# 객체프로그래밍 13주차 정리노트

202204103 ict융합공학부 이승훈
202204023 ict융합공학부 김보민

## [스트림(stream)]

연속적인 데이터의 흐름이나 데이터를 전송하는 소프트웨어 모듈

**[입력 스트림]**
-키보드, 네트워크, 파일 등 입력 장치로부터 입력된 데이터를 순서대로 프로그램에 전달하는 객체이다.

**[출력 스트림]**
-프로그램에서 출력한 데이터를 프린터나 하드 디스크, 스크린, 네트워크, 파일 등과 같은 목적 장치로 순서대로 내보내는 객체이다.

## [C++ 입출력 스트림 버퍼]

**[키 입력 스트림의 버퍼]**
-<Backspace>키로 키 입력 도중 수정 가능, <Enter>키가 입력되면, 키 입력 버퍼에서 프로그램이 읽기 시작

**[스크린 출력 스트림 버퍼]**
-스크린에 연결된 cout 출력 스트림이 존재한다. 출력 스트림은 보통 '\n'이 도착하거나 버퍼가 꽉 찰 때 스크린에 출력시킴

## [입출력 방식]

**[스트림 입출력 방식(stream I/O)]**
-입력된 키는 버퍼에 저장 -> 엔터 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
-C++표준은 스트림 입출력 방식만 지원

**[저 수준 입출력 방식(raw level console I/O)]**
-키가 입력되는 즉시 프로그램에게 키 값 전달
-프로그램이 출력하는 즉시 출력 장치에 출력됨

# [입출력 클래스 소개]

**[ios 클래스]**
-모든 입출력 스트림 클래스의 기본 (base) 클래스이며, 스트림 입출력에 필요한 공통 함수와 상수,멤버 변수 선언.
-입출력 작업을 위해 바탕을 깔아주는 클래스

**[istream,ostream,iostream 클래스]**
-실제로 입력을 수행하는 클래스
-istream은 문자 단위 입력 스트림, ostream은 문자 단위 출력 스트림, iostream은 문자 단위 입출력을 동시에 할 수 있는 스트림 클래스

**[ifstream,ofstream,fstream 클래스]**
-파일에서 읽고 쓰는 기능을 가진 파일 입출력 스트림 클래스
-ifstream은 파일 읽기, ofstream은 파일에 쓰기, fstream은 읽고 쓰기를 동시에 할 수 있는 스트림 클래스

# [스트림 객체]

**[cin]**
-isream 타입의 스트림 객체로 키보드와 연결

**[cout]**
-ostream 타입의 스트림 객체로 스크린 장치와 연결

**[cerr]**
-ostream 타입의 스트림 객체로 스크린 장치와 연결
-오류 메시지 출력 목적
-스트림 내부 버퍼를 거치지 않고 출력

**[clog]**
-ostream 타입의 스트림 객체로 스크린 장치와 연결
-오류 메시지 출력 목적
-스트림 내부 버퍼를 거치고 출력

# [ostream 클래스의 멤버 함수]

**[ostream&put(char ch)]**
-ch문자를 스트림에 출력

**[ostream&write(char*str,int n)]**
-str 문자열에 있는 n개의 문자를 스트림에 출력

**[ostream&flush()]**
-현재 스트림 버퍼에 있는 내용 강제 출력

**[예제 11-1] ostream 멤버 함수를 이용한 문자 출력**

```
//보민 , ->승훈
#include <iostream>
using namespace std;

int main() {
	cout.put('H'); // cout 객체를 출력할 때  cout<< 대신 cout.을 사용하는 이유는
		c++에서 스트림 객체의 멤버 함수를 호출하고 값을 출력하기 위해서야.
		이러한 방식은 << 연산자를 사용하는 것보다 유연한 출력이 가능하며,
	여러개의 값을 연속적으로 출력할 때 더 간결한 코드를 작성할 수 있는 특징이 있어.
	cout.put('i');
	cout.put(33);
	cout.put('\n');

	cout.put('C').put('+').put('+').put(' ');

	char str[] = "I love programming";
	cout.write(str, 6);
}
// 그리고 이 코드를 잘 살펴보면 ostream 멤버 함수를 이용한 문자 출력의 예시라는 걸 알수 있어. put() 이라는 멤버 함수를 통해 Hi!라는 출력값을 만들어줬고, put() 멤버 함수를 연속해서 사용하여 c++이라는 값을 만든 뒤에 write() 멤버 함수를 사용해서 문자열의 일부를 가져올 수 있도록 하여 C++ I love 값을 만들어줬다고 볼 수 있지.
```

**[예제 11-2] get()과 get(char&)을 이용한 한 줄의 문자 읽기**

```
int main() {
        get1(); -> 메시지 출력, 변수
        get2();
}
```

**[함수 작성]**

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

void get1() {
        cout << "cin.get()로 <Enter> 키까지 입력 받고 출력합니다>>";
        int ch;
        while ((ch = cin.get()) != EOF) {
                cout.put(ch);
                if (ch == '\n')
                        break; -> 한 줄의 문장을 읽기 위하여 이렇게 코드를 작성해봤어.
                                  변수를 int로 선언해서 EOF와 비교할 수 있게 해줬고,
while                             루프문을 사용하여 문자를 읽고 출력하도록 했어.
        }
}

void get2() {
        cout << "cin.get(char&)로 <Enter> 키까지 입력 받고 출력합니다>>";
        char ch;
        while (true) {
                cin.get(ch);
                if (cin.eof()) break;
                cout.put(ch);
                if (ch == '\n')
                        break; -> 다음 get2를 구현할 때는 문자를 저장하고 출력할 수
                                  있도록 char형 변수를 만들어줬어.
        }
}
// 근데 이 코드에서 get1()의 변수를 char로 받으면 안 되는 거야 ?
-> 아니 가능해. 대신 줄바꿈 문자를 EOF와 비교하는 것이 아닌
while ((ch = cin.get()) != '\n') 이런식으로 문자로 비교해줘야 한다는 점만 알고 있으면 돼.
```

**[예제 11-4] getline()으로 한 줄 단위로 문장 읽기**

```
#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;
	while (true) {
		cout << "라인 " << no << " >> ";
		cin.getline(line, 80);
// 여기서 getline이 아닌 get을 쓰게 됐을 때 실행결과를 비교해보자. 어떻게 될 거 같아 ?
-> 음 get은 줄바꿈 문자 이전까지의 문자열을 읽어온다고 아는데 이때 줄바꿈 문자 또한 문자의 일부로 인식하기 때문에 무한 루프에 빠지게 될 거 같아.
// 맞아, cin.get() 함수는 줄바꿈 문자를 만나면 읽기를 중단하고 리턴하여서 줄바꿈 문자가 입력스트림 버퍼에 남아있게 돼. 만약 이 상태에서 다시 읽기를 하게 되면 입력 스트림에 남아있는 줄바꿈 문자부터 읽기 시작해서 아무것도 읽지 않고 바로 리턴하게 되지. 이 때문에 무한루프에 빠질 수 있어.
-> 그럼 get을 사용해서 코드의 오류가 안 나게 하려면 어떻게 해야 되지 ?
// cin.ignore()  함수를 통해서 문자열을 읽은 후 입력 스트림 버퍼에 남아있는 줄바꿈 문자를 제거해주면 돼. 그럼 무한루프에 빠지지 않게 될 거야.
		if (strcmp(line, "exit") == 0)
			break;
		cout << "echo --> ";;
		cout << line << endl;
		no++;
	}
}
```

# [포맷 입출력]

**[포맷 입출력 방법 3가지]**

ㄴ포맷 플래그

ㄴ포맷 함수

ㄴ조작자

**[포맷 플래그]**
-입출력 스트림에서 입출력 형식을 지정하기 위한 플래그

**[예제 11-5] setf(), unsetf()를 사용한 포맷 출력**

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

int main() {
	cout << 30 << endl;

	cout.unsetf(ios::dec);
	cout.setf(ios::hex);
	cout << 30 << endl;

	cout.setf(ios::showbase);
	cout << 30 << endl;

	cout.setf(ios::uppercase);
	cout << 30 << endl;

	cout.setf(ios::dec | ios::showpoint);
	cout << 23.5 << endl;

	cout.setf(ios::scientific);
	cout << 23.5 << endl;

	cout.setf(ios::showpos);
	cout << 23.5;
}
```

// 이 코드는 unsetf()와 setf() 함수를 사용하여 출력 형식을 변환시키는 예야. unsetf는 10진수 플래구룰 해제하고 setf는 16진수로 설정해줘. 따라서 이후에 출력되는 값은 16진수로 표현되게 돼. 따라서 두 번째 출력값은 30의 16진수 값인 1e로 출력되게 되는 걸 볼 수 있어. 다음은 어떻게 될지 예상해봐.

-> 세 번째 출력값은 16진수에 0x 접두어를 붙여 0x1e, 네 번째 출력값은 10진수 표현과 동시에 실수에 소숫점 이하 나머지는 0으로 출력되기 때문에 23.5000, 다섯 번째는 실수를 과학 산술용 표현으로 출력해야 하니까 인터넷을 참고해서 확인해보면 2.35999E+001, 마지막은 양수인 경우 +부호도 같이 출력되기 때문에 다섯 번째 출력 결과 앞에 플러스 부호를 붙여주면 될 거 같아.

**[포맷 함수 활용]**

ㄴint width(int minWidth)

-출력되는 필드의 최소 너비를 minWidth로 설정하고 이전에 설정된 너비 값 리턴

ㄴchar fill(char cFill)

-필드의 빈칸을 cFill 문자로 채우도록 지정하고 이전 문자 값 리턴

ㄴint percision(int np)

-출력되는 수의 유효 숫자 자리수를 np개로 설정, 정수 부분과 소수점 이하의 수의 자리를 모두 포함하고 소주점(.)은 제외

**[조작자]**

ㄴmanipulator, 스트림 조작자(stream manipulator)

ㄴ조작자는 함수

-C++ 표준 라이브러리의 구현된 조작자: 입출력 포맷 지정 목적

-개발자 만의 조작자 작성 가능: 다양한 목적

-매개 변수 없는 조작자와 가진 조작자로 구분

**[예제 11-7] 매개변수 없는 조작자 사용**

```
#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;
}
```

// 메인함수의 첫 줄에는 hex, showbase, endl 3개의 조력자가 쓰이고 있는 걸 볼 수 있어. hex는 16진수 형식으로 출력하도록 포맥하며, showbase는 숫자의 진수를 표기하도록 포맷하기 때문에 출력되는 정수가 16진수일 경우 0x르, 8진수인 경우는 0을 숫자 앞에 덧붙여. 마지막 조력자인 endl은 버퍼에 있는 데이터를 모두 출력하고 한 줄 띄우도록 하는 조작자이기 때문에 이 코드의 실행 결과는 0x1e가 되는 것을 볼 수 있지.

-> 그럼 다음줄은 내가 해석해보도록 할게. 이 코드도 dec, showpos, endl이라는 세 개의 조력자를 가지고 있어. dec는 조작자에 의해 10진수로 출력되게 되고 showpos는 조작자에 의해 + 부호를 붙여서 출력하게 돼 그럼 결과적으로 +100이 출력된 후 endl 조력자를 통해 다음줄로 넘어가게 되지.

// 조작자에 대해 잘 이해한 거 같아. 마저 코드를 보면 true는 +1로 false는 +0으로 출력되는 것을 볼 수 있고 마지막 줄에서는 boolalpha 조작자에 의해서 true와 false가 문자열로 출력되는 것을 볼 수 있어. 매개변수가 없는 조작자를 사용하기 위해선 <iostream> 헤더파일을 include 해야 한다는 것만 주의하면 될 거 같아

**[예제 11-9] Point 객체를 스트림에 출력하는 << 연산자**

**[원래 코드]**

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

class Point {
	int x, y;
public:
	Point(int x = 0, int y = 0) {
		this->x = x;
		this->y = y;
	}
	friend ostream& operator << (ostream& stream, Point a);
};

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

int main() {
	Point p(3, 4);
	cout << p << endl;

	Point q(1, 100), r(2, 200);
	cout << q << r < endl;
}
```

**[friend 함수 없이 작성]**

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

class Point {
	int x, y;
public:
```

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

        ostream& display(ostream& stream) const {
                stream << "(" << x << "," << y << ")";
                return stream;
        }
};

ostream& operator<<(ostream& stream, const Point& a) {
        return a.display(stream);
}

int main() {
        Point p(3, 4);
        cout << p << endl;

        Point q(1, 100), r(2, 200);
        cout << q << r << endl;
}
```

// 이 코드는 내가 설명해볼게. 이 코드는 friend 없이도 실행되는 이유를 알아보면 operator 함수가 전역함수로 정의되어있기 때문이야. 전역함수로 정의된 operator 함수는 원래 ostream 클래스랑 Point 클래스 사이에서 동작하는 연산자로 사용돼. 그래서 이 경우에는 friend를 사용해서 클래스 내부에 선언할 필요가 없게 되지. 따라서 ostream& operator<<(ostream& stream, const Point& a) dl 함수는 전역함수로 사용되므로 해당 클래스의 멤버에 직접 접근하지 않고도 동작할 수 있게 되는 거야.

**[예제 11-11] Point 객체를 입력 받는 >> 연산자 작성**

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

class Point {
	int x, y;
public:
	Point(int x = 0, int y = 0) {
		this->x = x;
		this->y = y;
	}
	friend istream& operator >> (istream& ins, Point &a);
	friend ostream& operator << (ostream& stream, Point a);
};

istream& operator >> (istream& ins, Point &a) {
	cout << "x 좌표>>";
	ins >> a.x;
	cout << "y 좌표>>";
	ins >> a.y;
	return ins;
}
ostream& operator << (ostream& stream, Point a) {
	stream << "(" << a.x << "," << a.y << ")";
	return stream;
}

int main() {
	Point p;
	cin >> p;
	cout << p;
}
```

-> operator>> 함수의 첫 번째 매개변수에는 입력 스트림을 나타내는 객체로 두고, 두 번째 매개변수는 Point 객체에 값을 저장하기 위한 참조를 준 뒤 어떻게 해야 할지 조금 막히는 것 같아.

// 매개변수 두 개를 잘 만들어줬어. 그 다음에는 우리가 사용자로부터 값을 입력 받아야 하니까 operator >> 함수 내부에서 ins 객체를 사용하여 사용자로부터 x,y 값을 받아주면 돼. 이 값을 a 객체의 멤버 변수에 저장해주고 이 함수를 반환하면 되니까 그렇게 해서 나온 코드는

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

```
        cout << "x 좌표>>";
        ins >> a.x;
        cout << "y 좌표>>";
        ins >> a.y;
        return ins;
}
ostream& operator << (ostream& stream, Point a) {
        stream << "(" << a.x << "," << a.y << ")";
        return stream;
}
```

이런식으로 작성할 수 있게 될 거야. 그리고 operator >> 함수의 반환 타입은 istream& 이라는 걸 알고 넘어가면 될 거 같아. 나도 작성해보니까 헷갈리는 부분이 많다. 과제를 하면서 더 연습해보자