7.1 C++ 프렌드 개념

프렌드 클래스 외부에 작성된 함수를 friend키워드로 선언

클래스 멤버 함수와 동일한 접근 자격을 부여

멤버함수가 아니므로 상속되지는 않음

클래스의 모든 변수, 함수에 접근 가능

friend 선언은 함수만 가능

프렌드 함수가 필요한 이유 클래스의 멤버함수로는 적합하지 않지만, private, protected 멤버를 접근해야 하는 경우

ex) 연산자 함수

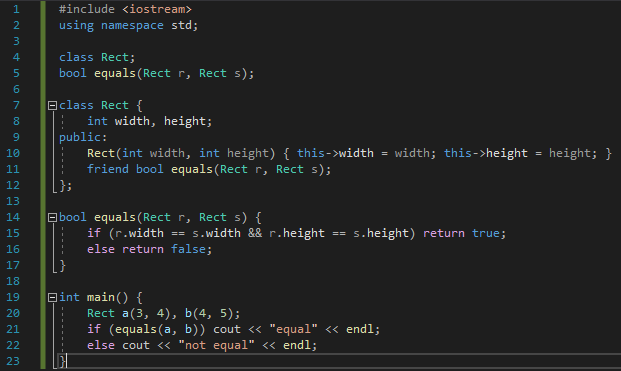
프렌드 함수 선언 클래스 외부에 구현된 함수를 friend 키워드로 클래스 내에 선언

ex) class Rect {

...

friend bool equals(Rect r, Rect s);

};

예제7-1 

전방선언 변수, 함수, 클래스의 이름을 선언한 뒤 그 이름을 참조

프렌드 멤버 선언 다른 클래스의 멤버 함수를 클래스의 프렌드 함수

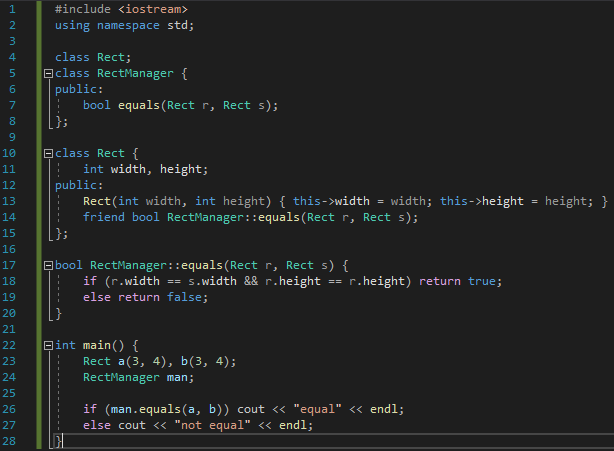
ex) RectManager 클래스의 멤버함수 equal(Rect r, Rect s)를 Rect클래스에 프렌드에 선언함

class Rect {

.....

friend bool RectManager::equals(Rect r, Rect s) //RectManager의 equals()멤버 함수를 프렌드로 초대

};

예제 7-2 

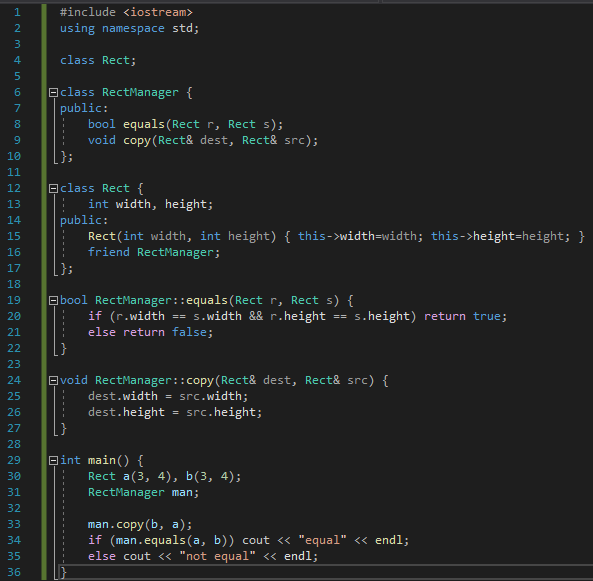
프렌드 클래스 선언 다른 클래스의 모든 멤버 함수를 클래스의 프렌드 함수로 한번에 초대 가능

ex) class Rect {

.....

friend RecManager; //RectManager 클래스의 모든 함수를 프렌드로 초대

};

예제 7-3 

check time 1. 2

2. friend f(Sample)

friend Test::f(Sample)

friend Test

7.2 연산자 중복

연산자 중복의 개념 다형성 연산자 중복 동일한 연산∙기호를 대상에 따라 서로 다른 의미로 해석

적합한 연산자를 새로 작성하는 것

함수중복

ex) 두 개의 정수 더하기, 두 개의 문자열 합치기, 클래스끼리 더하기

연산자 중복의 특징 C++언어에 본래 있는 연산자만 중복 가능 +, -, \*, /, == … 등 원래있는 C++연산자에 새로운 의미를 부여

%%, ##등 새로운 연산자를 만들어낼 순 없음

연산자 중복은 피연산자의 타입이 다른 연산을 새로 정의 피연산자를 새로 정의함

연산자 중복은 함수를 통해 이루어진다 새로운 연산 처리를 수행하는 함수(연산자 함수)를 구현함

연산자 중복은 반드시 클래스와 관계를 가진다 중복된 연산자는 반드시 피연산자에 객체를 동반함

연산자 함수 클래스의 멤버 함수로 구현

전역함수로 구현하고 클래스에 프렌트 함수로 선언

연산자 중복으로 피연산자의 개수를 바꿀 수 없다 이항연산자를 단항연산자, 삼항연산자으로 중복할 수 없음

연산자 중복으로 연산의 우선순위를 바꿀 수 없다 연산자의 중복을 통해 연산의 순위나 방향을 바꿀 수 없음

모든 연산자가 중복가능한 것은 아니다 . ∙ .\* ∙ :: ∙ ?: 을 제외한 연산자

연산자 함수 선언과 연산자 함수 개요 연산자 함수를 통해 연산자 중복 구현 클래스의 멤버함수로 구현

외부함수로 구현하고 클래스의 프렌드 함수로 구현

형식 리턴타입 operater 연산자(매개변수리스트)

이름이 operator키워드와 연산자로 구성되는 점 빼곤 보통 함수 선언과 같음

외부함수로 구현하고 클래스에 프렌드 함수로 선언하는 경우 ex) bool operator+ (Color op1, Color op2);

Color operator+ (Color op1, Color op2);

class Color {

...

friend Color operator+ (Color op1, Color op2);

friend bool operator+ (Color op1, Color op2);

};

클래스의 멤버함수로 선언되는 경우 ex) class Color {

Color operator +(Color op2);

bool operator ==(Color op2);

};

연산자의 오른쪽 피연산자만 매개변수에 전달되면, 왼쪽 피연산자는 객체 자신

check time 1. 3

2. Money operator+(Money op1, Money op2);

3. 4

7.3 이항 연산자 중복

+연산자 중복 class Power {

int kick;

int punch;

public:

Power(int kick = 0, int punch = 0) { this->kick = kick; this->punch = punch; }

};

연산자 착안 +의 의미를 정해야함

개발자의 몫

ex) 두 객체의 kick, punch를 각각 더하기

연산자 함수 선언 컴파일러가 기존의 +연산으로 할 수 없음을 판단 🡪 식 변환 ex) a+b 🡪 a.+(b) //a객체의 +멤버함수.;

operator+() 함수 호출

연산자 함수 구현 ex) Power Power::operator+(Power op2) {

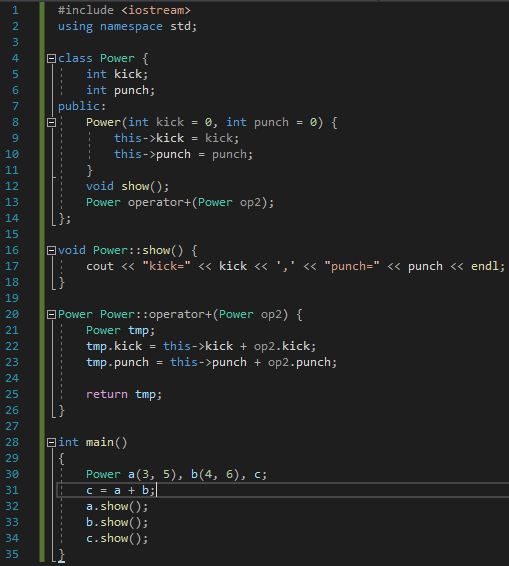
Power tmp;

tmp.kick = this->kick + op2.kick;

tmp.punch = this->punch + op2.punch;

return tmp;

}

예제7-4 

==연산자 중복 연산자 착안 두 객체의 kick, punch가 서로 같으면 true 반환

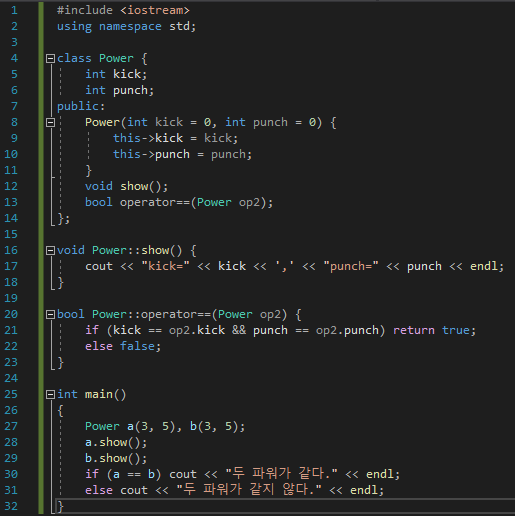
연산자 함수 선언 컴파일러의 식 변환 a==b 🡪 a.==(b)

연산자 함수 코드 bool Power::operator==(Power op2) {

if (kick == op2.kick && punch == op2.punch) return true;

else return false;

}

예제7-5 

+=연산자 중복 연산자 착안 a객체, b객체를 합쳐 새로운 a객체를 갱신하는 것

연산자 함수 선언 복사본을 리턴하는 것이 아닌 객체의 참조를 리턴해야함

컴파일러의 식 변형 ex) a+=b 🡪 a.+=(b)

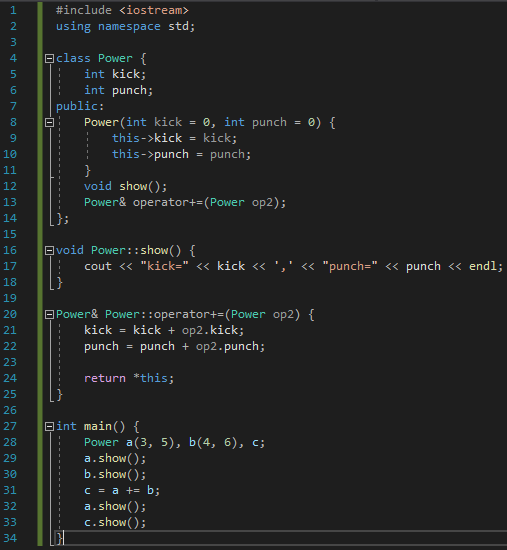
연산자 함수 코드 Power& Power::operator+=(Power op2) {

kick = kick + op2.kick;

punch = punch + op2.punch;

return \*this;

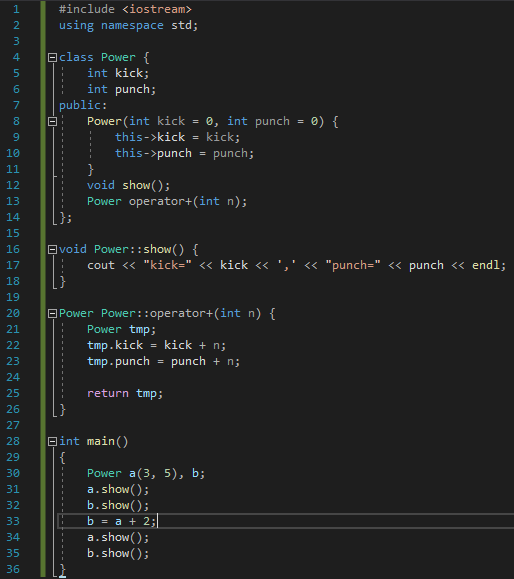
}

예제7-6 

+연산자 작성 실습: b=a+2 연산자 착안 a+2의 의미를 kick과 punch값에 각각 2를 더한 것으로 정의한다

연산자 함수 선언 컴파일러의 식변형 a+2 🡪 a.+(2)

연산자 함수 구현

예제7-7 

check time 1. (1) bool operator>(int n);

1. (2) Power operator&(Power op2)

2. (1) cout.<<(n)

2. (2) ostream operator<<(int n)

7.4 단항 연산자 중복

단항연산자 피연산자가 하나

전위연산자 연산자가 피연산자 앞에 오는 경우

후위연산자 연산자가 피연산자 뒤에 오는 경우

class Power {

int kick;

int punch;

public:

Power(int kick = 0, int punch = 0) { this->kick = kick; this->punch = punch;

}

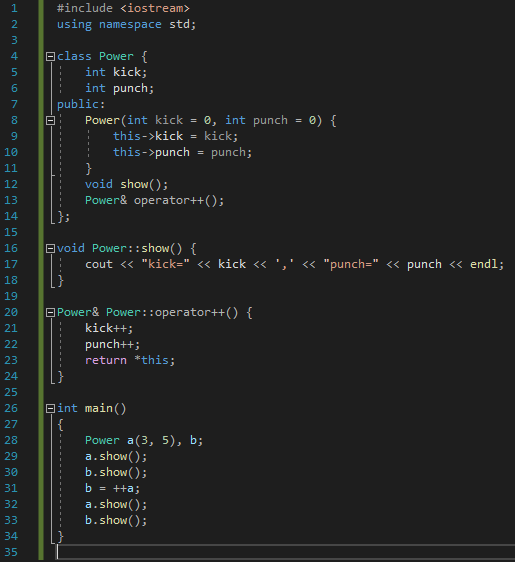
전위 ++ 연산자 중복 연산자 착안 ++객체

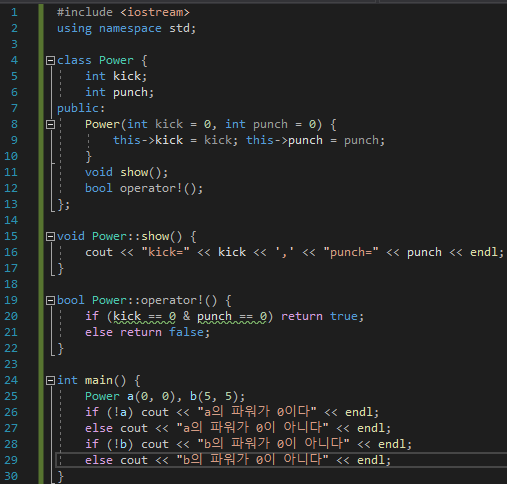
객체의 모든 멤버의 값을 1씩 증가시킨 후, 객체 a의 참조를 리턴

연산자 함수 선언 컴파일러의 식 변형 a++ 🡪 a.++()

Power& operator++();

참조를 리턴하는 연산자 함수 구현 자신의 kick, punch를 각각 1씩 증가시킨 후, 자신에 대한 참조 반환

예제7-8 

예제7-9 

후위 ++ 연산자 중복 후위연산자, 전위연산자를 구분하기 위해 후위연산자함수는 매개변수를 가지도록 선언됨

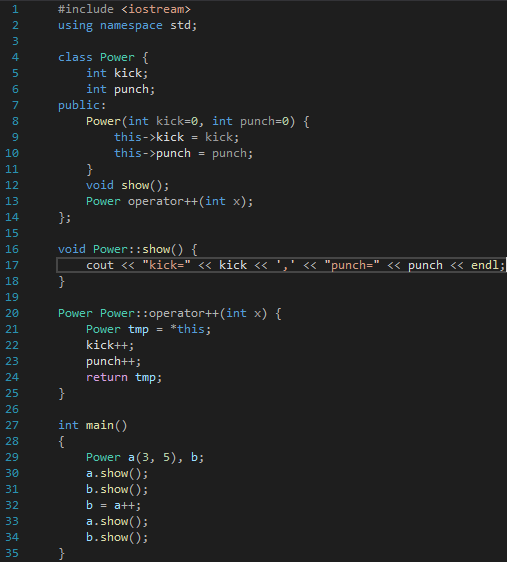
연산자 착안 ++객체

객체의 kick, punch를 각각 1씩 증가, 증가 이전의 객체리턴

연산자 함수 선언 컴파일러의 식 변환 a++ 🡪 a.++(임의정수) //주로 0

Power operator++(int x)

연산자 함수 구현 현재 객체를 복사해두고, 현재 객체의 kick과 punch를 1씩 증가시킨 후 복사해둔 객체 리턴

예제7-10 

7.5 프렌드를 이용한 연산자 중복

연산자 함수를 클래스 바깥의 외부 전역 함수로 작성 🡪 클래스 내부에서 friend로 선언

2+a 연산자 +연산자를 외부함수로 작성 Power a(3,4), b

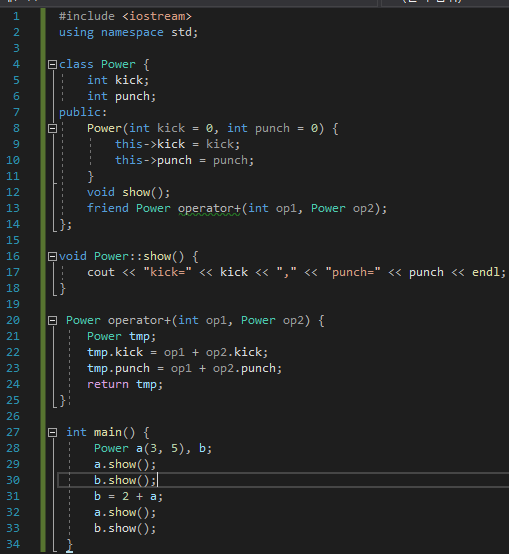
b=a+2

컴파일러의 식 변형 b=a+2 🡪 b=2.+(a) //2는 객체가 아님, 이 변형식은 틀림

b=a+2 🡪 b=+(2,a) //컴파일러는 이 식으로 변형시킴 🡪 외부함수로만 가능함

Power

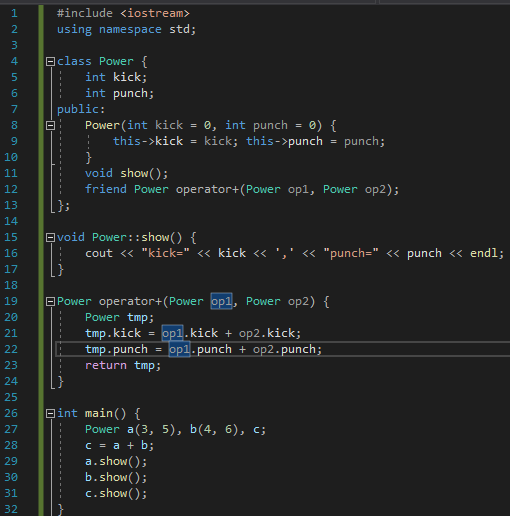
외부연산자 함수의 프렌드 선언 외부 연산자함수 내에서 private멤버인 kick, punch멤버변수 접근 불가 🡪 friend선언

예제7-11 

+연산자를 외부 프렌드 함수로 작성 Power a(3,4), b(4,5), c;

c=a+b;

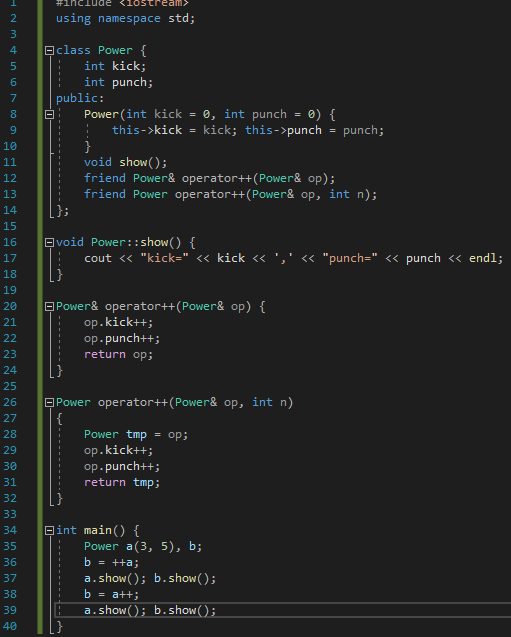
컴파일러의 식 변환 c=a+b 🡪 c =+(a,b)

예제 7-12 

단항연산자 ++를 프렌드로 작성하기 후위연산자 함수는 전위연산자 함수와 구분하기 위해 int형 변수를 추가로 가짐

참조 매개변수와 참조리턴 사용 두 연산자 모두 참조매개변수 사용

전위 연산자의 경우 참조 리턴

예제 7-13 

check time 1. 1

2. 컴파일러가 변형한 변형식에서 객체의 멤버함수로 연산자함수가 쓰이지 않을 경우

7.6 참조를 리턴하는 <<연산자 작성 실습

참조리턴

연산자 착안 << 오른쪽 피연산자 숫자를 <<왼쪽 피연산자 객체의 kick, punch에 더함

연산자 함수 구현 컴파일러의 식 변형 a<<3 🡪 a.<<(3)

예제7-14 