7장 프렌드와 연산자 중복 정리노트

송승준, 정지원

- 프렌드 함수 - 클래스의 멤버 함수가 아닌 외부함수
- 전역 함수 또는 다른 클래스의 멤버 함수라는 조건을 가짐
- friend키워드로 선언
- 필요성 - 클래스의 멤버로 선언하기에는 무리가 있고 클래스의 모든 멤버를 접근할 수 있는 외부함수가 있을 때.

클래스의 멤버로 선언하기에 어떤 무리가 있는지?
A) 함수가 행하는 주 역할이 다른 클래스에서 주로 여겨질 경우 혹은 한 클래스의 멤버로 한정될 수 없는 전역 함수 등을 사용할 때 무리가 됨.

- 연산자 중복 - 본래부터 있던 연산자에 새로운 의미를 정의 -> 높은 가독성
- ex) 정수 더하기, 문자열 합치기, 색 섞기, 배열 합치기
- 정수에 국한되어있던 피연산자의 범위를 객체를 통해 넓은 의미로 다양한 타입 사용
- 반드시 클래스와 관계를 가지며 피연산자의 개수를 바꿀 수 없음, 또한 연산의 우선순위 변경 안됨
- 방법 - 1) 클래스의 멤버 함수로 구현
  2) 외부 함수로 구현 후 클래스에 프렌드 함수로 선언

type) 리턴타입 operator연산자(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
            ...
}
```

다음과 같이 코딩을 할 시, 방법 1을 사용하면

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

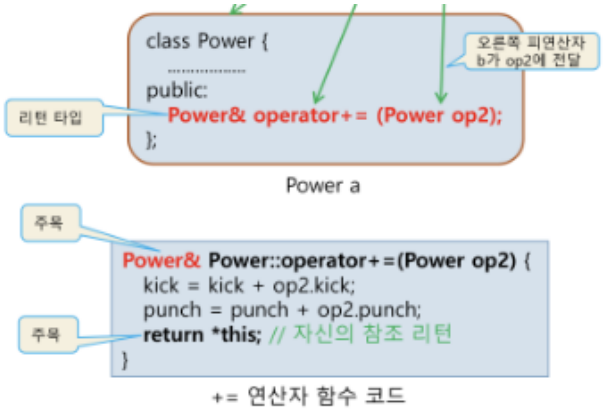
클래스의 멤버 함수로 작성된다.
다만 연산자 함수의 코드는 내부에 있지 않을 수 있다.

방법2를 사용하면,

```
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);
};
```

외부 함수로 각각 구현한 후, 클래스 내에 프렌드로 선언된다.



다음과 같이 += 연산자 중복이 일어날 경우, 함수를 선언하는 부분에서 Power&는 참조값을 의미하며 포인터 값이 아닌 *this를 리턴하는 경우에 객체가 복제되어 원하는 값이 반환되지 않는 것을 방지한다.

- 단항 연산자 - 피연산자가 하나 뿐인 연산자
- 종류 - 전위 연산자 (!, ~, ++op, --op) , 후위 연산자 (op++, op--)

다음은 전위 ++연산자 중복에 대한 예시

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

참조와 *this를 이용하여 각각 kick과 punch가 증가된 후 본인 스스로를 리턴한다

후위 ++ 연산자 중복에 대한 예시

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

tmp라는 객체를 생성하여 이전 상태를 저장한 후 kick과 punch가 증가된 후 증가 이

전의 객체 tmp를 반환한다

- 2 + a 덧셈을 위한 + 연산자 함수 작성시, b = 2 + a;를 컴파일러는

b = 2. + (a);로 변환할 수 없어 b = +(2 , a);와 같이 변환하여 a와 2 모두 피연산자로 취급합니다.

- 왜 첫번째 방법인 b = 2. + (a);로 변환할 수 없는 건지?

A) 보통 a. ++();를 예시로 들면 a객체의 ++함수를 호출하여 매개변수를 전달하는 방식으로 시행되는데 위에서는 2라는 객체를 생성할 수 없으므로 모두 피연산자로 취급을 할 수 밖에 없다.