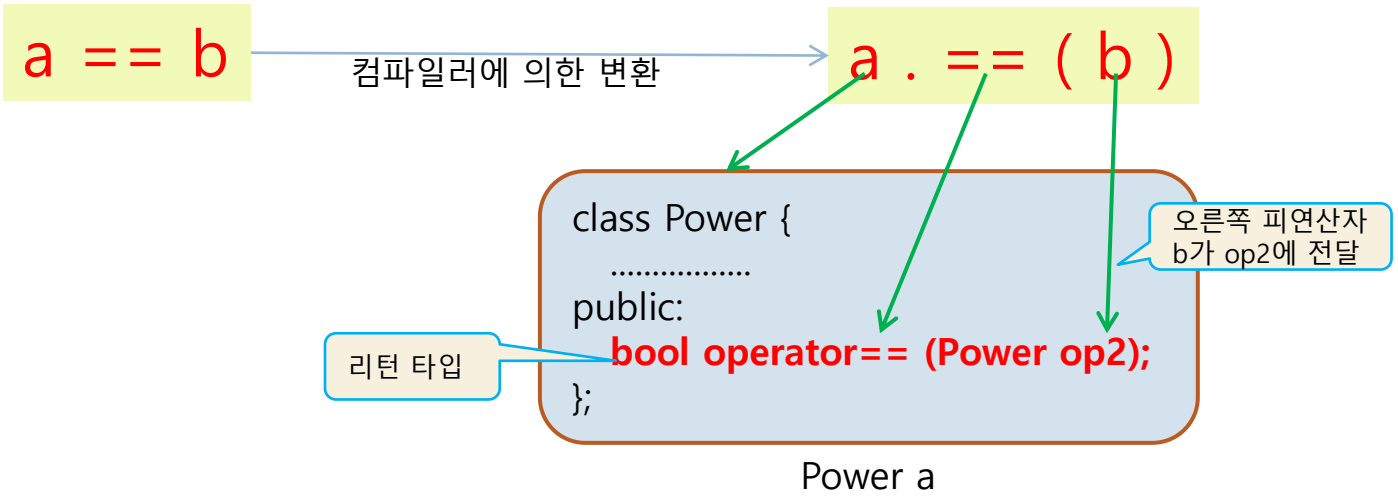
07

프렌드와 연산자 중복

== 연산자 중복




```
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;
}
```

== 연산자 멤버 함수 구현

operator==() 멤버 함수 호출

```
int main() {



}
```

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

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;
}
```

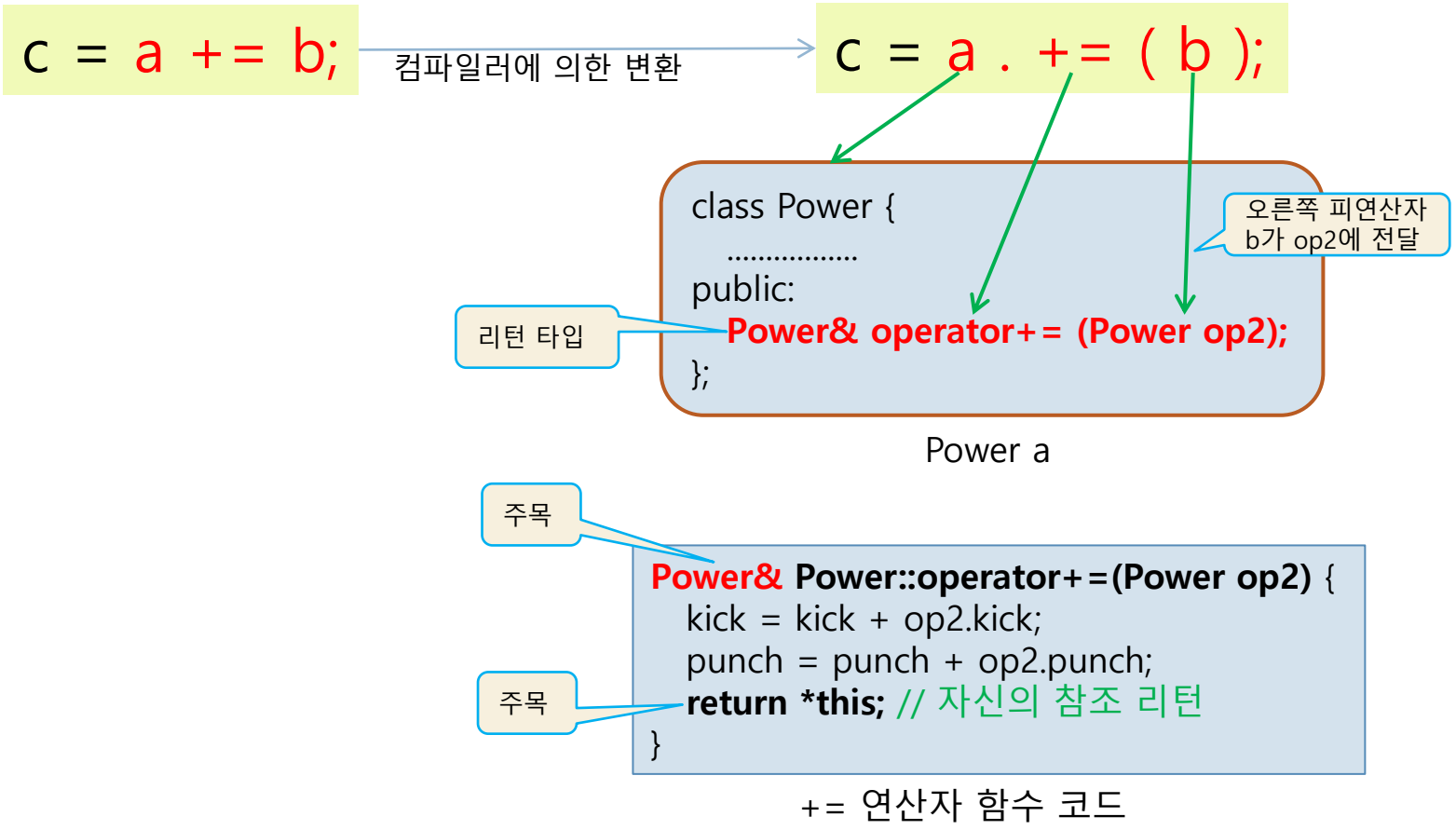
== 연산자 멤버 함수 구현

```
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);
};
오른쪽 피연산자
b가 op2에 전달
리턴 타입
Power a
주목
Power& Power::operator+=(Power op2) {
kick = kick + op2.kick;
punch = punch + op2.punch;
return *this; // 자신의 참조 리턴
}
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;
}
```

+= 연산자 멤버 함수 구현

operator+=() 멤버 함수 호출

```
int main() {



}
```

```
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

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(실습): 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;
}
```

+ 연산자 멤버 함수 구현

```
int main() {


}
```

operator+(int) 함수 호출

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

a, b 출력

b = a + 2 후 a, b 출력

+ 연산자 작성(실습): 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
kick=0,punch=0
kick=3,punch=5
kick=5,punch=7
```

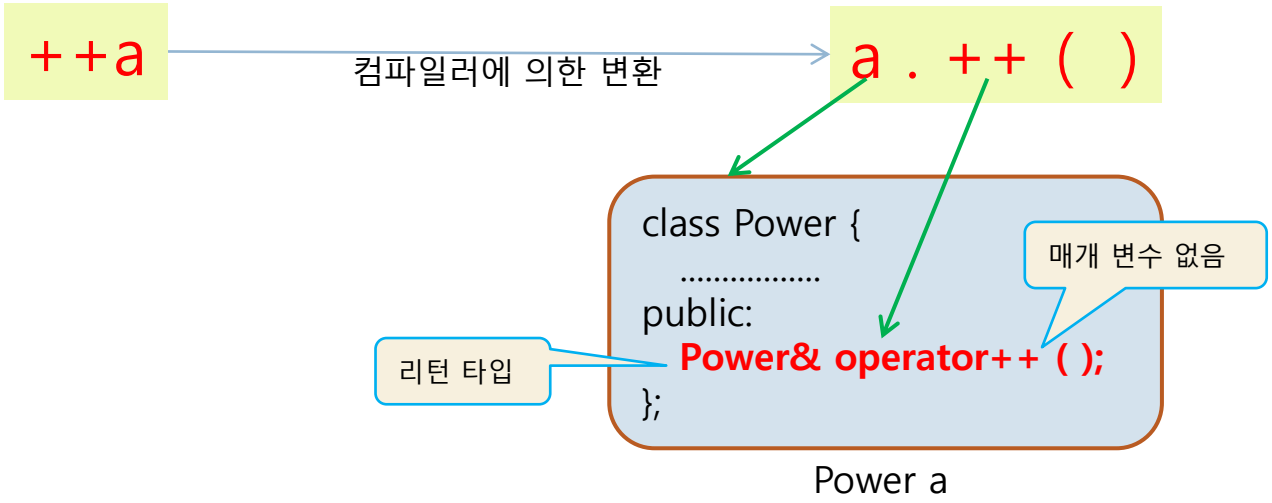
a, b 출력

b = a + 2 후 a, b 출력

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




```
Power& Power::operator++( ) {
   // kick과 punch는 a의 멤버
   kick++;
   punch++;
   return *this; // 변경된 객체 자신(객체 a)의 참조 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)의 참조 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(실습) 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;
}
```

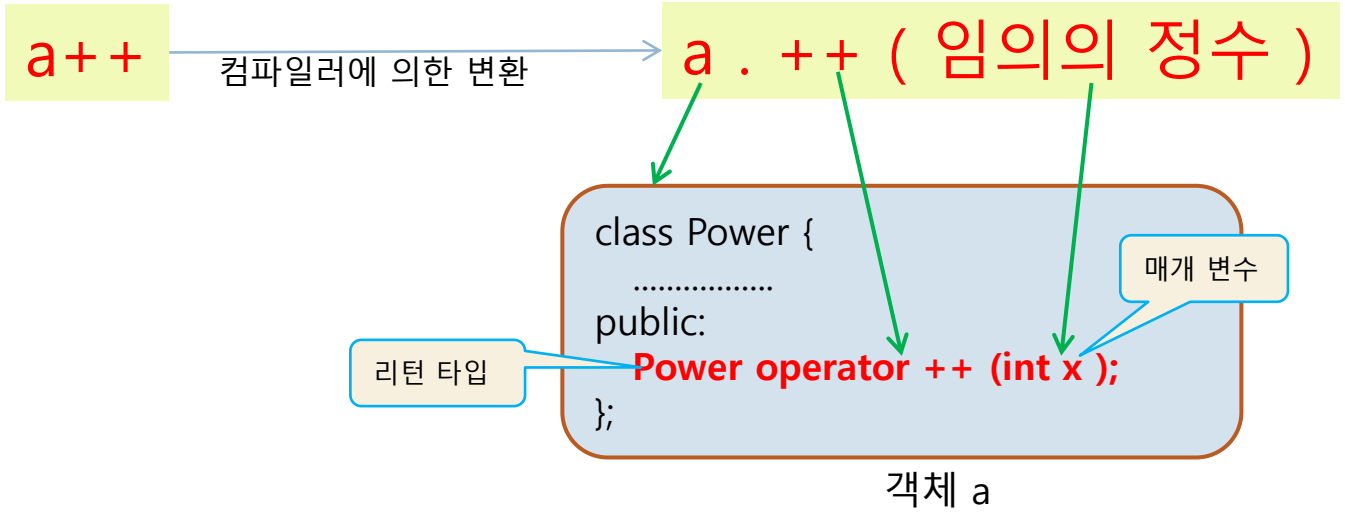
! 연산자 멤버 함수 구현

```
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;
}
```

operator!() 함수 호출

```
a의 파워가 0이다.
b의 파워가 0이 아니다.
```

후위 연산자 중복, ++ 연산자




```
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

2 + a 덧셈을 위한 + 연산자 함수 작성


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

b = 2 + a;

① ~~변환 불가능~~ → ~~b = 2 . + (a);~~

② 변환 가능 → b = + (2 , a);

- 외부 연산자 함수명
- 왼쪽 피연산자
- 오른쪽 피연산자

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();
}
```

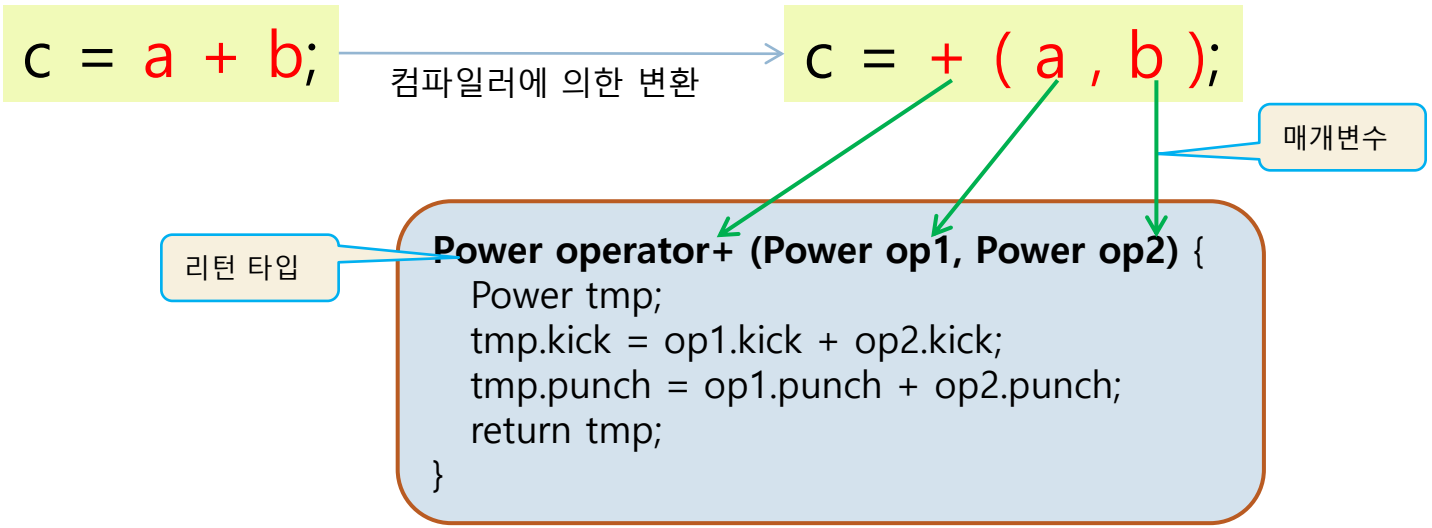
operator+(2, a) 함수 호출

```
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 프렌드 함수로 구현



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();
}
```

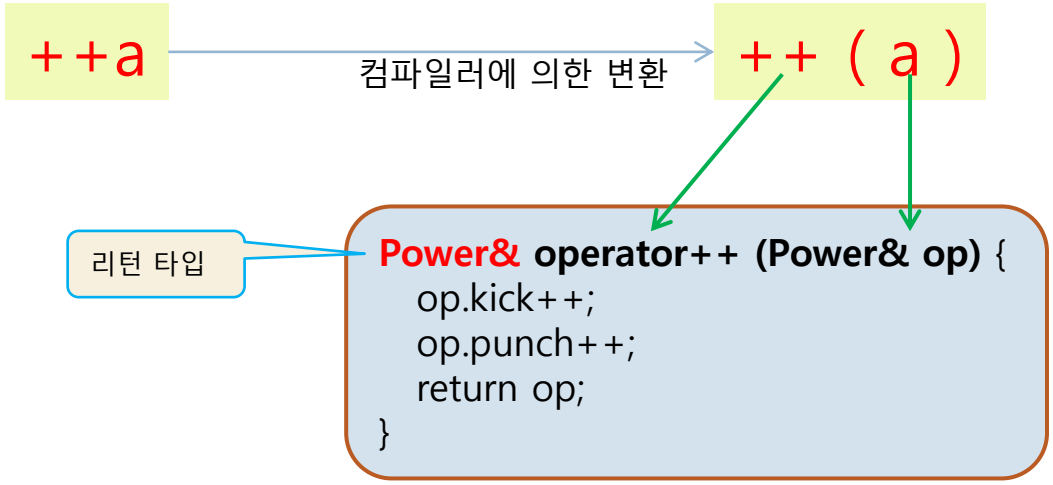
operator+(a,b) 함수 호출

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

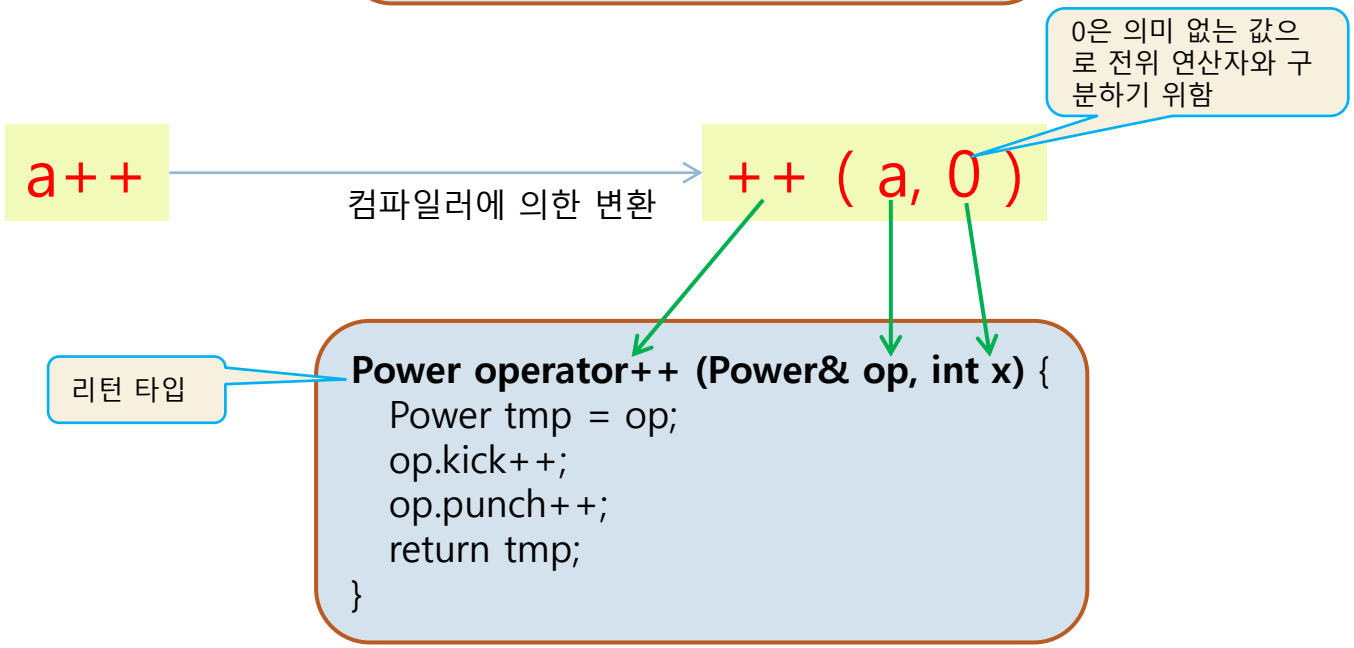
객체 a, b, c 순으로 출력

단항 연산자 ++를 프렌드로 작성하기

(a) 전위 연산자



(b) 후위 연산자



예제 7-13 ++연산자를 프렌드로 작성한 예

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

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();
}
```

참조 매개 변수 사용에 주목

참조 매개 변수 사용에 주목

```
#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;
}
```

```
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

예제 7-14 참조를 리턴하는 << 연산자 작성

Power 객체의 kick과 punch에 정수를 더하는 << 연산자를 멤버 함수로 작성하라

```
#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();
                                    // 연산 후 Power 객체의 참조 리턴
};

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

참조 리턴

```
int main() {
    Power a(1, 2);
    a << 3 << 5 << 6;
    a.show();
}
```

객체 a에 3, 5, 6이 순서대로 더해진다.

```
kick=15,punch=16
```