13주차 입출력

스트림

- 데이터의 흐름, 데이터를 전송하는 소프트웨어 모듈
- 스트림의 양 끝에는 프로그램과 장치를 연결함

입력 스트림

- 입력장치로부터 입력된 데이터를 프로그램으로 전달하기 전에 일시 저장
- 키 입력 도중 수정 가능

출력 스트림

- 프로그램에서 출력된 데이터를 출력 장치로 보내기 전에 일시 저장
- 출력 장치를 반복적으로 사용하는 비효율성 개선

스트림 입출력 방식(stream I/O) (C++에서의 방식)

- 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
- Backspace키 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

문자를 한 바이트의 char로 처리

- cin >>로 문자를 읽을 때, 한글 문자 읽을 수 없음
- 영어나 기호 : 1 바이트의 문자 코드
- 한글 문자 : 2 바이트의 문자 코드
  -> 다양한 크기의 다국어 문자를 수용하기 위해, 입출력 라이브러리가 템플릿으로 작성됨

ios = input output stream

cin

- istream 타입의 스트림 객체로서 키보드 장치와 연결

  cout
- ostream 타입의 스트림 객체로서 스크린 장치와 연결

ostream 멤버함수 put함수
ex)11-1

```
#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
}
```

istream 멤버 함수 get함수
ex)11-2

```
#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. EOF 는 -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
}
```

get(char*, int)을 이용한 문자열 입력

ex)11-3

```
#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. //38
한글
                if(strcmp(cmd, "exit") == 0) {
                        cout << "프로그램을 종료합니다....";
                        return 0;
                }
                else
                        cin.ignore(1); // 버퍼에 남아 있는 <Enter> 키
('\n') 제거
        }
}
```

포맷 입출력 방법 3 가지

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

uppercase, skipws 등등
셋업과 언셋으로 사용할 포멧 플래그를 정의해주어야함

ex)11-5 setf(), unsetf()를 사용한 포맷 출력

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

int main() {
        cout << 30 << endl; // 10진수로 출력

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

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

```
        cout << 30 << endl; // 0x1e

        cout.setf(ios::uppercase); // 16진수의 A~F는 대문자로 출력
        cout << 30 << endl; // 0X1E

        cout.setf(ios::dec | ios::showpoint); // 10진수 표현과 동시에

// 소숫점 이하 나머지는 0으로 출력
        cout << 23.5 << endl;

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

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

- 포맷 함수

```
#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;
}
```

- 조작자(stream manipulator)
  개발자 만의 조작자 작성 가능 : 다양한 목적
  매개 변수 없는 조작자와 매개 변수를 가진 조작자로 구분
  `<<와 >>` 연산자와 같이 사용
  ex)11-7 매개변수 없는 조작자

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

int main() {
        cout << hex << showbase << 30 << endl; // 0x1e
        cout << dec << showpos << 100 << endl; // +100
        cout << true << ' ' << false << endl; // +1 +0
        cout << boolalpha << true << ' ' << false << endl;
//true false
}
```

ex)11-8 매개변수를 가진 조작자

```
#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
        }
}
// 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
```

사용자정의 삽입 연산자 함수

ex)11-10 Book class << 사용자 정의 연산자

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

class Book { // 책을 표현하는 클래스
        string title;
        string press;
        string author;
public:
        Book(string title="", string press="", string author="")
{
                this->title = title;
                this->press = press;
                this->author = author;
```

```
        }
        friend ostream& operator << (ostream& stream, Book b);
        // 연산자함수를 friend로 선언
};

&ostream operator << (ostream& stream, Book b){
        stream << b.title << "," << b.press << "," << b.author
<< endl;
        return stream;
}

int main() {
        Book book("소유냐 존재냐", "한국출판사", "에리히프롬"); // Book 객
체 생성
        cout << book; // Book 객체 book 화면 출력
}
```

추출연산자 사용자정의 연산자

ex)11-11

```
#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 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
}
```

사용자 정의 조작자

ex)11-12

```
#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(삑 소리) 발생
}

int main() {
```

```
        cout << "벨이 울립니다" << beep << endl;
        cout << "C" << rightarrow << "C++" << rightarrow <<
"Java" << endl;
        cout << "Visual" << fivestar << "C++" << endl;
}
```

ex)11-13

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

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;
}
```

**팀원과 논의한 점

사용자 정의 삽입 및 추출 연산자 함수의 장단점과 실사용

-> 사용자 정의 연산자 함수는 코드의 가독성과 유지보수성을 높일 수 있다고 생각함

-> 사용자가 정의한 클래스를 다룰 때, 기본 입출력 연산자는 해당 클래스에 적합한 형태로 출력하거나 입력받기 어려울 수 있음, 이런 경우 사용자 정의 연산자 함수를 통해 적절한 입출력 형태를 정의할 수 있습니다.

-> 사용자 정의 연산자 함수를 사용하면 어떤 이점이 있는가?

-> 가독성이 향상됩니다. 해당 클래스에 대한 지식이 없어도 명확한 출력이 가능하죠.
-> 또, 유지보수성이 증가합니다. 클래스의 내부 구조가 변경되더라도 해당 연산자 함수만 수

정하면 됨

조작자를 사용하면 가독성이나 유지보수성에 어떤 영향이 있는가?

-> 조작자를 사용하면 코드가 간결해지고 일관성을 유지할 수 있지만, 가독성이 감소함.
코드가 길어질수록 어떤 조작자가 어떤 역할을 하는지 파악하기 어려워지는 현상이 나타남.

그렇다면 둘 중 어느것이 더 효율적인 것인가?

-> 상황에 따라 다를것같다, 하지만 내 생각에는 일반적으로 사용자 정의 연산자 함수를 사용하면 클래스와 관련된 출력이나 입력이 필요한 경우에 더 유용할 것 같음.

-> 클래스가 복잡하고 그 구조가 변경될 가능성이 있을 때는 사용자 정의 연산자 함수를 고려,
-> 간단한 클래스이거나 구조가 거의 변하지 않는 코드라면 조작자를 사용하는 것도 좋음.