
    cout << "b : " << b << endl;
```

```
    cout.setf(ios::showpoint);
    cout << "(showpoint) b : " << b << endl;
```

```
    return 0;
```

```
}
```



```
a : 123
<16진수> a : 0x7b
b : 234
<showpoint> b : 234.000
계속하려면 아무 키나 누르십시오 . . .
```


■ cout에서 사용하는 함수 (ostream 클래스의 함수)

■ **width(int)** 함수

- 필드의 폭을 지정한다.
- 바로 다음에 출력할 항목에만 적용되고, 그 이후에는 다시 디폴트값으로 되돌아간다.

■ **fill(char)** 함수

- 필드의 폭의 남은 부분은 기본적으로 빈 칸이다.
- fill() 함수를 사용하여 비워있는 부분을 지정한 문자로 채우도록 바꿀 수 있다. 다시 변경할 때까지 계속 유지된다.

[예시]

```
cout << '*';
```

```
cout.width(10);           // 출력할 필드의 폭을 10 자리로 지정한다.
```

```
cout << 123 << '*' << 123 << '*' << endl;
```

```
cout.fill('*');           // 글자 사이의 빈 부분을 '*'로 채운다.
```

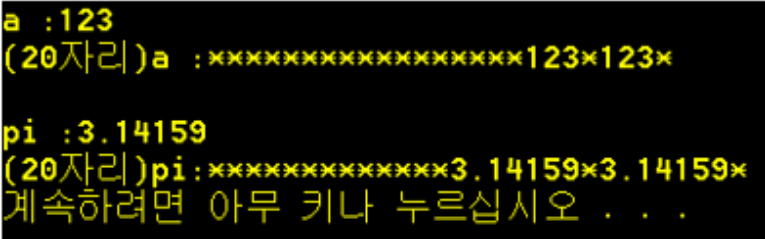
■ ex11_3.cpp (cout에서 사용하는 함수)

```
#include <iostream>
using namespace std;
int main()
{
    int a = 123;
    const double pi = 3.141592;

    cout << "a :" << a << endl ;
    cout << "(20자리)a :" ;
    cout.width(20);
    cout.fill('*');
    cout << a << '*' << a << '*' << endl << endl;

    cout << "pi :" << pi << endl ;
    cout << "(20자리)pi:" ;
    cout.width(20);
    cout << pi << '*' << pi << '*' << endl;

    return 0;
}
```



```
a :123
(20자리)a :xxxxxxxxxxxxxxxxxxxx123x123x

pi :3.14159
(20자리)pi:xxxxxxxxxxxxxxxx3.14159x3.14159x
계속하려면 아무 키나 누르십시오 . . .
```

■ cout에서 사용하는 함수(ostream 클래스의 함수)

■ precision(int) 함수

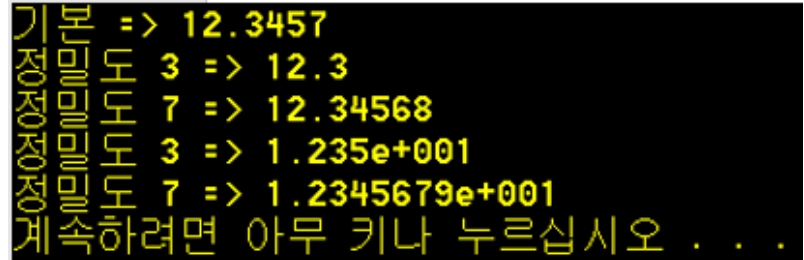
- 정밀도 (precision, 유효자리 숫자) 를 지정
- fixed나 scientific이 설정되어 있지 않은 기본 표기의 정밀도는 6이다. 정밀도보다 작아도 0을 붙여주지 않음
- 출력 형식을 고정소수점 표기 형식(fixed)으로 지정하면 정수 부분과 소수점 이하 부분의 자리수를 합하여 자리수를 지정
- 출력 형식을 지수 표기 형식(scientific)으로 지정하면 소수점 이하 부분의 자리수를 나타냄
- 정밀도로 지정된 값은 다시 설정할 때까지 계속 유지됨

■ ex11_4.cpp (정밀도 지정)

```
#include <iostream>
using namespace std;
int main()
{
    const double a = 12.3456789;

    cout << "기본 => " << a << endl;
    cout.precision(3);
    cout << "정밀도 3 => " << a << endl;
    cout.precision(7);
    cout << "정밀도 7 => " << a << endl;
    cout.setf(ios::scientific);
    cout.precision(3);
    cout << "정밀도 3 => " << a << endl;
    cout.precision(7);
    cout << "정밀도 7 => " << a << endl;

    return 0;
}
```



```
기본 => 12.3457
정밀도 3 => 12.3
정밀도 7 => 12.34568
정밀도 3 => 1.235e+001
정밀도 7 => 1.2345679e+001
계속하려면 아무 키나 누르십시오 . . .
```

■ cout에서 사용하는 함수 (ostream 클래스의 함수)

■ put(char) 함수

- 문자를 출력하고, 호출한 객체에 대한 참조를 반환

[예시]

```
int a = 'A';  
cout << a;      // 65를 출력한다.  
cout.put(a);   // 'A'를 출력한다.
```

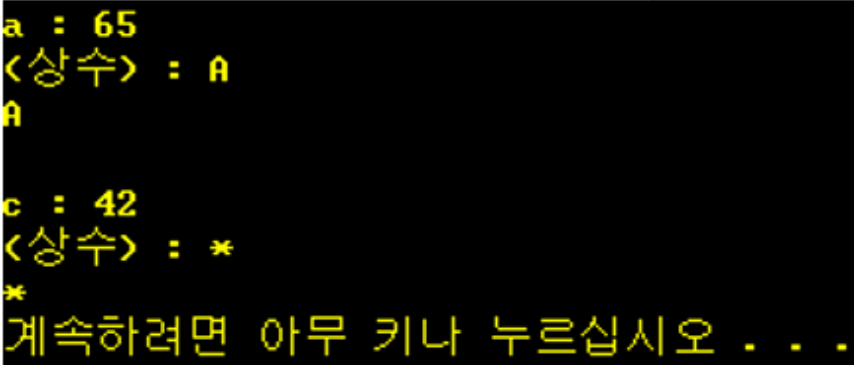
■ ex11_5.cpp (put() 함수 사용)

```
#include <iostream>
using namespace std;
int main()
{
    int a = 'A' ;
    int c = '*' ;

    cout << "a : " << a << endl;
    cout << "(상수) : " << 'A' << endl;
    cout.put(a).put('\n');
    cout << endl;

    cout << "c : " << c << endl;
    cout << "(상수) : " << '*' << endl;
    cout.put(c).put('\n');

    return 0;
}
```



```
a : 65
<상수> : A
A

c : 42
<상수> : *
*
계속하려면 아무 키나 누르십시오 . . .
```

■ 조작자(manipulator)

- cout은 입출력 형식을 편하게 제어하기 위한 조작자를 제공
- 조작자를 사용할 때는 iomanip 헤더 파일을 포함
- 조작자를 사용하면 출력 형태를 쉽게 바꿀 수 있음

[예시]

```
cout << showbase;
```

// cout.setf(**ios_base::showbase**)를 호출하여 출력에 진법 표시 접두어(0, 0x)를 사용하도록 설정한다.

```
cout << noshowbase;
```

// cout.unsetf(**ios_base::showbase**)를 호출하여 진법 표시 접두어(0, 0x) 사용을 해제한다.

■ C++에서 제공하는 조작자

조작자	기 능
endl	줄바꿈 문자를 출력한다.
ends	NULL을 출력한다.
oct	8진수 형식으로 출력한다.
dec	10진수 형식으로 출력한다.
hex	16진수 형식으로 출력한다.
uppercase	16진수의 문자와 과학적 표기법의 e를 대문자로 표시.
nouppercase	소문자로 돌아간다.
left	왼쪽으로 정렬한다. 기본 상태이다.
right	오른쪽으로 정렬한다.
fixed	고정소수점 표시 모드로, 소수 이하 6자리만 출력.
scientific	과학적 표기법모드로 설정한다.

■ C++에서 제공하는 조작자

조작자	기 능
showpoint	실수에 뒤에 붙는 0과 소수점(.)을 출력한다.
noshowpoint	소수점 표시를 해제한다.
showpos	수 앞에 + 부호를 출력한다.
noshowpos	양수 앞에 +기호 표시를 해제한다.
showbase	출력에 진법 표시 접두어(0, 0x)를 사용한다.
noshowbase	진법 표시 접두어(0, 0x)를 사용을 해제한다.
setw(int width)	너비를 width로 설정한다.
setprecision(int prec)	소수점 아래 prec 자리까지만 출력하도록 설정한다.
setposition(int pos)	pos 개의 자리수로 설정한다.
setfill(char c)	빈자리를 c 문자로 채우도록 설정한다.

cout 의 조작자

■ ex11_6.cpp (1) (조작자 사용)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int a = 123;
    double b = 234.0;
    const double pi = 3.141592;
```

```
(10진수)a : 123
(8진수)a : 173
(showbase)a : 0173

b : 234
(showpoint)b : 234.000

(setw(10),setfill(*)) a : *****0173, 0173
(setw(10),setfill(*))pi : ***3.14159
```

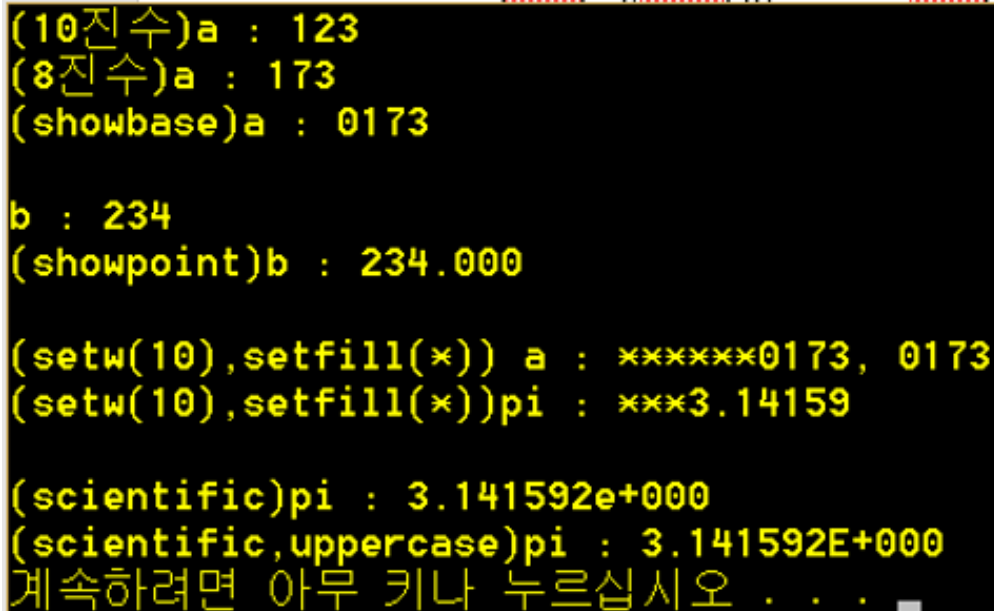
```
cout << "(10진수)a : " << dec << a << endl;
cout << "(8진수)a : " << oct << a << endl;
cout << "(showbase)a : " << showbase << oct << a << endl << endl;
```

```
cout << "b : " << b << endl;
cout << "(showpoint)b : " << showpoint << b << endl << endl;
```

```
cout << "(setw(10),setfill(*)) a : ";
cout << setw(10) << setfill('*') << a << ", " << a << endl;
cout << "(setw(10), setfill(*))pi : " << setw(10) << setfill('*') << pi << endl << endl;
```

■ ex11_6.cpp (2) (조작자 사용)

```
cout << "(scientific)pi : " << scientific << pi << endl;  
cout << "(scientific,uppercase)pi : " << scientific << uppercase << pi << endl;  
  
return 0;  
}
```



```
(10진수)a : 123  
(8진수)a : 173  
(showbase)a : 0173  
  
b : 234  
(showpoint)b : 234.000  
  
(setw(10),setfill(*)) a : *****0173, 0173  
(setw(10),setfill(*))pi : ***3.14159  
  
(scientific)pi : 3.141592e+000  
(scientific,uppercase)pi : 3.141592E+000  
계속하려면 아무 키나 누르십시오 . . .
```

cin에서 사용하는 함수

■ cin에서 사용하는 함수 (iostream 함수)

- >> 연산자를 사용하여 문자열을 입력하면 빈 칸과 그 이후의 문자들을 일어 들
일 수 없어 **get(), getline() 함수를 사용하면 빈 칸을 포함한 문자열을 읽을 수
있음**
- **get(char &) 함수**
 - **빈 칸을 포함하여 입력 문자를 읽고**, 전달인수에 대입
 - 자신을 호출하는데 사용한 **istream 객체에 대한 참조를 반환**
- **get(char *, int) 함수**
 - **한 행을 한 번에 읽을 수 있고, 줄바꿈 문자까지 읽음**
 - 첫 번째 인수는 입력 문자열을 저장할 메모리의 주소이고, 두 번째 인수는 읽어
들일 최대 문자수 보다 1만큼 큰 값
 - 자신을 호출하는데 사용한 **istream 객체에 대한 참조를 반환**
 - 줄바꿈 문자를 읽어서 버리지 않고 **입력 큐에 그대로 남겨둠**
 - 다음 번 호출에서 남아있는 줄바꿈 문자를 만나게 되고, **바로 행의 끝에 도달한 것으로
처리함**

cin에서 사용하는 함수

■ cin에서 사용하는 함수 (iostream 함수)

■ `getline(char *, int)` 함수

- 한 행을 한 번에 읽으며, 줄바꿈 문자까지 읽음
- 첫 번째 인수는 입력 문자열을 저장할 메모리의 주소이고, 두 번째 인수는 읽어 들일 최대 문자수 보다 1만큼 큰 값
- 입력 스트림으로부터 줄바꿈 문자를 읽어 제거

[예시]

<code>cin.get(ch);</code>	// 입력 문자를 전달인수 ch에 대입한다.
<code>cin.get(&ch[5], 6);</code>	// 최대 5개까지의 문자를 읽어 ch[5]에 대입한다.
<code>cin.getline(&ch[5], 6);</code>	// 최대 5개까지의 문자를 읽어 ch[5]에 대입한다.

■ ex11_7.cpp (1) (문자열 입력)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    char name[20];
    char a, b, c;

    cout << "3 개의 문자를 입력하세요: " ;
    cin.get(a).get(b).get(c);
    cout << "" << a << " " << b << " " << c << "" << endl;

    cout << "1.문자열을 입력하세요: " ;
    cin >> name;
    cout << name << endl;
    fflush(stdin);                // 입력 스트림을 비운다.

    cout << "2.문자열을 입력하세요: " ;
    cin.getline(name, 20);
    cout << name << endl;        // 입력 스트림을 비우지 않아도 된다.
```

cin에서 사용하는 함수

■ ex11_7.cpp (2) (문자열 입력)

```
cout << "3.문자열을 입력하세요: " ;
```

```
cin.get(name, 20);
```

```
cout << name << endl;    // 입력 스트림을 비워야 한다.
```

```
cout << "4.문자열을 입력하세요: " ;
```

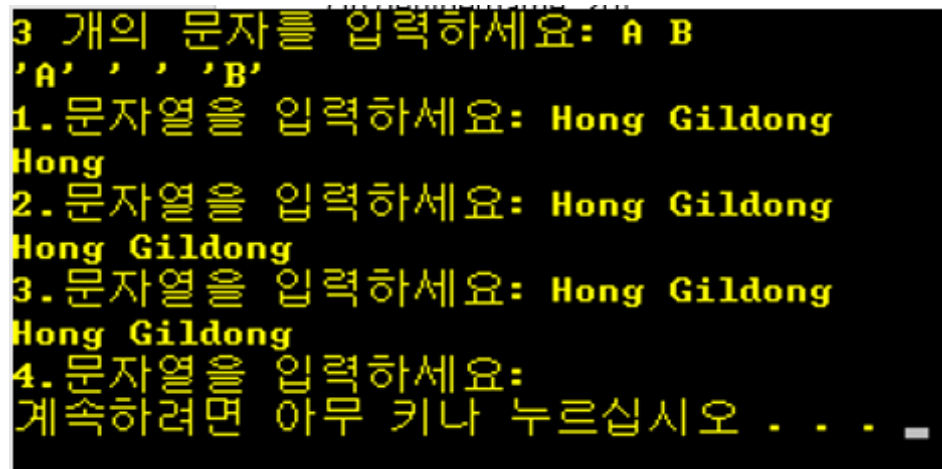
```
cin.getline(name, 20);
```

// 입력 스트림을 비우지 않아서 제대로 입력을 받을 수 없다.

```
cout << name << endl;
```

```
return 0;
```

```
}
```



```
3 개의 문자를 입력하세요: A B
'A' ' ' 'B'
1.문자열을 입력하세요: Hong Gildong
Hong
2.문자열을 입력하세요: Hong Gildong
Hong Gildong
3.문자열을 입력하세요: Hong Gildong
Hong Gildong
4.문자열을 입력하세요:
계속하려면 아무 키나 누르십시오 . . .
```

조작자 만들기

■ 입출력 조작자

- 조작자를 사용하여 **연속된 입출력**이 이루어지도록 하기 위하여 그것이 호출하는 스트림에 대한 참조를 받고 반환

■ 입력 조작자 함수

```
istream &입력_조작자_이름(istream &stream)
{
    stream 지정;
    ...
    return stream;
}
```


조작자 만들기

■ 출력 조작자 함수

```
ostream &출력_조작자_이름(ostream &stream)
{
    stream 지정;
    ...
    return stream;
}
```

[예시]

```
ostream &myform(ostream &stream)
{
    stream.width(6);
    stream.precision(2);
    stream.fill('*');
    ...
    return stream;
}
```

조작자 만들기

■ ex11_8.cpp (조작자 만들기)

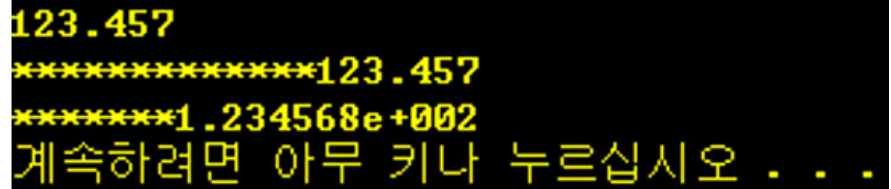
```
#include <iostream>
#include <iomanip>
using namespace std;
ostream &myform(ostream &stream)
{
    stream.width(20);
    stream.precision(6);
    stream.fill('*');

    return stream;
}

int main()
{
    double num = 123.456789;

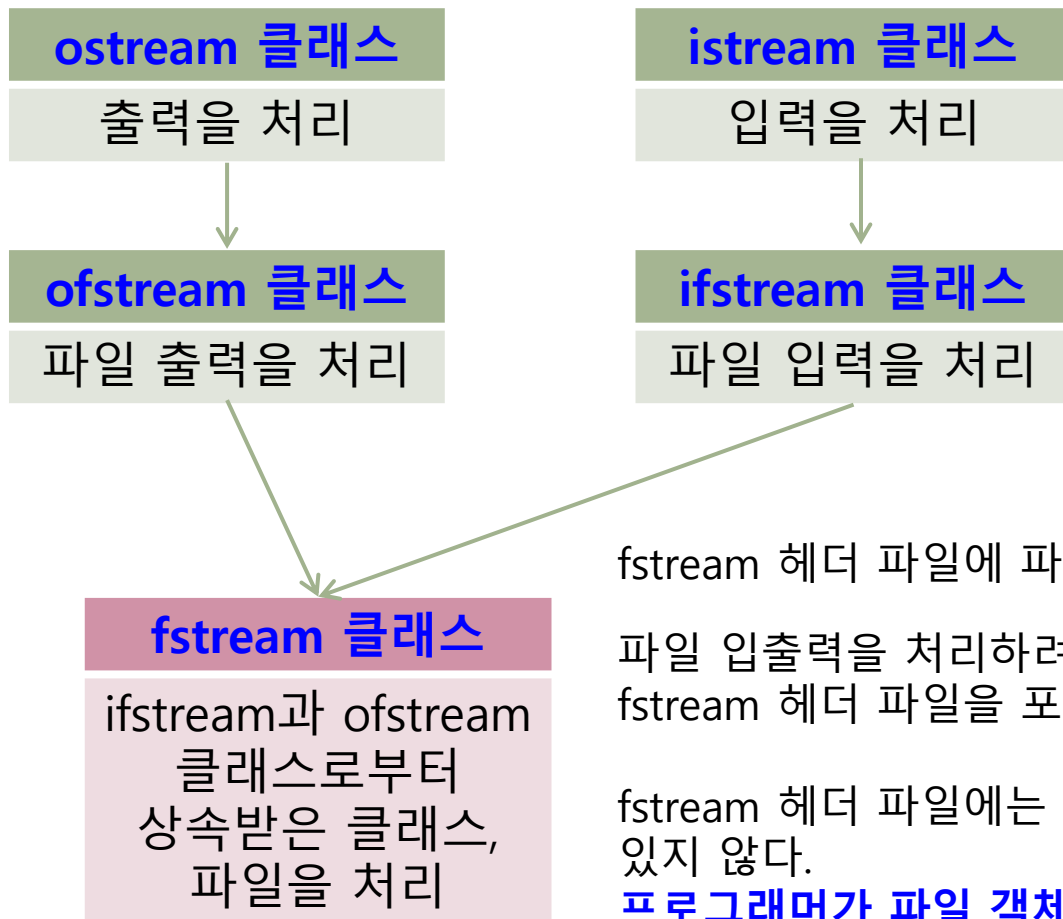
    cout << num << endl;
    cout << myform << num << endl;
    cout << scientific << myform << num << endl;

    return 0;
}
```



```
123.457
*****123.457
*****1.234568e+002
계속하려면 아무 키나 누르십시오 . . .
```

■ fstream 클래스



fstream 헤더 파일에 파일 처리 클래스가 선언되어 있다.

파일 입출력을 처리하려면 **#include <fstream>** 으로 fstream 헤더 파일을 포함시켜야 한다.

fstream 헤더 파일에는 클래스에 대한 객체가 선언되어 있지 않다.

프로그래머가 파일 객체를 선언하고 파일 사용 후 파일을 닫아주어야 한다.

■ C++ 파일 열기 모드

파일 열기 모드	기 능
<code>ios::in</code>	읽기 가능하게 파일을 연다.
<code>ios::out</code>	쓰기 가능하게 파일을 연다.
<code>ios::trunc</code>	기존 파일을 삭제하고 크기가 0인 파일을 새로 연다.
<code>ios::app</code>	파일을 쓸 때, 파일의 끝에 추가로 입력하게 한다.
<code>ios::binary</code>	이진 모드로 파일을 연다.
<code>ios::ate</code>	파일을 열 때, 파일 포인터를 파일의 끝부분으로 이동한다

■ C++와 C의 파일 열기 모드 비교

C++ 모드	C 모드	기 능
ios::in	"r"	읽기 위해 파일을 연다.
ios::out	"w"	쓰기 위해 열고, 파일이 존재하면 기존 파일을 삭제하고 크기가 0인 파일을 새로 연다.
ios::out ios::trunc	"w"	
ios::out ios::app	"a"	쓰기 위해 열고, 파일의 끝에 추가한다.
ios::in ios::out	"r+"	읽기/쓰기 겸용으로 연다. 파일이 없으면 새로 만든다.
ios::in ios::out ios::trunc	"w+"	읽기/쓰기 겸용으로 연다. 파일이 이미 존재하면 기존파일을 지운다.

■ C++ 파일 열기

```
(1) ofstream outputFile;           // 객체를 먼저 선언하고  
(2) outputFile.open("myfile.txt");  // 파일의 속성을 다음에 지정한다.
```

■ 혹은

```
(1) ifstream inputFile.open("myfile.txt", ios::in);  
(2) ofstream outputFile.open("myfile.txt", ios::out | ios::trunc);  
    // 파일을 열 때 객체와 속성을 한 번에 지정
```

■ C++ 파일 닫기

```
inputFile.close();  
outputFile.close();
```

■ ex11_9.cpp (1) (파일에 쓰기)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream myfile1("myfile1.txt");
    myfile1 << "This is a myfile1." << endl;
    myfile1 << "11, 12, 13, 14, 15, 16, 17, 18, 19." ;
    myfile1.close();
    cout << "myfile1.txt was created." << endl;

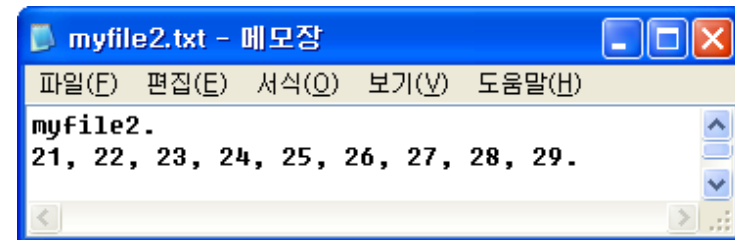
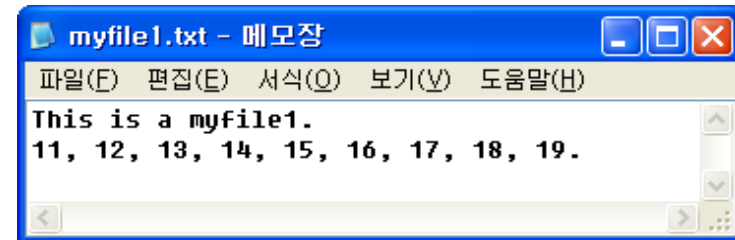
    ofstream myfile2;
    myfile2.open("myfile2.txt", ios::out | ios::trunc);
    myfile2 << "myfile2." << endl;
    myfile2 << "21, 22, 23, 24, 25, 26, 27, 28, 29." ;
    myfile2.close();
    cout << "myfile2.txt was created." << endl;

    return 0;
}
```

• 실행 결과 출력

```
myfile1.txt was created.
myfile2.txt was created.
계속하려면 아무 키나 누르십시오 . . .
```

• 실행한 후 생성된 파일



■ 파일 읽기

- myfile.txt이라는 파일로부터 입력하려면

(1) `ifstream inputFile;`

(2) `inputFile.open("myfile.txt");`

혹은

(1) `ifstream inputFile("myfile.txt");`

- cin 객체는 istream 클래스에서 파생된 ifstream 클래스의 객체이므로 inputFile 객체는 cin 객체의 모든 기능을 사용할 수 있음

[예시]

myfile.txt 파일로부터 읽어 `char *str;`에 저장하려면

`inputFile >> str;`

여기서 >> 연산자를 사용하면 공백을 만났을 때 읽기를 멈추므로 공백 이전까지만 읽어 반환한다.

■ ex11_10.cpp(myfile1.txt 파일을 읽어 myfile3.txt에 복사)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
```

```
    ifstream inputfile("..\\..\\..\\ex11_9\\..\\ex11_9\\..\\myfile1.txt");    //읽기 파일 포인터
    ofstream outputfile("myfile3.txt");                                // 쓰기 파일 포인터
    char buf[255];                                                    // 파일에서 읽어 저장할 공간 할당
```

```
    while( inputfile.getline(buf, 255) )    //inputfile에서 255Byte를 읽어 buf에 저장
    {
        outputfile << buf << endl;        // buf 데이터를 outputfile에 저장
    }
```

```
    inputfile.close();
    outputfile.close();
```

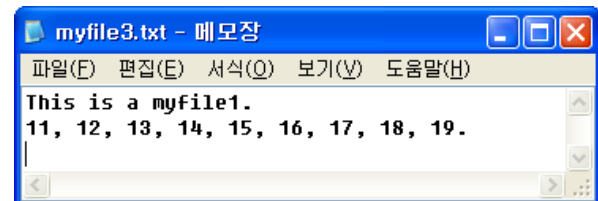
```
    cout << "파일 복사 끝(myfile1.txt ==> myfile3.txt)" << endl;
    return 0;
```

```
}
```

- 실행 결과 출력

```
파일 복사 끝<myfile1.txt ==> myfile3.txt>
계속하려면 아무 키나 누르십시오 . . .
```

- 실행한 후 생성된 파일



이진 파일 입출력

■ 이진 파일의 입력과 출력

- 이진 파일은 메모리의 값을 그대로 읽고 그대로 저장
- 이진 파일 입출력을 하려면 << 와 >> 연산자를 사용해서는 안 되고 **read()**와 **write()** 함수를 사용
- **read(buf, size)**
 - ifstream의 멤버함수로 이진파일을 입력할 때 사용
 - 현재 파일 포인터 위치에서 size 크기만큼 읽어 와서 buf 주소로 시작하는 메모리에 저장
- **write(buf, size);**
 - ofstream의 멤버함수로 이진파일을 출력할 때 사용
 - buf 주소로 시작하는 메모리에서 size 크기만큼의 데이터를 읽어 와서 파일에 기록

■ 이진 파일의 입력과 출력

[예시]

```
ifstream ifile;  
ifile.open("myfile_1", ios::binary | ios::in);  
ifile.read(char *buf, int count);  
  
ofstream ofile;  
ofile.open("myfile_2", ios::binary | ios::out);  
ofile.write(char *buf, int count);
```

이진 파일 입출력

■ 이진 파일의 입력과 출력

- 파일 스트림 클래스는 파일을 단순히 바이트들의 흐름 즉 바이트의 배열로 보고, 열려있는 파일은 파일로부터 읽어 들일 바이트와 출력하여 쓰고자 하는 바이트의 두 가지의 위치를 가지고 있음
- ifstream과 istream과 같은 입력 스트림에서는 읽을 요소를 가리키는 포인터로 **get 포인터를 사용**하고,
- ofstream과 ostream과 같은 출력 스트림에서는 쓰여질 위치를 가리키는 포인터로 **put 포인터를 사용**
- **tellg(), tellp()**
 - get과 put 포인터의 현재 위치를 나타낼 때 사용한다
- **seekg(offset, seekdir), seekp(offset, seekdir)**
 - get과 put 스트림 포인터의 위치를 변경할 수 있다

■ 이진 파일의 입력과 출력

- seekg(offset, seekdir), seekp(offset, seekdir) 에서 스트림 포인터의 위치를 나타내는 seekdir

seekdir	설 명
ios::beg	스트림의 시작
ios::cur	스트림 포인터의 현재 위치
ios::end	스트림의 끝

■ [예시]

```
ifstream ifile("ex1.txt");    // 파일을 연다.  
fbegin = ifile.tellg();       // 파일에서 현재(시작) 위치를 읽어온다.  
ifile.seekg(0, ios::end);     // 파일의 마지막 위치를 찾아간다.  
fend = ifile.tellg();         // 파일에서 현재(마지막) 위치를 읽어온다.  
ifile.close();                // 파일을 닫는다.
```

■ ex11_11.cpp(생성된 실행 파일(ex11_11.exe)의 크기 구하기)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    long fbegin, fend ;
    ifstream ifile("../debug\\ex11_11.exe", ios::binary | ios::in);
    if (!ifile.is_open())           // 파일 열기 성공여부 확인
    {
        cout << "파일을 열 수 없습니다." << endl ;
    }
    else
    {
        fbegin = ifile.tellg();      // 파일의 현재(시작) 위치 읽기
        ifile.seekg(0, ios::end);    // 파일 마지막 위치로 이동
        fend = ifile.tellg();        // 파일의 현재(마지막) 위치 읽기
        ifile.close();               // 파일 닫기
        cout << "파일의 크기는 " << fend - fbegin << " 바이트" << endl;
    }
    return 0;
}
```

파일의 크기는 93184 바이트
계속하려면 아무 키나 누르십시오 . . .

■ ex11_12.cpp (1) (파일을 읽어서 화면에 출력하는 예)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    int fsize;
    char *buf;
    ifstream ifile("WWEx11_12.cpp", ios::in|ios::binary|ios::ate);

    if (!ifile.is_open())                // 파일 열기 성공여부 확인
    {
        cout << "파일을 열 수 없습니다." << endl ;
    }
    else {
        ifile.seekg(0, ios::end);        // 파일 끝으로 파일 포인터 이동
        fsize = ifile.tellg();          // 파일의 현재(마지막) 위치 읽기
        buf = new char [fsize + 1];    // 동적 메모리 공간 할당
        ifile.seekg (0, ios::beg);      // 파일 시작위치로 파일 포인터 이동
        ifile.read (buf, fsize);        // 파일 데이터를 fsize만큼 읽어 buf에 저장
        buf[fsize] = '\0' ;            // 파일 끝에 '\0' 저장
        ifile.close();                 // 파일 닫기
    }
}
```

■ ex11_12.cpp(2)(파일을 읽어서 화면에 출력하는 예)

```
cout << "다음은 읽은 파일의 내용입니다." << endl ;  
cout << buf << endl;           // 파일에서 읽은 데이터 화면 출력  
delete[] buf;                   // 동적 메모리 공간 해제
```

```
}  
return 0;  
}
```

```
다음은 읽은 파일의 내용입니다.  
#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main()  
{  
    int fsize;  
    char *buf;  
    ifstream ifile("Ex11_12.cpp", ios::in|ios::binary|ios::ate);  
  
    if (!ifile.is_open()) {  
        cout << "파일을 열 수 없습니다." << endl ;  
    }  
    else {  
        ifile.seekg(0, ios::end);  
        fsize = ifile.tellg();  
        buf = new char [fsize + 1];  
  
        ifile.seekg (0, ios::beg);  
        ifile.read (buf, fsize);  
        buf[fsize] = '\0' ;  
        ifile.close();  
  
        cout << "다음은 읽은 파일의 내용입니다." << endl ;  
        cout << buf << endl;  
        delete[] buf;  
    }  
  
    return 0;  
}  
계속하려면 아무 키나 누르십시오 . . .
```




Thank You
