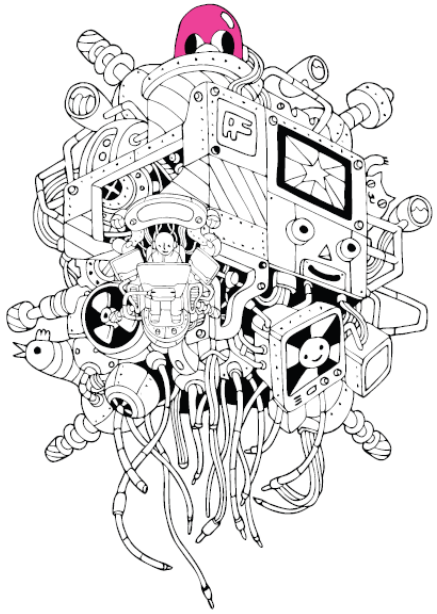
윤성우의
열혈 C++ 프로그래밍



윤성우 저 열혈강의 C++ 프로그래밍 개정판

Chapter 11. 연산자 오버로딩2

윤성우의

열혈 C++ 프로그래밍

Chapter 11-1. 반드시 해야 하는 대입 연산자의 오버로딩

윤성우 저 열혈강의 C++ 프로그래밍 개정판

객체간 대입연산의 비밀: 디폴트 대입 연산자

복사 생성자의 호출

```
int main(void)
{
    Point pos1(5, 7);
    Point pos2=pos1;
    . . . .
}
```

대입 연산자의 호출

```
int main(void)
{
    Point pos1(5, 7);
    Point pos2(9, 10);
    pos2=pos1;
    . . . .
}
```

pos2.operator=(pos1);

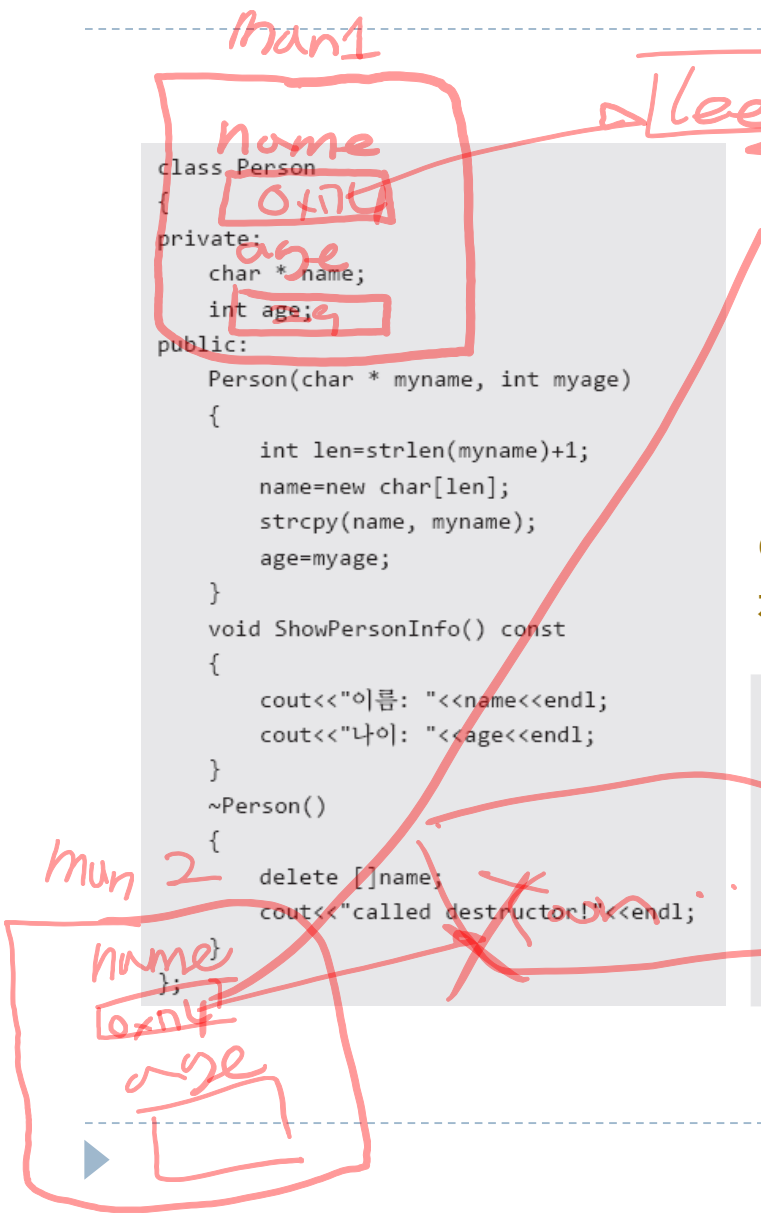
```
class First
{
private:
    int num1, num2;
public:
    First(int n1=0, int n2=0) : num1(n1), num2(n2)
    { }
    void ShowData() { cout<<num1<<", "<<num2<<endl; }
};
```

멤버 대 멤버의 복사를 진행하는 디폴트 대입 연산자 삽입!

```
First& operator=(const First& ref)
{
    num1=ref.num1;
    num2=ref.num2;
    return *this;
}
```

Frist 클래스의 디폴트 대입 연산자

디폴트 대입 연산자의 문제점




```
class Person
{
private:
    char * name;
    int age;
public:
    Person(char * myname, int myage)
    {
        int len=strlen(myname)+1;
        name=new char[len];
        strcpy(name, myname);
        age=myage;
    }
    void ShowPersonInfo() const
    {
        cout<<"이름: "<<name<<endl;
        cout<<"나이: "<<age<<endl;
    }
    ~Person()
    {
        delete []name;
        cout<<"called destructor!"<<endl;
    }
};
```

해결책이 되는 대입 연산자의 오버라이딩

```
Person& operator=(const Person& ref)
{
    delete []name;    // 메모리의 누수를 막기 위한 메모리 해제 연산
    int len=strlen(ref.name)+1;
    name= new char[len];
    strcpy(name, ref.name);
    age=ref.age;
    return *this;
}
```

이전에 공부한 디폴트 복사 생성자의 문제점과 동일한 문제점을 확인할 수 있다!

```
int main(void)
{
    Person man1("Lee dong woo", 29);
    Person man2("Yoon ji yul", 22);
    man2=man1;
    man1.ShowPersonInfo();
    man2.ShowPersonInfo();
    return 0;
}
```

실행결과

```
이름: Lee dong woo
나이: 29
이름: Lee dong woo
나이: 29
called destructor!
```

상속 구조에서의 대입 연산자 호출


```
class First
{
private:
    int num1, num2;
public:
    First(int n1=0, int n2=0) : num1(n1), num2(n2)
    { }
    void ShowData() { cout<<num1<<", "<<num2<<endl; }

    First& operator=(const First& ref)
    {
        cout<<"First& operator=()"<<endl;
        num1=ref.num1;
        num2=ref.num2;
        return *this;
    }
};
```

```
class Second : public First
{
private:
    int num3, num4;
public:
    Second(int n1, int n2, int n3, int n4)
        : First(n1, n2), num3(n3), num4(n4)
    { }
    void ShowData()
    {
        First::ShowData();
        cout<<num3<<", "<<num4<<endl;
    }

    /*
    Second& operator=(const Second& ref)
    {
        cout<<"Second& operator=()"<<endl;
        num3=ref.num3;
        num4=ref.num4;
        return *this;
    }
    */
};
```

```
Second& operator=(const Second& ref)
{
    cout<<"Second& operator=()"<<endl;
    First::operator=(ref);
    num3=ref.num3;
    num4=ref.num4;
    return *this;
}
```

디폴트 대입 연산자는 기초 클래스의 대입연산자를 호출해준다. 그러나 명시적으로 대입 연산자를 정의하게 되면, 기초 클래스의 대입 연산자 호출도 오른쪽의 예와 같이 별도로 명시해야 한다.

이니셜라이저의 성능 향상 도움

```
class AAA
{
private:
    int num;
public:
    AAA(int n=0): num(n)
    {
        cout<<"AAA(int n=0)"<<endl;
    }
    AAA(const AAA &ref): num(ref.num)
    {
        cout<<"AAA(const AAA & ref)"<<endl;
    }
    AAA & operator=(const AAA &ref)
    {
      num=ref.num;
      cout<<"operator=(const AAA &ref)"<<endl;
      return *this;
    }
};
```

```
class BBB
{
private:
    AAA mem;
public:
    BBB(const AAA& ref) : mem(ref) { }
};

class CCC
{
private:
    AAA mem;
public:
    CCC(const AAA& ref) { mem=ref; }
};

int main(void)
{
    AAA obj1(12);
    cout<<"**********"<<endl;
    BBB obj2(obj1);
    cout<<"**********"<<endl;
    CCC obj3(obj1);
    return 0;
}
```

함수2회

함수2회

이니셜라이저를 이용해서 멤버를 초기화하면, 함수호출의 수를 1회 줄일 수 있다!

실행결과

```
AAA(int n=0)
**********
AAA(const AAA& ref)
**********
AAA(int n=0)
operator=(const AAA& ref)
```