

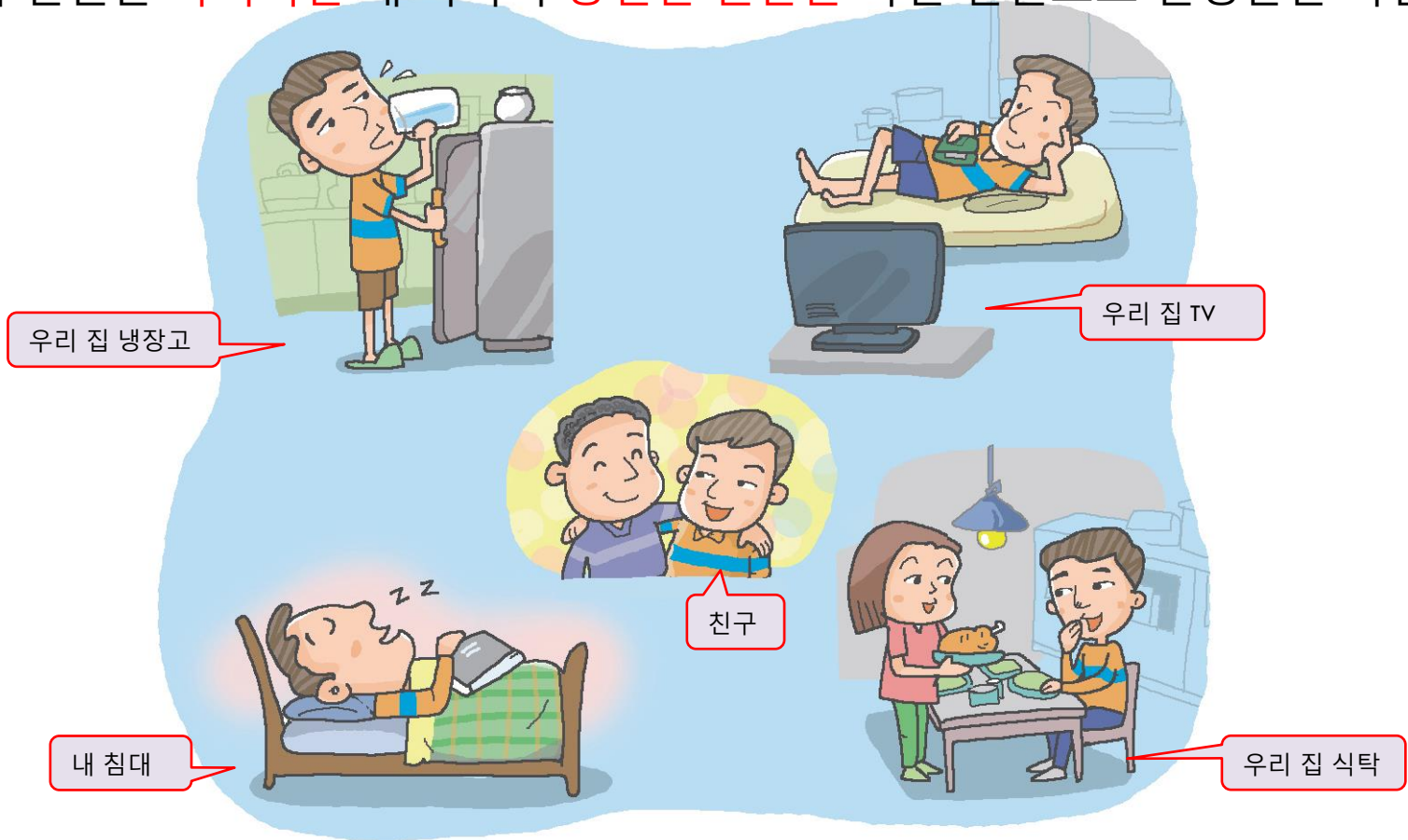
# 7장 프렌드와 연산자 중복

- 1) C++ 프렌드 개념
- 2) 연산자 중복
- 3) 이항연산자 중복
- 4) 단항연산자 중복
- 5) 프렌드를 이용한 연산자 중복

# 친구란?

친구?

내 가족의 일원은 아니지만 내 가족과 동일한 권한을 가진 일원으로 인정받은 사람



# C++ 프렌드

- 프렌드란?

- ▶ 클래스 외부에 작성된 함수에 대해 클래스 멤버 함수와 동일한 접근 자격을 부여하는 명령
- ▶ 즉, 클래스의 멤버가 아니지만 멤버의 권한을 가지도록 함
- ▶ friend 키워드 사용

```
class Rect { // Rect 클래스 선언
    ...
    friend bool equals(Rect r, Rect s);
};
```

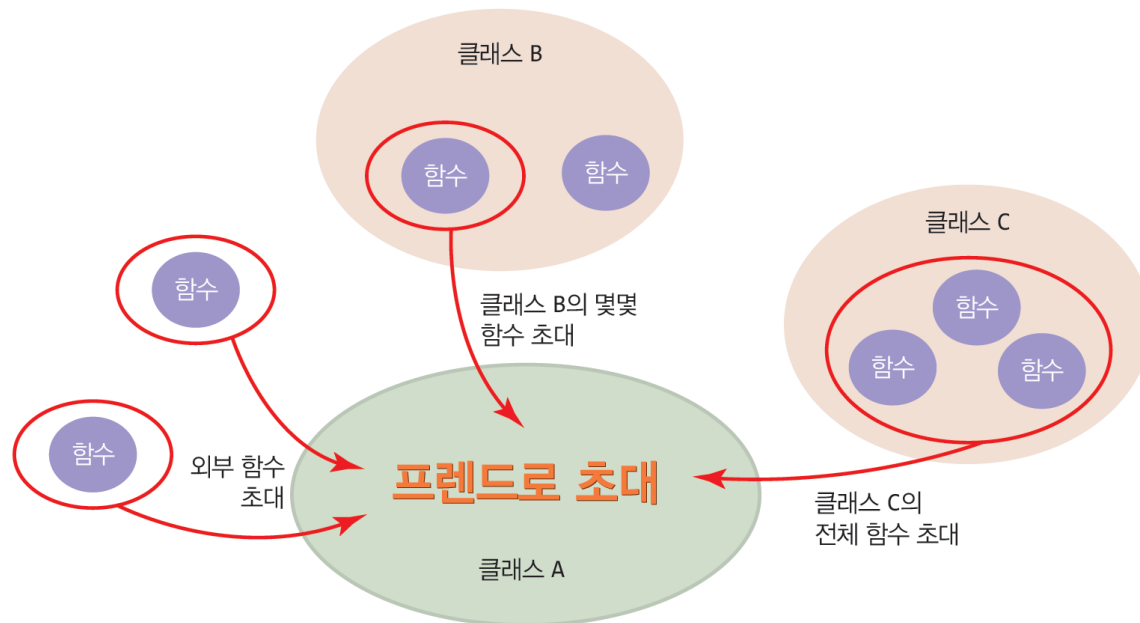
- 프렌드 개념이 필요한 경우

- ▶ 클래스의 멤버로 선언하기에는 무리가 있고, 클래스의 모든 멤버를 자유롭게 접근할 수 있는 일부 외부 함수 작성이 필요한 경우에 사용

# 프렌드로 초대하는 3 가지 유형

- 프렌드 함수가 되는 3 가지

- 1) 전역 함수 : 클래스 외부에 선언된 전역 함수
- 2) 다른 클래스의 멤버 함수 : 다른 클래스의 특정 멤버 함수
- 3) 다른 클래스 전체 : 다른 클래스의 모든 멤버 함수



# 프렌드 선언 3 종류

## 1. 외부 함수 equals()를 Rect 클래스에 프렌드로 선언

```
class Rect { // Rect 클래스 선언
    ...
    friend bool equals(Rect r, Rect s);
};
```

## 2. RectManager 클래스의 equals() 멤버 함수를 Rect 클래스에 프렌드로 선언

```
class Rect {
    .....
    friend bool RectManager::equals(Rect r, Rect s);
};
```

## 3. RectManager 클래스의 모든 멤버 함수를 Rect 클래스에 프렌드로 선언

```
class Rect {
    .....
    friend RectManager;
};
```

# 예제 7-1 프렌드 함수 만들기

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

```
class Rect;
```

Rect 클래스가 선언되기 전에 먼저 참조되는 컴파일 오류(forward reference)를 막기 위한 선언문

```
bool equals(Rect r, Rect s); // equals() 함수 선언
```

```
class Rect { // Rect 클래스 선언
```

```
    int width, height;
```

```
public:
```

```
    Rect(int width, int height) { this->width = width; this->height = height; }
```

```
    friend bool equals(Rect r, Rect s);
```

```
};
```

equals() 함수를  
프렌드로 선언

```
bool equals(Rect r, Rect s) { // 외부 함수
```

```
    if(r.width == s.width && r.height == s.height) return true;
```

```
    else return false;
```

```
}
```

equals() 함수는 private 속성을 가진  
width, height에 접근할 수 있다.

```
int main() {
```

```
    Rect a(3,4), b(4,5);
```

```
    if(equals(a, b)) cout << "equal" << endl;
```

```
    else cout << "not equal" << endl;
```

```
}
```

객체 a와 b는 동일한 크기의 사각  
형이므로 "not equal" 출력

not equal

## 예제 7-2 다른 클래스의 멤버 함수를 프렌드

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

class Rect;

class RectManager { // RectManager 클래스 선언
public:
    bool equals(Rect r, Rect s);
};

class Rect { // Rect 클래스 선언
    int width, height;
public:
    Rect(int width, int height) { this->width = width; this->height = height; }
    friend bool RectManager::equals(Rect r, Rect s);
};

bool RectManager::equals(Rect r, Rect s) {
    if(r.width == s.width && r.height == s.height) return true;
    else return false;
}

int main() {
    Rect a(3,4), b(3,4);
    RectManager man;

    if(man.equals(a, b)) cout << "equal" << endl;
    else cout << "not equal" << endl;
}
```

Rect 클래스가 선언되기 전에 먼저 참조되는 컴파일 오류(forward reference)를 막기 위한 선언문

RectManager 클래스의 equals() 멤버를 프렌드로 선언

객체 a와 b는 동일한 크기의 사각형이므로 "equal" 출력

equal

## 예제 7-3 다른 클래스 전체를 프렌드로 선언

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

```
class Rect;
```

Rect 클래스가 선언되기 전에 먼저 참조되는 컴파일 오류(forward reference)를 막기 위한 선언문

```
class RectManager { // RectManager 클래스 선언
```

```
public:
```

```
    bool equals(Rect r, Rect s);
```

```
    void copy(Rect& dest, Rect& src);
```

```
};
```

```
class Rect { // Rect 클래스 선언
```

```
    int width, height;
```

```
public:
```

```
    Rect(int width, int height) { this->width = width; this->height = height; }
```

```
    friend RectManager;
```

```
};
```

RectManager 클래스를 프렌드 함수로 선언

```
bool RectManager::equals(Rect r, Rect s) { // r과 s가 같으면 true 리턴
```

```
    if(r.width == s.width && r.height == s.height) return true;
```

```
    else return false;
```

```
}
```

```
void RectManager::copy(Rect& dest, Rect& src) { // src를 dest에 복사
```

```
    dest.width = src.width; dest.height = src.height;
```

```
}
```

```
int main() {
```

```
    Rect a(3,4), b(5,6);
```

```
    RectManager man;
```

객체 b의 width, height 값이 a와 같아진다.

```
    man.copy(b, a); // a를 b에 복사한다.
```

```
    if(man.equals(a, b)) cout << "equal" << endl;
```

```
    else cout << "not equal" << endl;
```

```
}
```

equal

man.copy(b,a)를 통해 객체 b와 a의 크기가 동일하므로 "equal" 출력





연산자 중복

# 연산자 중복

- 일상 생활에서의 기호 사용

- ▶ + 기호의 사례

- ▶ 숫자 더하기 :  $2 + 3 = 5$

- ▶ 색 혼합 : 빨강 + 파랑 = 보라

- ▶ 생활 : 남자 + 여자 = 결혼

- ▶ + 기호를 숫자와 물체(객체)에 적용, 중복 사용

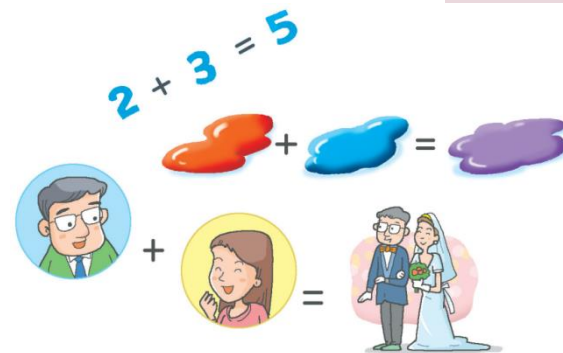
- ▶ + 기호를 숫자가 아닌 곳에도 사용하여 간결한 의미 전달

- 연산자 중복은 연산자 오버로딩 => 다형성의 구현

- ▶ 함수의 중복(오버로딩)과 같이 연산자도 하나의 함수라는 개념을 사용하여 중복 정의

- C++ 언어에서의 연산자 중복

- ▶ C++ 언어에 있는 연산자에 새로운 의미를 재정의하여 사용



# 연산자 중복의 사례 : + 연산자에 대해

- 정수 더하기

```
int a=2, b=3, c;  
c = a + b; // + 결과는 5. 정수가 피연산자일 때 2와 3을 더하기
```

- 문자열 합치기

```
string a="C", c;  
c = a + "++"; // + 결과는 "C++". 문자열이 피연산자일 때 두 개의 문자열 합치기
```

- 색 섞기

```
Color a(BLUE), b(RED), c;  
c = a + b; // c = VIOLET. a, b의 두 색을 섞은 새로운 Color 객체 c
```

- 배열 합치기

```
SortedArray a(2,5,9), b(3,7,10), c;  
c = a + b; // c = {2,3,5,7,9,10}. 정렬된 두 배열을 결합한(merge) 새로운 배열 생성
```

# 연산자 중복의 특징

- C++에 본래 있는 연산자만 중복 가능
  - ▶ `3%%5` // 컴파일 오류-C++에 `%%` 연산자가 없음
  - ▶ `6### 7` // 컴파일 오류
- 피 연산자 타입이 다른 새로운 연산을 정의하는 것임
  - ▶ 수+객체, 객체+수, 객체+객체
- 연산자는 함수 형태로 구현
  - ▶ 연산자 함수(operator function)
- 반드시 클래스와 관계를 가짐
  - ▶ 피연산자에 객체를 동반하기 때문에 반드시 클래스의 멤버함수로 구현하든지, 전역변수로 구현할 경우 클래스의 프렌드 함수로 구현해야 함

# 연산자 중복의 특징

- 피연산자의 개수를 바꿀 수 없음
  - ▶ 단항 연산자(!, ++, -- 등)은 단항 연산자로 구현해야 하고
  - ▶ 이항 연산자(+, -, \*, / 등)은 이항 연산자로 구현해야 한다.
- 연산의 우선 순위 변경 안됨
- 모든 연산자가 중복 가능하지 않음

# 연산자 중복의 특징

- 중복 가능한 연산자들

+	-	*	/	%	^	&
	~	!	=	<	>	+=
--	*=	/=	%=	^=	&=	=
<<	>>	>>=	<<=	==	!=	>=
<=	&&		++	--	->*	,
->	[]	()	new	delete	new[]	delete[]

- 중복 불가능한 연산자들

.	.*	::(범위지정 연산자)	? : (3항 연산자)
---	----	--------------	--------------

# 연산자 중복을 위한 2가지 방법

- 연산자 함수 구현 방법 2 가지
  1. 클래스의 멤버 함수로 구현하는 방법
  2. 외부 함수로 구현하고 클래스에 프렌드 함수로 선언하는 방법

# 연산자 함수 형식

- 연산자 함수 형식

- ▶ 사례

리턴타입 **operator** 연산자(매개변수리스트);

```
Color a(BLUE), b(RED), c;
```

```
c = a + b; // a와 b를 더하기 위한 + 연산자 작성 필요
```

```
if(a == b) { // a와 b를 비교하기 위한 == 연산자 작성 필요
```

```
...
```

```
}
```



# +와 == 연산자의 작성 사례

## 1) 클래스의 멤버 함수로 작성되는 경우

```
class Color {  
    ...  
    Color operator+ (Color op2); //왼쪽 피연산자가 객체 자신이고 오른쪽 피연산자가 op2에 전달  
    bool operator== (Color op2); //왼쪽 피연산자가 객체 자신이고 오른쪽 피연산자가 op2에 전달  
};
```

# +와 == 연산자의 작성 사례

## 2) 외부 함수로 구현되고 클래스에 프렌드로 선언되는 경우

```
Color operator + (Color op1, Color op2) {  
    ...  
}  
bool operator == (Color op1, Color op2) {  
    ...  
}  
  
class Color {  
    ...  
    friend Color operator+ (Color op1, Color op2);  
    friend bool operator== (Color op1, Color op2);  
};
```

# 연산자 함수 구현 방법 - 클래스의 멤버 함수로 구현

- 연산자 중복 설명에 사용할 클래스

```
class Power { // 에너지를 표현하는 파워 클래스
    int kick; // 발로 차는 힘
    int punch; // 주먹으로 치는 힘

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

- 클래스 Power에 이항 연산자 +, ==, += 연산자 함수와 단항 연산자 전위 ++, !, 후위 ++ 연산자 함수를 구현해 보자.

# 이항 연산자 중복 : + 연산자

`c = a + b;`

컴파일러에 의한 변환

`c = a . + ( b );`

`Power a(3,5), b(4,6), c;`

```
class Power {  
    int kick;  
    int punch;  
public:  
    .....  
    Power operator+ (Power op2);  
};
```

리턴 타입

오른쪽 피연산자  
b가 op2에 전달

Power a

```
Power Power::operator+(Power op2) {  
    Power tmp;  
    tmp.kick = this->kick + op2.kick;  
    tmp.punch = this->punch + op2.punch;  
    return tmp;  
}
```

+ 연산자 함수 코드

## 예제 7-4 두 개의 Power 객체를 더하는 + 연산자 작성

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    Power operator+ (Power op2); // + 연산자 함수 선언
};

void Power::show() {
    cout << "kick=" << kick << ' ' << "punch=" << punch << endl;
}

Power Power::operator+(Power op2) {
    Power tmp; // 임시 객체 생성
    tmp.kick = this->kick + op2.kick; // kick 더하기
    tmp.punch = this->punch + op2.punch; // punch 더하기
    return tmp; // 더한 결과 리턴
}
```

+ 연산자 멤버 함수 구현

```
int main() {
    Power a(3,5), b(4,6), c;
    c = a + b; // 파워 객체 + 연산
    a.show();
    b.show();
    c.show();
}
```

객체 a의  
operator+() 멤버  
함수 호출

```
kick=3,punch=5
kick=4,punch=6
kick=7,punch=11
```

객체 a, b, c 순  
으로 출력

# == 연산자 중복

**a == b**

컴파일러에 의한 변환

**a . == ( b )**

Power a(3,5), b(4,6), c;

리턴 타입

```
class Power {  
    .....  
public:  
    bool operator==(Power op2);  
};
```

오른쪽 피연산자  
b가 op2에 전달

Power a

```
bool Power::operator==(Power op2) {  
    if(kick==op2.kick && punch==op2.punch)  
        return true;  
    else  
        return false;  
}
```

== 연산자 함수 코드

# 예제 7-5 두 개의 Power 객체를 비교하는 == 연산자 작성

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    bool operator==(Power op2); // == 연산자 함수 선언
};

void Power::show() {
    cout << "kick=" << kick << ' '
        << "punch=" << punch << endl;
}

bool Power::operator==(Power op2) {
    if(kick==op2.kick && punch==op2.punch) return true;
    else return false;
}
```

== 연산자 멤버 함수 구현

```
int main() {
    Power a(3,5), b(3,5); // 2 개의 동일한 파워 객체 생성
    a.show();
    b.show();
    if(a == b) cout << "두 파워가 같다." << endl;
    else cout << "두 파워가 같지 않다." << endl;
}
```

operator==( ) 멤버 함수 호출

kick=3,punch=5  
kick=3,punch=5  
두 파워가 같다.

# += 연산자 중복

`c = a += b;`

컴파일러에 의한 변환

`c = a . += ( b );`

```
class Power {
```

```
.....
```

```
public:
```

```
Power operator+=( Power op2);
```

```
};
```

리턴 타입

오른쪽 피연산자  
b가 op2에 전달

Power a

```
Power Power::operator+=(Power op2) {
```

```
    kick = kick + op2.kick;
```

```
    punch = punch + op2.punch;
```

```
    return *this;
```

```
}
```

주목

+= 연산자 함수 코드



## 예제 7-6 두 Power 객체를 더하는 += 연산자 작성

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    Power operator+=(Power op2); // += 연산자 함수 선언
};

void Power::show() {
    cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}

Power Power::operator+=(Power op2) {
    kick = kick + op2.kick; // kick 더하기
    punch = punch + op2.punch; // punch 더하기
    return *this; // 합한 결과 리턴
}
```

+= 연산자 멤버 함수 구현

```
int main() {
    Power a(3,5), b(4,6), c;
    a.show();
    b.show();
    c = a += b; // 파워 객체 더하기
    a.show();
    c.show();
}
```

operator+=( ) 멤버 함수 호출

```
kick=3,punch=5
kick=4,punch=6
kick=7,punch=11
kick=7,punch=11
```

a, b 출력

a+=b 후 a, c 출력

# + 연산자 작성 : $b = a + 2$ ;

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    Power operator+ (int op2); // + 연산자 함수 선언
};

void Power::show() {
    cout << "kick=" << kick << ' ' << "punch=" << punch << endl;
}

Power Power::operator+(int op2) {
    Power tmp; // 임시 객체 생성
    tmp.kick = kick + op2; // kick에 op2 더하기
    tmp.punch = punch + op2; // punch에 op2 더하기
    return tmp; // 임시 객체 리턴
}
```

+ 연산자 멤버 함수 구현

```
int main() {
    Power a(3,5), b;
    a.show();
    b.show();
    b = a + 2; // 파워 객체와 정수 더하기
    a.show();
    b.show();
}
```

operator+(int) 함수 호출

kick=3,punch=5	}	a, b 출력
kick=0,punch=0		
kick=3,punch=5	}	b = a + 2 후 a, b 출력
kick=5,punch=7		

# 단항 연산자 중복

- 단항 연산자
  - ▶ 피연산자가 하나 뿐인 연산자
    - ▶ 연산자 중복 방식은 이항 연산자의 경우와 거의 유사함
  - ▶ 단항 연산자 종류
    - ▶ 전위 연산자(prefix operator)
      - ▶ !op, ~op, ++op, --op
    - ▶ 후위 연산자(postfix operator)
      - ▶ op++, op--

# 전위 ++ 연산자 중복

**++a**

컴파일러에 의한 변환

**a . ++ ( )**

Power a(3,5), b(4,6), c;

class Power {

.....

public:

**Power operator++ ( );**

};

리턴 타입

매개 변수 없음

Power a

**Power Power::operator++ ( ) {**

// kick과 punch는 a의 멤버

kick++;

punch++;

return \*this; // 변경된 객체 자신(객체 a) 리턴

**}**

전위 ++ 연산자 함수 코드

## 예제 7-8 전위 ++ 연산자 작성

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    Power operator++ (); // 전위 ++ 연산자 함수 선언
};

void Power::show() {
    cout << "kick=" << kick << ' ' << "punch=" << punch << endl;
}

Power Power::operator++() {
    kick++;
    punch++;
    return *this; // 변경된 객체 자신(객체 a) 리턴
}
```

전위 ++ 연산자 멤버 함수 구현

```
int main() {
    Power a(3,5), b;
    a.show();
    b.show();
    b = ++a; // 전위 ++ 연산자 사용
    a.show();
    b.show();
}
```

operator++() 함수 호출

```
kick=3,punch=5
kick=0,punch=0
kick=4,punch=6
kick=4,punch=6
```

a, b 출력

b = ++a 후 a, b 출력

# 예제 7-9 Power 클래스에 ! 연산자 작성

! 연산자를 Power 클래스의 멤버 함수로 작성하라.

!a는 a의 kick, punch 파워가 모두 0이면 true, 아니면 false를 리턴한다.

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

```
class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    bool operator! (); // ! 연산자 함수 선언
};
```

```
void Power::show() {
    cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}
```

```
bool Power::operator!() {
    if(kick == 0 && punch == 0) return true;
    else return false;
}
```

operator!() 함수 호출

! 연산자 멤버 함수 구현

```
int main() {
    Power a(0,0), b(5,5);
    if(!a) cout << "a의 파워가 0이다." << endl; // ! 연산자 호출
    else cout << "a의 파워가 0이 아니다." << endl;
    if(!b) cout << "b의 파워가 0이다." << endl; // ! 연산자 호출
    else cout << "b의 파워가 0이 아니다." << endl;
}
```

a의 파워가 0이다.  
b의 파워가 0이 아니다.

# 후위 연산자 중복, ++ 연산자

**a++**

컴파일러에 의한 변환

**a . ++ ( 임의의 정수 )**

class Power {

.....

public:

**Power operator ++ (int x);**

};

리턴 타입

매개 변수

객체 a

임의의 정수를 매개변수로  
지정하는 이유  
=> 전위 연산자 중복과 구분하기 위해

```
Power Power::operator++(int x) {  
    Power tmp = *this; // 증가 이전 객체 상태 저장  
    kick++;  
    punch++;  
    return tmp; // 증가 이전의 객체(객체 a) 리턴  
}
```

후위 ++ 연산자 함수 코드

# 예제 7-10 후위 ++ 연산자 작성

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    Power operator++ (int x); // 후위 ++ 연산자 함수 선언
};

void Power::show() {
    cout << "kick=" << kick << ' '
        << "punch=" << punch << endl;
}

Power Power::operator++(int x) {
    Power tmp = *this; // 증가 이전 객체 상태를 저장
    kick++;
    punch++;
    return tmp; // 증가 이전 객체 상태 리턴
}
```

후위 ++ 연산자 멤버 함수 구현

```
int main() {
    Power a(3,5), b;
    a.show();
    b.show();
    b = a++; // 후위 ++ 연산자 사용
    a.show(); // a의 파워는 1 증가됨
    b.show(); // b는 a가 증가되기 이전 상태를 가짐
}
```

operator++(int) 함수 호출

```
kick=3,punch=5
kick=0,punch=0
kick=4,punch=6
kick=3,punch=5
```

a, b 출력

b = a++ 후 a, b 출력



# 연산자 함수 구현 방법 - 외부 함수로 구현

- 외부함수로 구현하고 클래스에 프렌드 함수로 선언하는 방법

- 예

- ▶ 상수 + 객체

```
Power a(3,4), b;  
b = 2 + a;
```

- ▶ 객체 + 객체

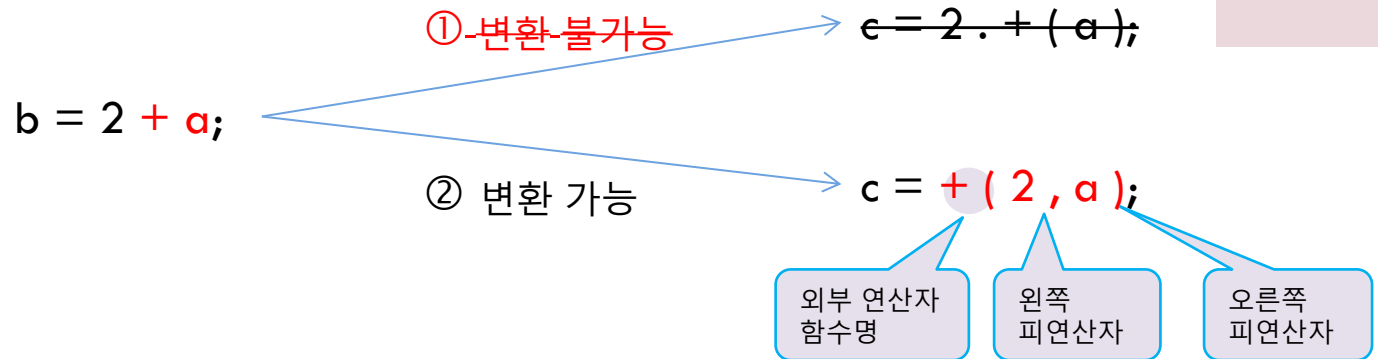
```
c = a+b;
```

- ▶ 단항 연산자를 프렌드 외부 함수로 구현

```
a++;
```

# 2 + a 덧셈을 위한 + 연산자 함수 작성

Power a(3,4), b;  
b = 2 + a;



b = 2 + a;

컴파일러에 의한 변환

b = + ( 2 , a );

매개변수

리턴 타입

```
Power operator+ (int op1, Power op2) {  
    Power tmp;  
    tmp.kick = op1 + op2.kick;  
    tmp.punch = op1 + op2.punch;  
    return tmp;  
}
```

# 예제 7-11 2+a를 위한 + 연산자 함수를 프렌드로 작성

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    friend Power operator+(int op1, Power op2); // 프렌드 선언
};

void Power::show() {
    cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}

Power operator+(int op1, Power op2) {
    Power tmp; // 임시 객체 생성
    tmp.kick = op1 + op2.kick; // kick 더하기
    tmp.punch = op1 + op2.punch; // punch 더하기
    return tmp; // 임시 객체 리턴
}
```

+ 연산자 함수를 외부 함수로 구현

private 속성인 kick, punch를 접근하도록 하기 위해,  
연산자 함수를 friend로 선언해야 함

```
int main() {
    Power a(3,5), b;
    a.show();
    b.show();
    b = 2 + a; // 파워 객체 더하기 연산
    a.show();
    b.show();
}
```

operator+(2, a) 함수 호출

```
kick=3,punch=5
kick=0,punch=0
kick=3,punch=5
kick=5,punch=7
```

a, b 출력

b = 2+a 후 a, b 출력

# + 연산자를 외부 함수로 구현

`c = a + b;`

컴파일러에 의한 변환

`c = + ( a , b );`

매개변수

리턴 타입

```
Power operator+ (Power op1, Power op2) {  
    Power tmp;  
    tmp.kick = op1.kick + op2.kick;  
    tmp.punch = op1.punch + op2.punch;  
    return tmp;  
}
```

# 예제 7-12 a+b를 위한 연산자 함수를 프렌드로 작성

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

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    void show();
    friend Power operator+(Power op1, Power op2); // 프렌드 선언
};

void Power::show() {
    cout << "kick=" << kick << ',' << "punch=" << punch << endl;
}

Power operator+(Power op1, Power op2) {
    Power tmp; // 임시 객체 생성
    tmp.kick = op1.kick + op2.kick; // kick 더하기
    tmp.punch = op1.punch + op2.punch; // punch 더하기
    return tmp; // 임시 객체 리턴
}
```

+ 연산자 함수 구현

```
int main() {
    Power a(3,5), b(4,6), c;
    c = a + b; // 파워 객체 + 연산
    a.show();
    b.show();
    c.show();
}
```

operator+(a,b) 함수 호출

```
kick=3,punch=5
kick=4,punch=6
kick=7,punch=11
```

객체 a, b, c 순  
으로 출력

# 단항 연산자 ++를 프렌드로 작성하기

(a) 전위 연산자

**++a**

컴파일러에 의한 변환

**++ ( a )**

리턴 타입

```
Power operator++ (Power& op) {  
    op.kick++;  
    op.punch++;  
    return op;  
}
```

0은 의미 없는 값으로  
전위 연산자와 구분  
하기 위함

(b) 후위 연산자

**a++**

컴파일러에 의한 변환

**++ ( a, 0 )**

리턴 타입

```
Power operator++ (Power& op, int x) {  
    Power tmp = op;  
    op.kick++;  
    op.punch++;  
    return tmp;  
}
```

## 예제 7-13

# ++연산자를 프렌드로 작성한 예

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

```
class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) { this->kick = kick; this->punch = punch; }
    void show();
    friend Power operator++(Power& op); // 전위 ++ 연산자 함수 프렌드 선언
    friend Power operator++(Power& op, int x); // 후위 ++ 연산자 함수 프렌드 선언
};

void Power::show() {
    cout << "kick=" << kick << ' ' << "punch=" << punch << endl;
}
```

```
Power operator++(Power& op) { // 전위 ++ 연산자 함수 구현
    op.kick++;
    op.punch++;
    return op; // 연산 결과 리턴
}
```

참조 매개 변수 사용에 주목

참조 매개 변수 사용에 주목

```
Power operator++(Power& op, int x) { // 후위 ++ 연산자 함수 구현
    Power tmp = op; // 변경하기 전의 op 상태 저장
    op.kick++;
    op.punch++;
    return tmp; // 변경 이전의 op 리턴
}
```

```
int main() {
    Power a(3,5), b;
    b = ++a; // 전위 ++ 연산자
    a.show(); b.show();

    b = a++; // 후위 ++ 연산자
    a.show(); b.show();
}
```

kick=4,punch=6  
kick=4,punch=6  
kick=5,punch=7  
kick=4,punch=6

b = ++a 실행 후  
a, b 출력

b = a++ 실행 후  
a, b 출력

# 사용자 정의 객체 입출력 연산자 <<, >> 구현

- C++의 입출력 연산자 <<, >> 도 연산자 오버로딩에 의해 구현된 것
  - ▶ 이름영역 std의 클래스 ostream, istream 클래스에서 정의
- 사용자 정의 객체에 <<, >> 연산자를 적용하려면 아래와 같은 전역함수(friend 함수) 정의

```
ostream& operator<<(ostream& os, class  
    ob)  
{  
    .....  
    return os;
```

```
istream& operator>>(istream& is, class  
    ob)  
{  
    .....  
    return is;
```



# 사용자 정의 객체 입출력 연산자 <<, >> 구현

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

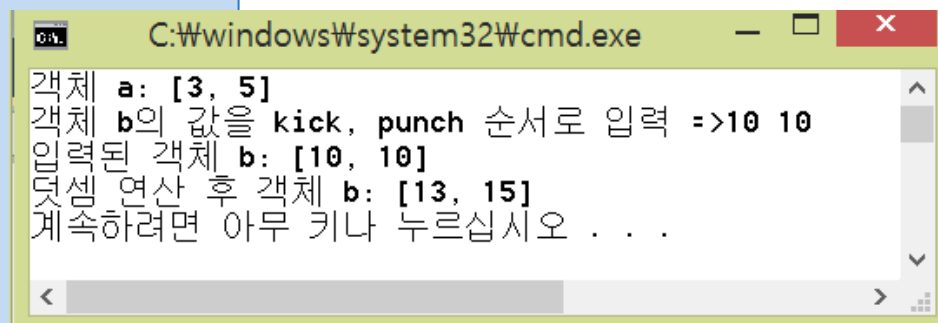
#include <iostream>
using namespace std;

class Power {
    int kick;
    int punch;
public:
    Power(int kick=0, int punch=0) {
        this->kick = kick; this->punch = punch;
    }
    friend Power operator+(Power op1, Power op2);
    friend ostream& operator<<(ostream& os, const Power& p);
    friend istream& operator>>(istream& is, Power& p);
};
```

# 사용자 정의 객체 입출력 연산자 <<, >> 구현

```
#include "op.h"
ostream& operator<<(ostream& os, const Power& p)
{
    os<<"["<<p.kick <<" , "<<p.punch<<"]"<<endl;
    return os;
}
istream& operator>>(istream& is, Power& p)
{
    is>>p.kick >>p.punch;
    return is;
}

int main()
{
    Power a(3,5), b;
    cout << "객체 a: " << a;
    cout << "객체 b의 값을 kick, punch 순서로 입력 =>";
    cin >> b;
    cout << "입력된 객체 b: " << b;
    b = a + b;
    cout << "덧셈 연산 후 객체 b: " << b;
}
```



```
C:\windows\system32\cmd.exe
객체 a: [3, 5]
객체 b의 값을 kick, punch 순서로 입력 =>10 10
입력된 객체 b: [10, 10]
덧셈 연산 후 객체 b: [13, 15]
계속하려면 아무 키나 누르십시오 . . .
```

# 연습문제 1- Point 클래스

- Point 클래스를 구현해서 main() 함수를 수행해 보자.

```
class MyPoint {
    int x, y;

public:
    MyPoint(int _x=0, int _y=0) { }
    MyPoint operator+(const MyPoint& p);
    MyPoint operator+(const int v);
    friend MyPoint operator+(int val, const MyPoint& p);
    MyPoint operator-(const MyPoint& p);
    MyPoint operator-(const int v);
    friend MyPoint operator-(int val, const MyPoint& p);
    MyPoint operator++(); //전위 연산자 ++
    MyPoint operator++ (int x); //후위 연산자 ++
    MyPoint operator--(); //전위 연산자 --
    MyPoint operator-- (int x); //후위 연산자 --
    bool operator<(const MyPoint& p);
    bool operator==(const MyPoint& p);
    bool operator!=(const MyPoint& p);
    friend ostream& operator<<(ostream& os, const MyPoint& p);
    friend istream& operator>>(istream& is, MyPoint& p);
};
```

# 연습문제 1 - 연산자 중복

```
int main()
{
    MyPoint p1(3,5), p2(5,5), p3;

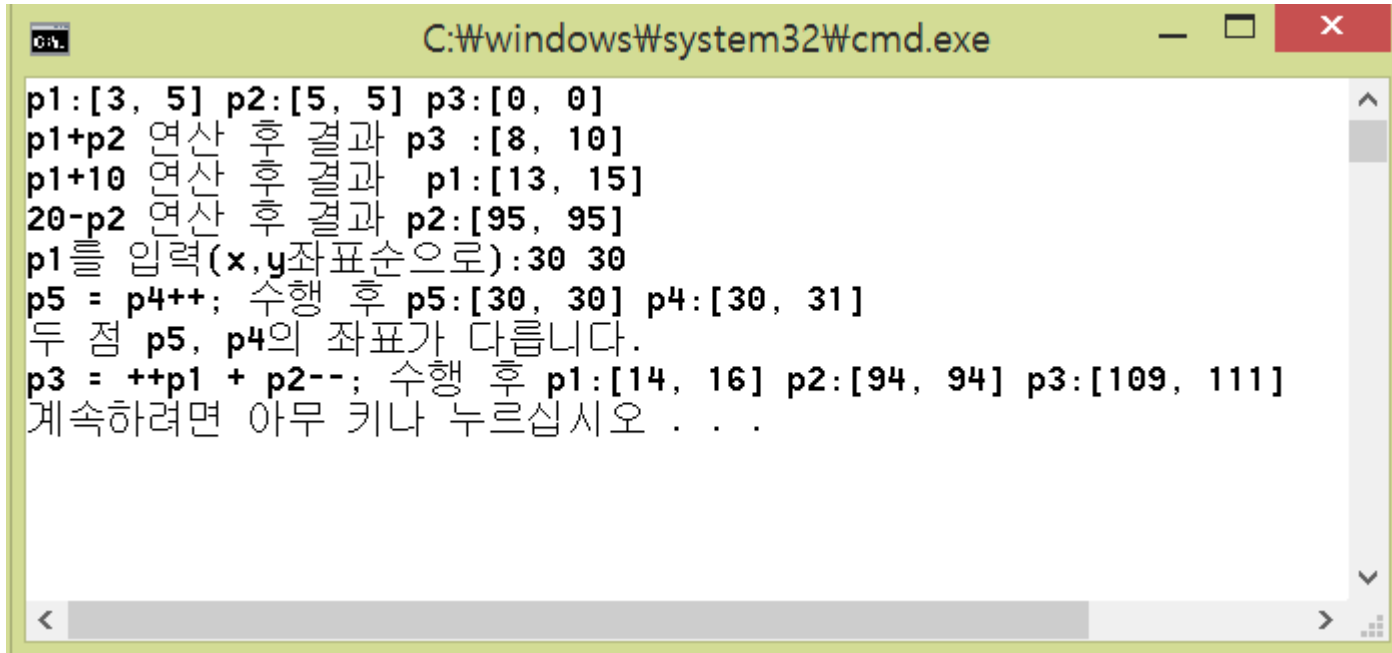
    cout << "p1:" << p1 << " p2:" << p2 << " p3:" << p3 << endl;
    p3 = p1+p2;
    cout << "p1+p2 연산 후 결과 p3 :" << p3 << endl;
    p1 = p1 + 10;
    cout << "p1+10 연산 후 결과 p1:" << p1 << endl;
    p2 = 100 - p2;
    cout << "20-p2 연산 후 결과 p2:" << p2 << endl;

    MyPoint p4, p5;
    cout << "p1를 입력(x,y좌표순으로):";
    cin >> p4;

    p5 = p4++;
    cout << "p5 = p4++; 수행 후 p5:" << p5 << " p4:" << p4 << endl ;
    if( p5 == p4 ) cout << "두 점 p5, p4의 좌표가 같습니다.\n";
    else cout << "두 점 p5, p4의 좌표가 다릅니다.\n";

    p3 = ++p1 + p2--;
    cout << "p3 = ++p1 + p2--; 수행 후 p1:" << p1 << " p2:" << p2 << " p3:" << p3 << endl ;
    return 0;
}
```

# 연습문제 1 - 연산자 중복



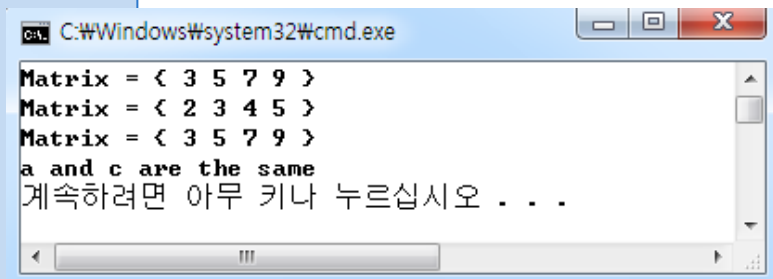
```
C:\Windows\system32\cmd.exe

p1:[3, 5] p2:[5, 5] p3:[0, 0]
p1+p2 연산 후 결과 p3 :[8, 10]
p1+10 연산 후 결과 p1:[13, 15]
20-p2 연산 후 결과 p2:[95, 95]
p1를 입력(x,y좌표순으로):30 30
p5 = p4++; 수행 후 p5:[30, 30] p4:[30, 31]
두 점 p5, p4의 좌표가 다릅니다.
p3 = ++p1 + p2--; 수행 후 p1:[14, 16] p2:[94, 94] p3:[109, 111]
계속하려면 아무 키나 누르십시오 . . .
```

## 연습문제 2 - Matrix 클래스

- 2차원 행렬을 추상화한 Matrix 클래스를 작성하고 show() 멤버 함수와 다음 연산이 가능하도록 연산자를 모두 구현 하시오.

```
int main() {  
    Matrix a(1,2,3,4), b(2,3,4,5), c;  
    c = a + b;  
    a += b;  
    a.show(); b.show(); c.show();  
    if(a == c)  
        cout << "a and c are the same" << endl;  
}
```



```
C:\Windows\system32\cmd.exe  
Matrix = < 3 5 7 9 >  
Matrix = < 2 3 4 5 >  
Matrix = < 3 5 7 9 >  
a and c are the same  
계속하려면 아무 키나 누르십시오 . . .
```

- ▶ 연산자 함수를 Matrix의 멤버 함수로 구현하라.
- ▶ 연산자 함수를 Matrix의 프렌드 함수로 구현하라.

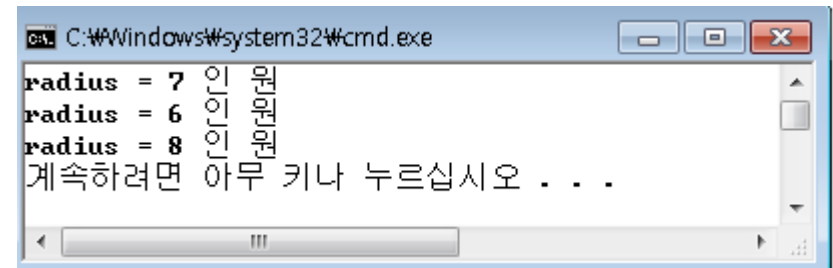
## 연습문제 3 - Circle 클래스

- 원을 추상화한 Circle 클래스는 다음과 같다

```
class Circle {  
    int radius;  
  
public:  
    Circle(int radius = 0) { this->radius = radius; }  
    void show() {  
        cout << "radius = " << radius << " 인 원" << endl;  
    }  
};
```

- 다음 연산이 가능하도록 연산자를 프렌드 함수로 작성하시오.

```
int main() {  
    Circle a(5), b(4);  
    ++a; // 반지름을 1 증가 시킨다.  
    b = a++; // 반지름을 1 증가 시킨다.  
    a.show();  
    b.show();  
    b = 1 + a;  
    b.show();  
}
```

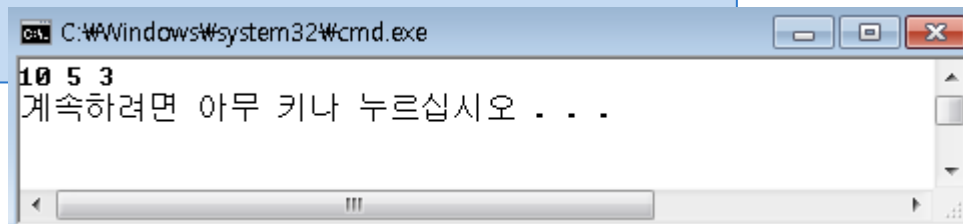


```
C:\Windows\system32\cmd.exe  
radius = 7 인 원  
radius = 6 인 원  
radius = 8 인 원  
계속하려면 아무 키나 누르십시오 . . .
```

## 연습문제 4 - Stack 클래스

- 스택 클래스 Stack을 만들고 푸시(push)용으로 << 연산자를 팝(pop)용으로 >> 연산자를, 비어 있는 스택인지를 알기 위해서는 ! 연산자를 작성하시오. main() 함수와 수행 결과는 다음과 같다.

```
int main() {  
    Stack stack;  
    stack << 3 << 5 << 10; // 3, 5, 10을 순서대로 푸시  
    while(true) {  
        if(!stack) break; // 스택 empty  
        int x;  
        stack >> x; // 스택의 탑에 있는 정수 팝  
        cout << x << ' ';  
    }  
    cout << endl;  
}
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window shows the output of the program: "10 5 3" followed by a space and the Korean text "계속하려면 아무 키나 누르십시오 . . .". The text is displayed in a monospaced font.