



학습 목표

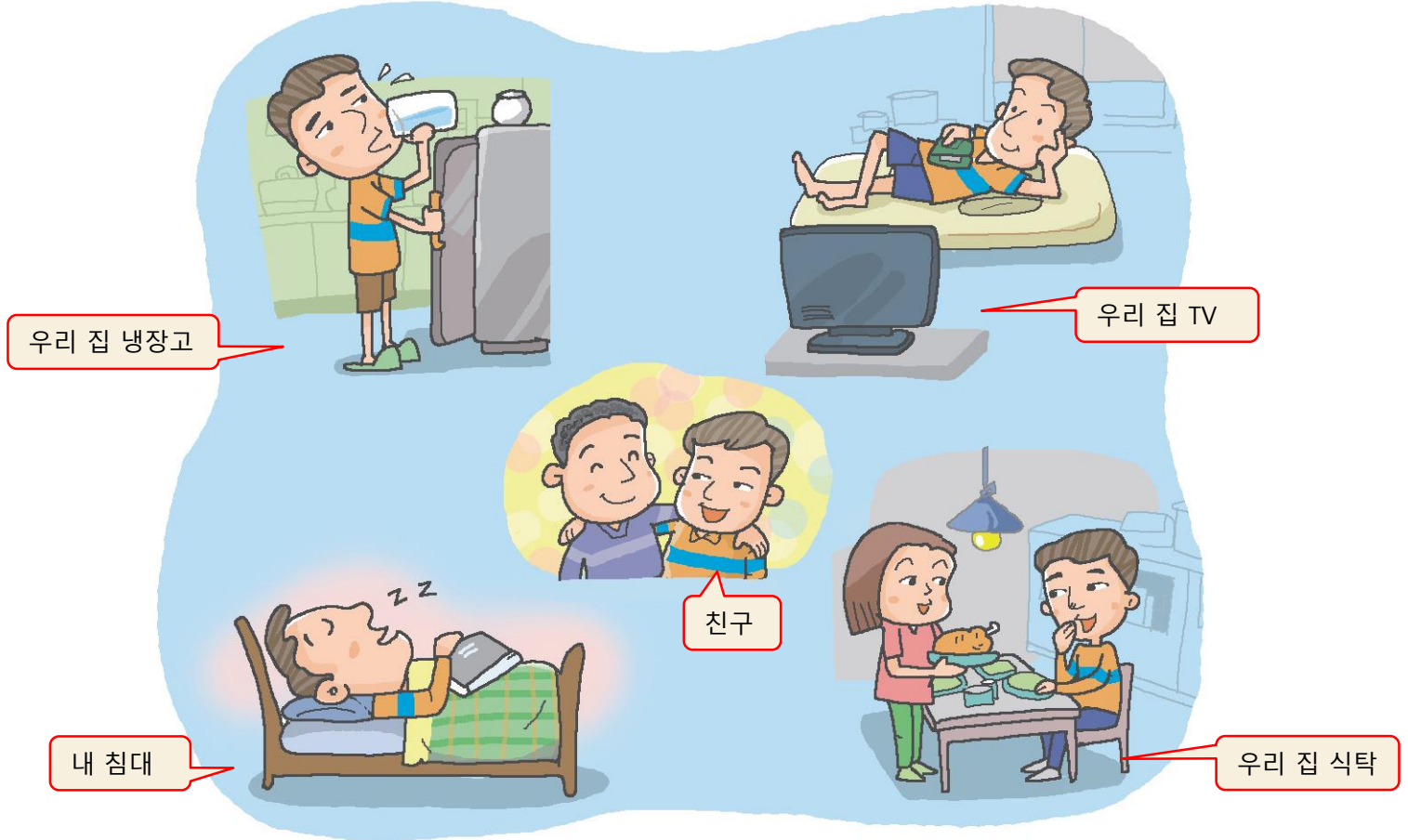
1. C++의 프렌드 함수의 개념을 이해하고, 프렌드 함수를 작성할 수 있다.
2. 연산자 중복의 개념을 이해하고
3. 연산자를 클래스 멤버 함수로 작성할 수 있다.
4. 연산자를 프렌드 함수로 작성할 수 있다.
5. 다양한 이항 연산자를 중복 할 수 있다.
6. 다양한 단항 연산자를 중복 할 수 있다.
7. 단항 연산자에서 전위 연산자와 후위 연산자를 구분하여 작성할 수 있다.

친구란?

3

친구?

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



C++ 프렌드

4

□ 프렌드 함수

▣ 클래스의 멤버 함수가 아닌 외부 함수

- 전역 함수
- 다른 클래스의 멤버 함수

▣ friend 키워드로 클래스 내에 선언된 함수

- 클래스의 모든 멤버를 접근할 수 있는 권한 부여
- 프렌드 함수라고 부름

▣ 프렌드 선언의 필요성

- 클래스의 멤버로 선언하기에는 무리가 있고, 클래스의 모든 멤버를 자유롭게 접근할 수 있는 일부 외부 함수 작성 시

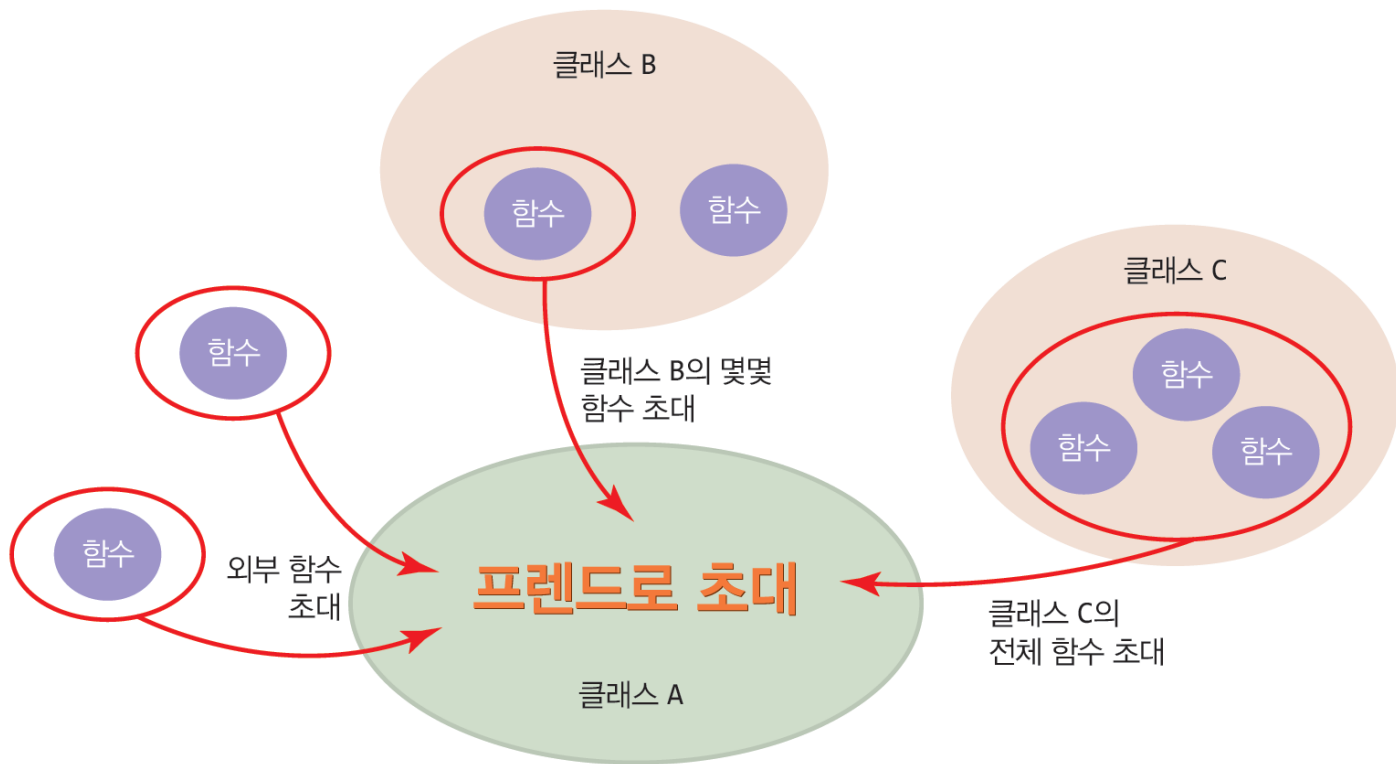
항목	세상의 친구	프렌드 함수
존재	가족이 아님. 외부인	클래스 외부에 작성된 함수. 멤버가 아님
자격	가족의 구성원으로 인정받음. 가족의 모든 살림살이에 접근 허용	클래스의 멤버 자격 부여. 클래스의 모든 멤버에 대해 접근 가능
선언	친구라고 소개	클래스 내에 friend 키워드로 선언
개수	친구의 명수에 제한 없음	프렌드 함수 개수에 제한 없음

프렌드로 초대하는 3 가지 유형

5

▣ 프렌드 함수가 되는 3 가지

- 전역 함수 : 클래스 외부에 선언된 전역 함수
- 다른 클래스의 멤버 함수 : 다른 클래스의 특정 멤버 함수
- 다른 클래스 전체 : 다른 클래스의 모든 멤버 함수



프렌드 선언 3 종류

6

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

7

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

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

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

```
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 사각형이므로 "not equal" 출력

not equal

예제 7-2 다른 클래스의 멤버 함수를 프렌드로 선언

8

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

```
class Rect;
```

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

```
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;
}
```

RectManager 클래스의 equals() 멤버를 프렌드로 선언

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

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

```
}
```

객체 a와 b는 동일한 크기의 사각형이므로 "equal" 출력

equal

예제 7-3 다른 클래스 전체를 프렌드로 선언

9

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

```
class Rect;
```

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

```
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

10

□ 일상 생활에서의 기호 사용

▣ + 기호의 사례

- 숫자 더하기 : $2 + 3 = 5$
- 색 혼합 : 빨강 + 파랑 = 보라
- 생활 : 남자 + 여자 = 결혼

▣ + 기호를 숫자와 물체에 적용, 중복 사용

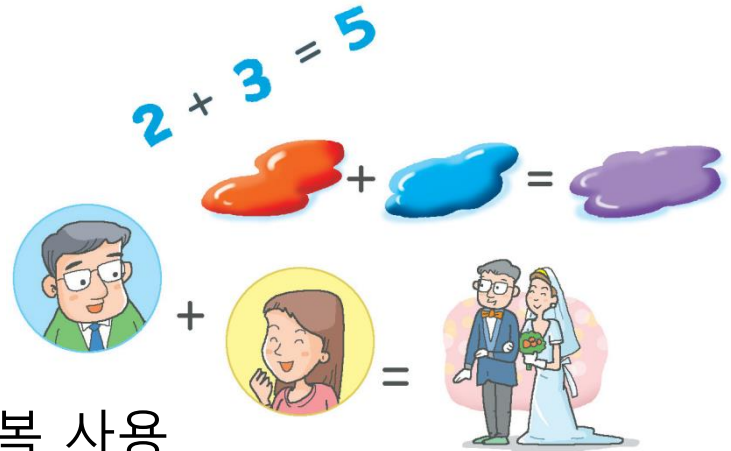
▣ + 기호를 숫자가 아닌 곳에도 사용

▣ 간결한 의미 전달

▣ 다형성

□ C++ 언어에서도 연산자 중복 가능

- ▣ C++ 언어에 본래부터 있던 연산자에 새로운 의미 정의
- ▣ 높은 프로그램 가독성



연산자 중복의 사례 : + 연산자에 대해

11

□ 정수 더하기

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

□ 문자열 합치기

```
string a="C", c;  
c = a + "++"; // + 결과 "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

12

- C++에 본래 있는 연산자만 중복 가능
 - ▣ 3%%5 // 컴파일 오류
 - ▣ 6## 7 // 컴파일 오류
- 피 연산자 타입이 다른 새로운 연산 정의
- 연산자는 함수 형태로 구현 - 연산자 함수(operator function)
- 반드시 클래스와 관계를 가짐
- 피연산자의 개수를 바꿀 수 없음
- 연산의 우선 순위 변경 안됨
- 모든 연산자가 중복 가능하지 않음

중복 가능한
연산자

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

중복 불가능한
연산자

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

연산자 함수

13

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

리턴타입 **operator**연산자(*매개변수리스트*);

+와 == 연산자의 작성 사례

14

연산자 함수 작성에 필요한 코드 사례

```
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  
    ...  
}
```

- 외부 함수로 구현되고
클래스에 프렌드로 선언되는 경우

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

- 클래스의 멤버 함수로 작성되는 경우

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

앞으로 연산자 함수 작성에 사용할 클래스

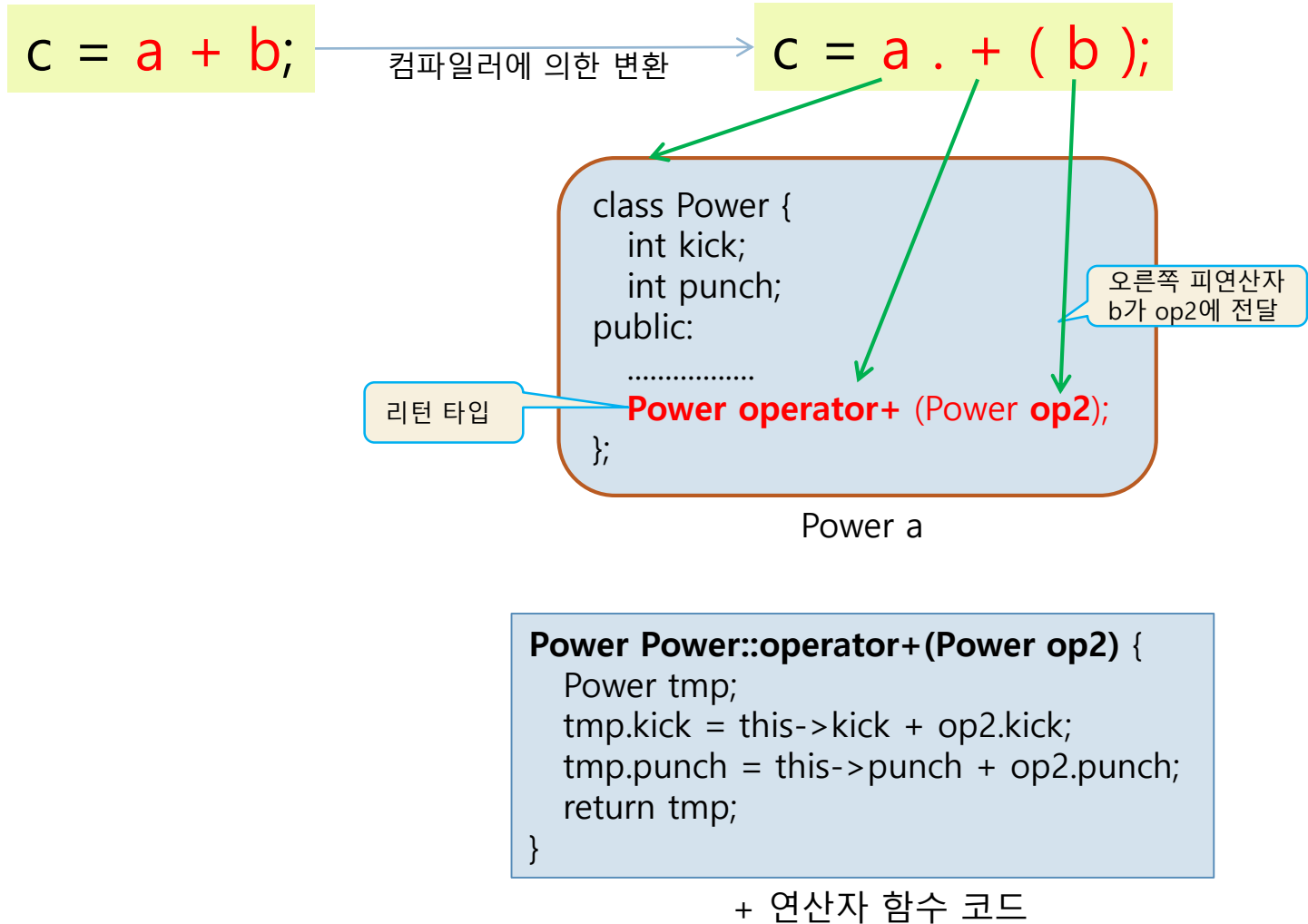
15

```
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;
    }
};
```

멤버 함수로 이항 연산자 구현

이항 연산자 중복 : + 연산자

17



예제 7-4 두 개의 Power 객체를 더하는 + 연산자 작성

18

```
#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) { // + 연산자 멤버 함수 구현
    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
}
```

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

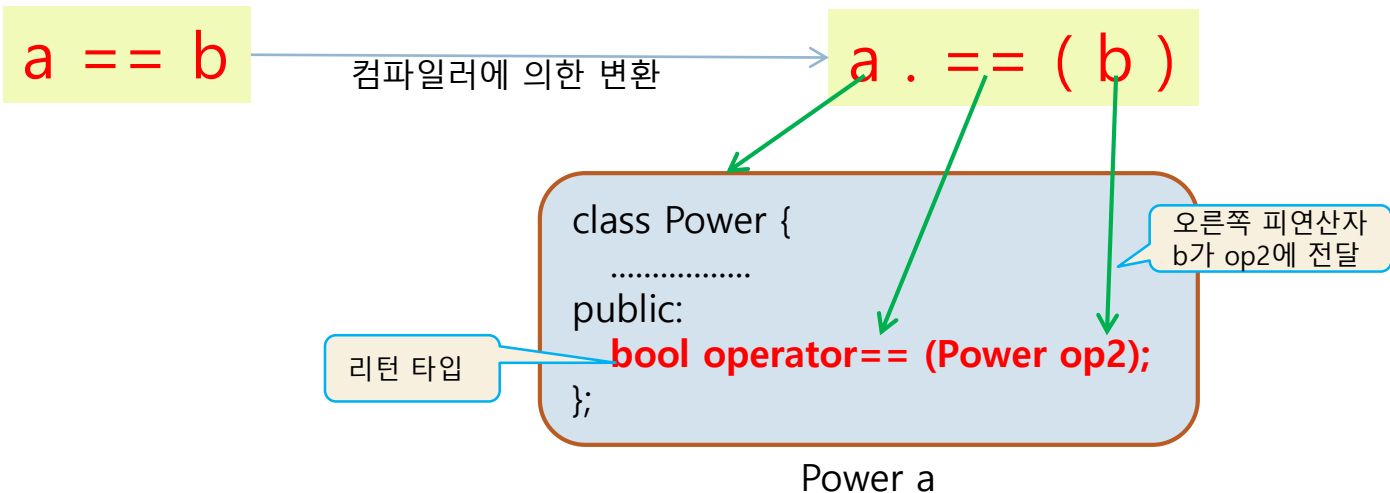
객체 a의
operator+() 멤버
함수 호출

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

객체 a, b, c
순으로 출력

== 연산자 중복

19



```
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

20

```
#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

21

`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

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

22

```
#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): $b = a + 2$;

23

```
#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

멤버 함수로 단항 연산자 구현

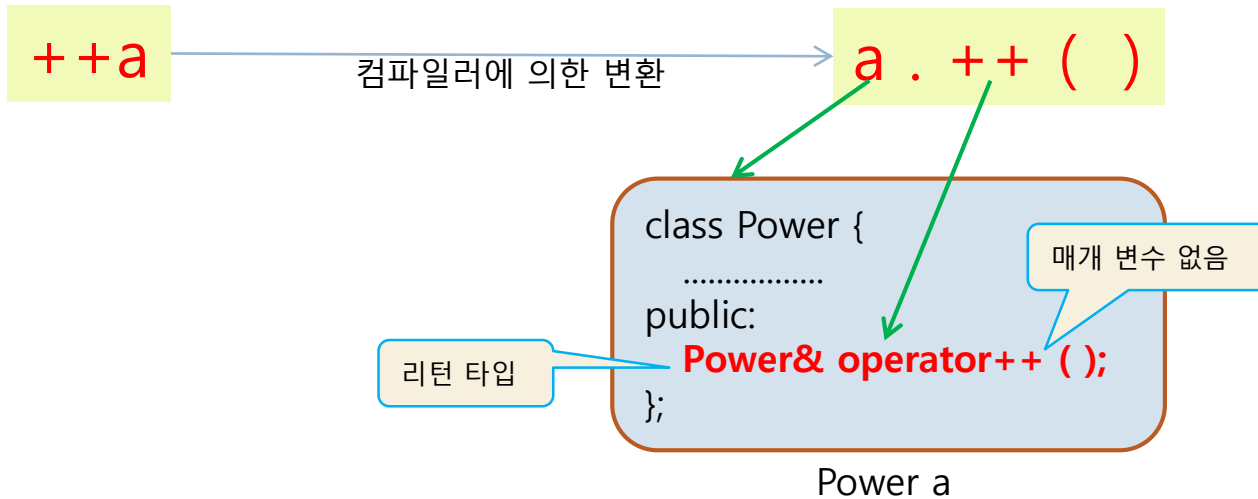
단항 연산자 중복

25

- 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

26



```
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

27

```
#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
}
```

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

operator++() 함수 호출

전위 ++ 연산자 멤버 함수 구현

```
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

28

! 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

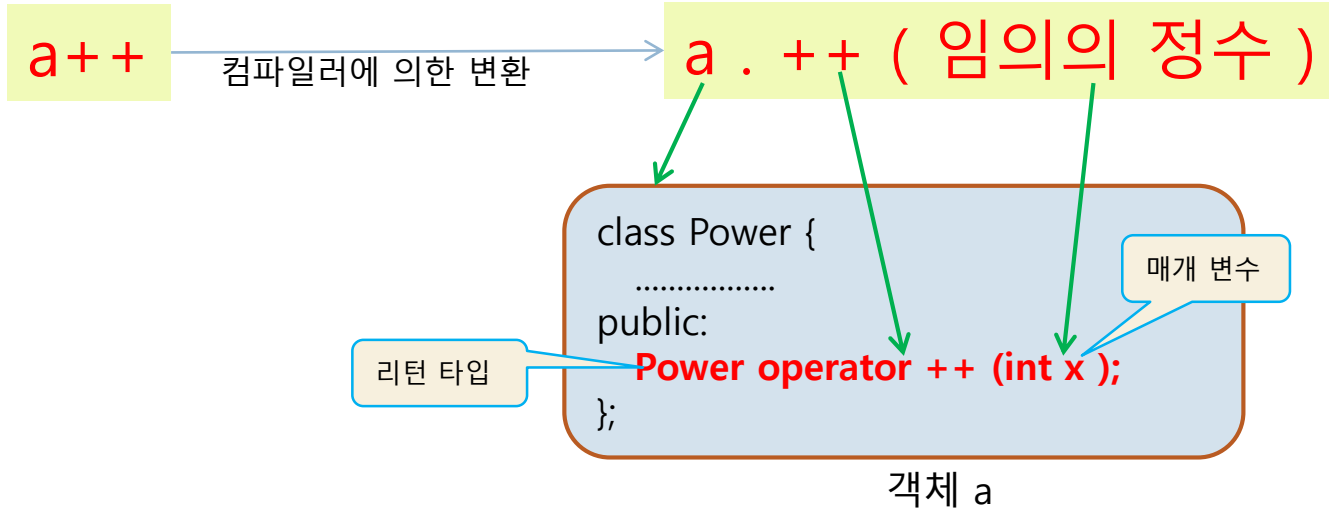
! 연산자 멤버 함수 구현

```
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

29



```
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

30

```
##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 }
```

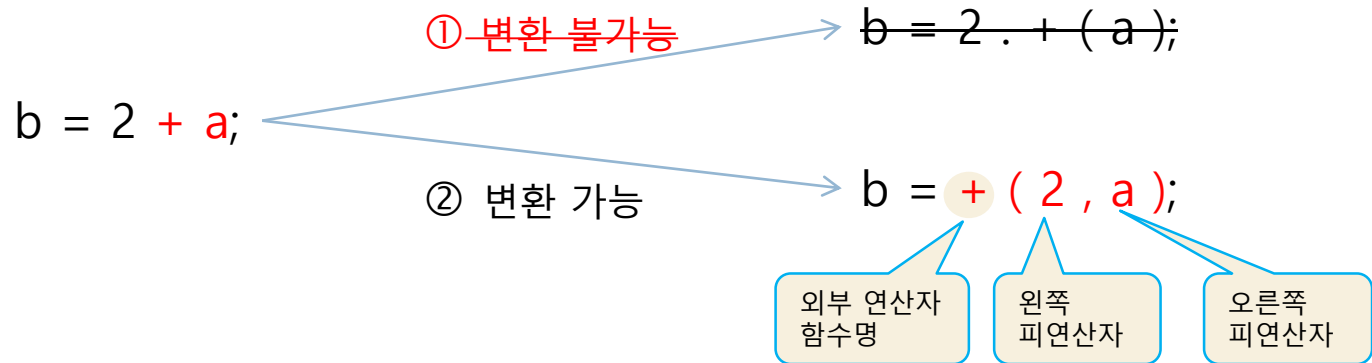
a, b 출력

b = a++ 후 a, b 출력

2 + a 덧셈을 위한 + 연산자 함수 작성

31

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

32

```
#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

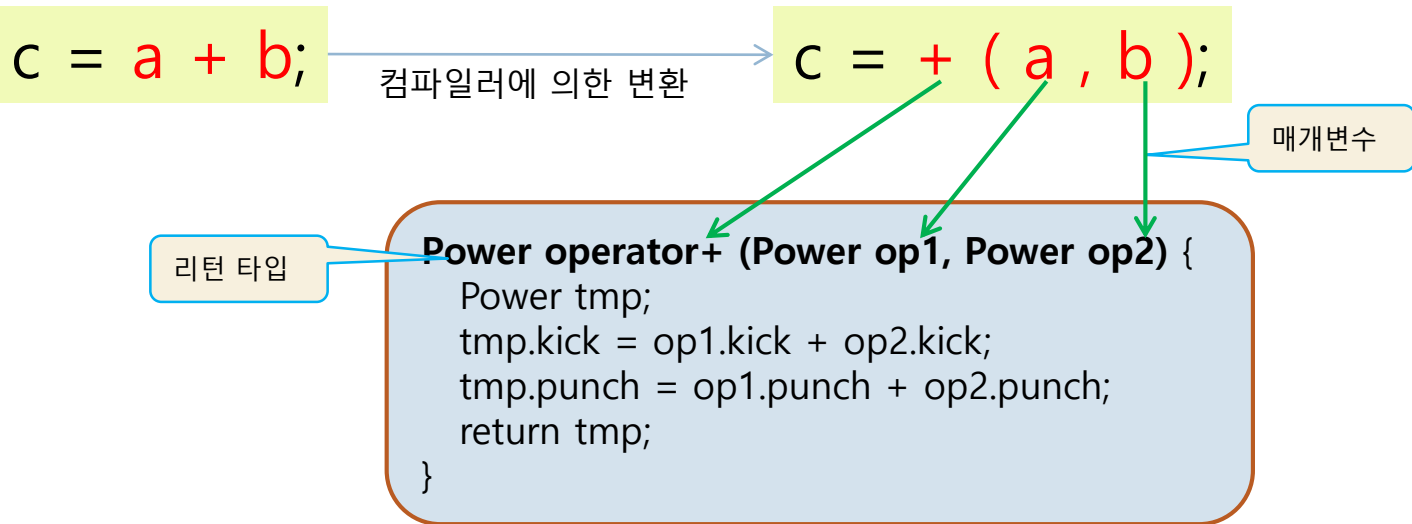
```
kick=3,punch=5
kick=0,punch=0
kick=3,punch=5
kick=5,punch=7
```

a, b 출력

b = 2+a 후 a, b 출력

+ 연산자를 외부 프렌드 함수로 구현

33



예제 7-12 a+b를 위한 연산자 함수를 프렌드로 작성

34

```
#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

35

(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 구분하기 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

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

참조 매개 변수 사
용에 주목

참조 매개 변수 사
용에 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();
}
```

```
#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

37

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();
    Power& operator << (int n); // 연산 후 Power 객체의 참조 리턴
};

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

Power& Power::operator <<(int n) {
    kick += n;
    punch += n;
    return *this; // 이 객체의 참조 리턴
}
```

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