

10장 연산자 정보과 프렌드 함수 (2)

2020. 11. 5

상치향대학교 컴퓨터 공학과

내용

- = (할당연산자)
- +=
- 전위 ++
- 후위 ++
- <<
- []

할당 연산자('=') 중복

- 할당연산자의 경우, 여러분이 중복하지 않으면 디폴트 버전의 할당연산자가 컴파일러에 의해서 생성
 - 이 버전은 멤버 단위 복사(shallow copy)를 수행
 - 이를 **복사 할당연산자**(copy assignment operator)라고 함
 - Deep copy 등을 수행하는 여러분의 버전으로 중복 가능함
- 다음 사항 고려:
 - 객체가 생성되면서 다른 객체로 초기화될 때
 - 생성된 객체를 다른 객체에 할당할 때

할당 연산자 중복 예제

```
class Box {
private:
    double length, width, height;
public:
    Box(double l = 0.0, double w = 0.0, double h = 0.0)
        : length{l}, width{w}, height{h} {    }
    void display() {
        cout << "(" << length << ", " << width << ", " << height << ")" << endl;
    }
    Box& operator=(const Box& b2) { // 객체 참조가 아닌 객체를 반환하면?
        this->length = b2.length;
        this->width = b2.width;
        this->height = b2.height;
        return *this;
    }
};

int main()
{
    Box b1(30.0, 30.0, 60.0), b2, b3;

    b1.display();
    b3 = b2 = b1; // what if '(b3 = b2) = b1;'
    b2.display();

    return 0;
}
```

`+=` 연산자 중복

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

컴파일러 변환

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

Power a

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

`+=` 연산자 함수

```
Power& Power::operator+=(Power op2) {  
    kick = kick + op2.kick;  
    punch = punch + op2.punch;  
    return *this;  
}
```

+= 연산자 중복 예제

```
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);  
};  
  
void Power::show() {  
    cout << "kick=" << kick << ',' << "punch=" << punch  
    << endl;  
}  
  
Power& Power::operator+=(Power op2) {  
    kick = kick + op2.kick;  
    punch = punch + op2.punch;  
    return *this;  
}
```

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

단항 연산자 전위 ++ 연산자 중복

++a

컴파일러 변환

a . ++ ()

Power a

```
class Power {  
    .....  
public:  
    Power& operator++ ( );  
};
```

매개 변수 없음

전위 ++ 연산자 함수

```
Power& Power::operator++( ) {  
    // kick과 punch는 a의 멤버  
    kick++;  
    punch++;  
  
    return *this;  
}
```

전위 ++ 연산자 중복 예제

```
class Power {  
    int kick;  
    int punch;  
public:  
    Power(int kick=0, int punch=0) {  
        this->kick = kick; this->punch = punch;  
    }  
    void show();  
    Power& operator++ ();  
};
```

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

```
Power& Power::operator++ () {  
    kick++;  
    punch++;  
    return *this;  
}
```

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


전위 ++ 연산자 중복 예제

```
class Power {  
    int kick;  
    int punch;  
public:  
    Power(int kick=0, int punch=0) {  
        this->kick = kick; this->punch = punch;  
    }  
    void show();  
    Power& operator++ ();  
};
```

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

```
Power& Power::operator++ () {  
    kick++;  
    punch++;  
    return *this;  
}
```

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

객체 참조를 반환하지 않고
객체를 반환한다면?

후위 연산자 중복, ++ 연산자

a++

컴파일러 변환

a . ++ (임의의 정수)

Power a

```
class Power {  
    .....  
public:  
    Power operator ++ (int x );  
};
```

매개 변수 필요

후위 ++ 연산자 함수

```
Power Power::operator++(int x) {  
    Power tmp = *this;  
    kick++;  
    punch++;  
    return tmp;  
}
```

후위 ++ 연산자 중복 예제

```
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);  
};
```

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

```
Power Power::operator++(int x) {  
    Power tmp = *this;  
    kick++;  
    punch++;  
    return tmp;  
}
```

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

증감 연산자 ++를 외부함수로 작성하기

(a) 전위 연산자

`++a`

컴파일러 변환

`++ (a)`

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

0은 의미 없는 값으로 전위 연산자와 구분하기 위함

(b) 후위 연산자

`a++`

컴파일러 변환

`++ (a, 0)`

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

외부함수로 ++연산자 중복 예

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

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

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

```
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);  
    friend Power operator++(Power& op, int x);  
};  
  
void Power::show() {  
    cout << "kick=" << kick << ',' << "punch=" << punch << endl;  
}
```

<< 연산자 중복

```
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);  
};
```

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

```
Power& Power::operator <<(int n) { // 객체 참조가 아닌 객체를 반환하면?  
    kick += n;  
    punch += n;  
    return *this;  
}
```

```
int main() {  
    Power a(1, 2);  
    a << 3 << 5 << 6;  
    a.show(); // kick, punch에 3, 5, 6이  
              // 연속적으로 더해지게 중복  
}
```

인덱스 연산자 []의 중복

- 인덱스가 배열 경계를 벗어나는 오류 상황을 피하도록 중복 정의하는 것이 가능함
- 예:

MyArray A; // [] 연산자가 중복 정의됨

...

A[3] = 10; // 컴파일러 변환: A.operator[](3) = 10;

인덱스 연산자 []의 중복 예제

```
const int SIZE = 10;
```

```
class MyArray {
```

```
private:
```

```
    int a[SIZE];
```

```
public:
```

```
    MyArray() {
```

```
        for (int i = 0; i < SIZE; i++)
```

```
            a[i] = 0;
```

```
    }
```

```
int &operator[](int i) {
```

```
    if (i >= SIZE) {
```

```
        cout << "잘못된 인덱스:" ;
```

```
        return a[0];
```

```
    }
```

```
    return a[i];
```

```
}
```

```
};
```

```
int main() {
```

```
    MyArray A;
```

```
    A[3] = 9;
```

```
    cout << "A[3]= " << A[3] << endl;
```

```
    cout << "A[16]= " << A[16] << endl;
```

```
    return 0;
```

```
}
```