복사 생성자

박 종 혁 교수

UCS Lab

Tel: 970-6702

Email: jhpark1@seoultech.ac.kr

SeoulTech

'복사 생성자'와의 첫 만남

복사 생성자란?

- 복사 생성자
  - 선언되는 객체와 같은 자료형의 객체를 인수로 전달하는 생성자
  - 인수는 참조자(& 선언)로 전달
    - 참조자가 아니면 무한루프에 빠짐
  - 전달되는 인수는 대개 const 선언
- 디폴트 복사 생성자
  - 복사 생성자 정의 생략 시 자동으로 삽입되는 복사 생성자
  - 인수로 전달되는 객체의 멤버변수를 선언되는 객체의 멤버변수로 복사

C++ 스타일의 초기화

C 스타일 초기화

```
int num=20;
int &ref=num;
```

C++ 스타일 초기화

```
int num(20);
int &ref(num);
```

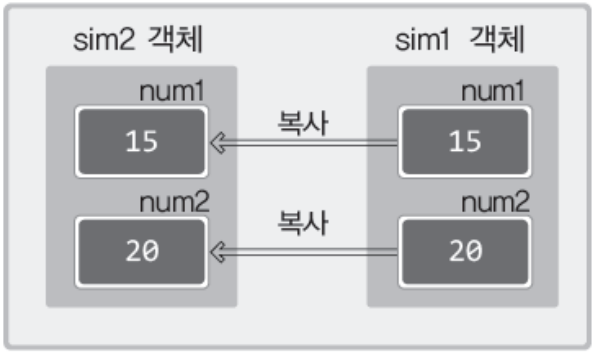
이렇듯, 다음 두 문장은 실제로 동일한 문장으로 해석된다.

SoSimple sim2=sim1;

SoSimple sim2(sim1);

```
class SoSimple
{
private:
    int num1;
    int num2;
public:
    SoSimple(int n1, int n2) : num1(n1), num2(n2)
    { }
    void ShowSimpleData()
    {
        cout<<num1<<endl;
        cout<<num2<<endl;
    }
};
```

```
int main(void)
{
    SoSimple sim1(15, 20);
    SoSimple sim2=sim1;
    sim2.ShowSimpleData();
    return 0;
}
```



대입연산의 의미처럼 실제 멤버 대 멤버의 복사가 일어난다!

SoSimple sim2(sim1);

SoSimple sim2(sim1)의 해석!

- SoSimple형 객체를 생성해라.
- 객체의 이름은 sim2로 정한다.
- sim1을 인자로 받을 수 있는 생성자의 호출을 통해서 객체생성을 완료한다.

```
SoSimple(SoSimple &copy)
{
    . . . .
}
```

SoSimple sim2=sim1 은 묵시적으로 SoSimple sim2(sim1) 으로 해석이 된다.

```
class SoSimple
{
    . . . .
public:
    SoSimple(int n1, int n2)
        : num1(n1), num2(n2)
    {
        // empty
    }
    SoSimple(SoSimple &copy)
        : num1(copy.num1), num2(copy.num2)
    {
        cout<<"Called SoSimple(SoSimple &copy)"<<endl;
    }
    . . . .
};
```

```
int main(void)
{
    SoSimple sim1(15, 30);
    cout<<"생성 및 초기화 직전"<<endl;
    SoSimple sim2=sim1;     // SoSimple sim2(sim1); 으로 변환!
    cout<<"생성 및 초기화 직후"<<endl;
    sim2.ShowSimpleData();
    return 0;
}
```

실행결과

```
생성 및 초기화 직전
Called SoSimple(SoSimple &copy)
생성 및 초기화 직후
15
30
```

classInit.cpp

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

class SoSimple
{
private:
	int num1;
	int num2;
public:
	SoSimple(int n1, int n2)
		: num1(n1), num2(n2)
	{
		// empty
	}
	SoSimple(SoSimple &copy)
		: num1(copy.num1), num2(copy.num2)
	{
		cout<<"Called SoSimple(SoSimple &copy)"<<endl;
	}
	void ShowSimpleData()
	{
		cout<<num1<<endl;
		cout<<num2<<endl;
	}
};

int main(void)
{
	SoSimple sim1(15, 30);
	cout<<"생성 및 초기화 직전"<<endl;
	SoSimple sim2=sim1;
	cout<<"생성 및 초기화 직후"<<endl;
	sim2.ShowSimpleData();
	return 0;
}
```

자동으로 삽입이 되는 디폴트 복사 생성자

```
class SoSimple
{
private:
    int num1;
    int num2;
public:
    SoSimple(int n1, int n2) : num1(n1), num2(n2)
    { }
    . . . .
};
```

```
class SoSimple
{
private:
    int num1;
    int num2;
public:
    SoSimple(int n1, int n2) : num1(n1), num2(n2)
    { }
    SoSimple(const SoSimple &copy) : num1(copy.num1), num2(copy.num2)
    { }
};
```

복사 생성자를 정의하지 않으면, 멤버 대 멤버의 복사를 진행하는 디폴트 복사 생성자가 삽입된다.

키워드 explicit

```
SoSimple sim2=sim1;   ➡   SoSimple sim2(sim1);
```

이러한 묵시적 형 변환은 복사 생성자를 explicit 으로 선언하면 막을 수 있다.

```
explicit SoSimple(const SoSimple &copy)
            : num1(copy.num1), num2(copy.num2)
{
    // empty!!
}
```

```
class AAA
{
private:
    int num;
public:
    AAA(int n) : num(n) { }
    . . . . .
};
```

AAA 생성자를 explicit로 선언하면 AAA obj=3 과 같은 형태로 객체 생성 불가!

‘깊은 복사’와 ‘얕은 복사’

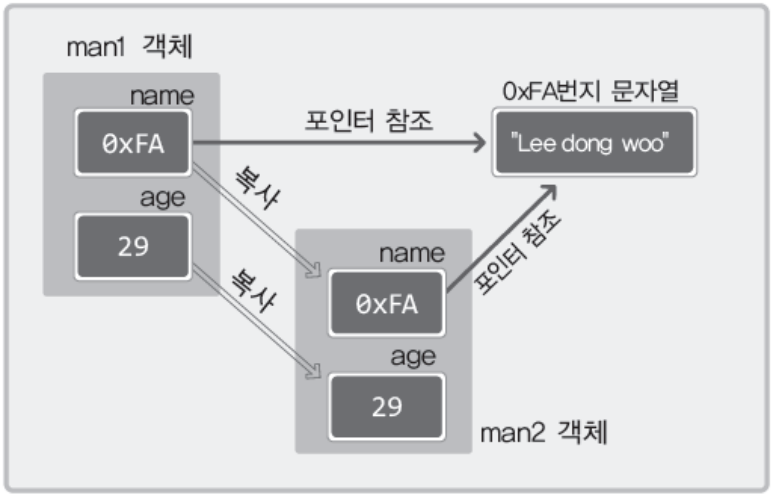
디폴트 복사 생성자의 문제점

```
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;
    }
    . . . .
    ~Person()
    {
        delete []name;
        cout<<"called destructor!"<<endl;
    }
};
```

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

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

실행결과



객체 소멸 시 문제가 되는 구조!!! 얕은 복사!

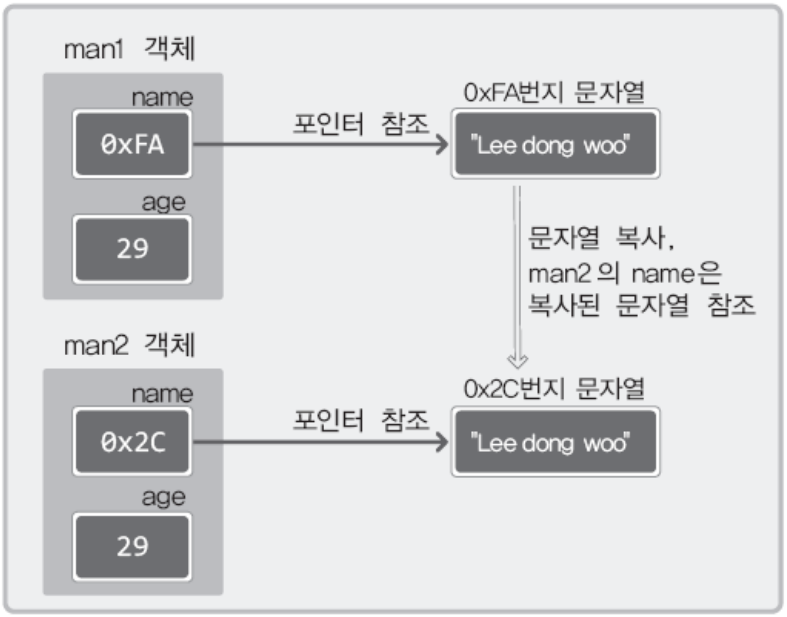
ShallowCopyError.cpp

```
#include <iostream>
#include <cstring>
using namespace std;
class Person
{
    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;
    }
};
int main(void)
{
    Person man1("Lee dong woo", 29);
    Person man2=man1;
    man1.ShowPersonInfo();
    man2.ShowPersonInfo();
    return 0;
}
```

‘깊은 복사’를 위한 복사 생성자의 정의

```
Person(const Person& copy) : age(copy.age)
{
     name=new char[strlen(copy.name)+1];
     strcpy(name, copy.name);
}
```

깊은 복사를 구성하는 복사 생성자!!!



복사 생성자의 호출시점

복사 생성자가 호출되는 시점

case 1: 기존에 생성된 객체를 이용해서 새로운 객체를 초기화하는 경우(앞서 보인 경우)

case 2: Call-by-value 방식의 함수호출 과정에서 객체를 인자로 전달하는 경우

case 3: 객체를 반환하되, 참조형으로 반환하지 않는 경우

메모리 공간의 할당과 초기화가 동시에 일어나는 상황

case 1

```
Person man1("Lee dong woo", 29);
Person man2=man1;      // 복사 생성자 호출
```

case 2 & case 3

```
SoSimple SimpleFuncObj(SoSimple ob)
{
    . . . . .
    return ob;
}
int main(void)
{
    SoSimple obj;
    SimpleFuncObj(obj);
    . . . .
}
```

인자 전달 시 선언과 동시에 초기화

```
int SimpleFunc(int n)
{
    . . . . .
    return n;
}
int main(void)
{
    int num=10;
    cout<<SimpleFunc(num)<<endl;
    . . . .
}
```

반환 시 메모리 공간 할당과 동시에 초기화

복사 생성자의 호출 case의 확인 (1)

```
class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n)
    { }
    SoSimple(const SoSimple& copy) : num(copy.num)
    {
        cout<<"Called SoSimple(const SoSimple& copy)"<<endl;
    }
    void ShowData()
    {
        cout<<"num: "<<num<<endl;
    }
};

void SimpleFuncObj(SoSimple ob)
{
    ob.ShowData();
}
```

```
int main(void)
{
    SoSimple obj(7);
    cout<<"함수호출 전"<<endl;
    SimpleFuncObj(obj);
    cout<<"함수호출 후"<<endl;
    return 0;
}
```

```
함수호출 전
Called SoSimple(const SoSimple& copy)
num: 7
함수호출 후
```

실행결과

PassObjCopycon.cpp

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

class SoSimple
{
private:
	int num;
public:
	SoSimple(int n) : num(n)
	{ }
	SoSimple(const SoSimple& copy) : num(copy.num)
	{
		cout<<"Called SoSimple(const SoSimple&
	copy)"<<endl;
	}
	void ShowData()
	{
		cout<<"num: "<<num<<endl;
	}
};

void SimpleFuncObj(SoSimple ob)
{
		ob.ShowData();
}

int main(void)
{
		SoSimple obj(7);
		cout<<"함수호출 전"<<endl;
		SimpleFuncObj(obj);
		cout<<"함수호출 후"<<endl;
		return 0;
}
```

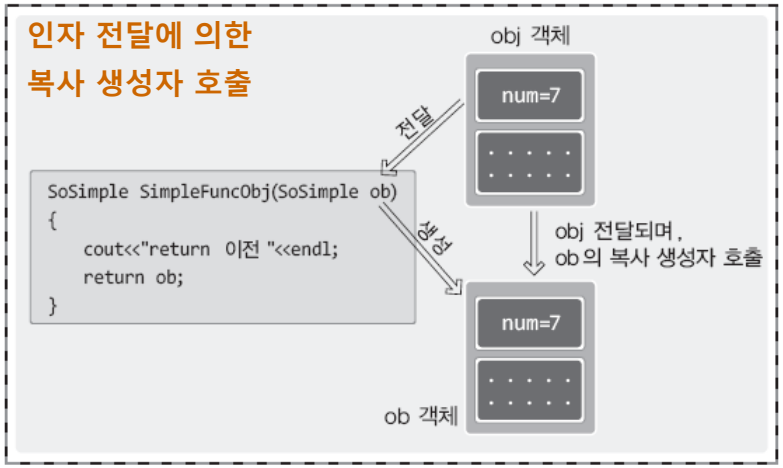
복사 생성자의 호출 case의 확인 (2)

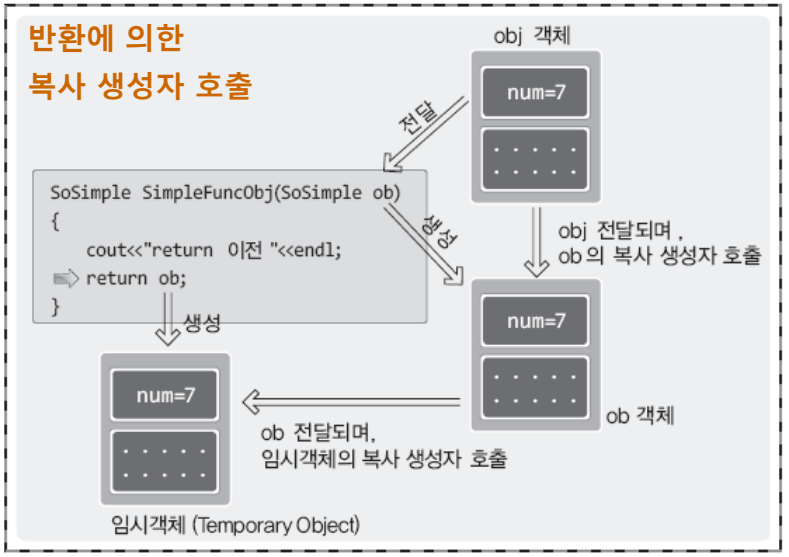
```
class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n)
    { }
    SoSimple(const SoSimple& copy) : num(copy.num)
    {
        cout<<"Called SoSimple(const SoSimple& copy)"<<endl;
    }
    SoSimple& AddNum(int n)
    {
        num+=n;
        return *this;
    }
    void ShowData()
    {
        cout<<"num: "<<num<<endl;
    }
};

SoSimple SimpleFuncObj(SoSimple ob)
{
    cout<<"return 이전"<<endl;
    return ob;
}
```

```
int main(void)
{
    SoSimple obj(7);
    SimpleFuncObj(obj).AddNum(30).ShowData();
    obj.ShowData();
    return 0;
}
```

```
Called SoSimple(const SoSimple& copy)
return 이전
Called SoSimple(const SoSimple& copy)
num: 37
num: 7
```

실행결과





ReturnObjCopycon.cpp

```
#include <iostream>
using namespace std;
class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n)
    { }
    SoSimple(const SoSimple& copy) : num(copy.num)
    {
        cout<<"Called SoSimple(const SoSimple&
    copy)"<<endl;
    }
    SoSimple& AddNum(int n)
    {
        num+=n;
        return *this;
    }
void ShowData()
        {
            cout<<"num: "<<num<<endl;
        }
};

SoSimple SimpleFuncObj(SoSimple ob)
{
        cout<<"return 이전"<<endl;
        return ob;
}

int main(void)
{
        SoSimple obj(7);
        SimpleFuncObj(obj).AddNum(30).ShowData();
        obj.ShowData();
        return 0;
}
```

반환할 때 만들어진 객체의 소멸 시점

```
class Temporary
{
private:
    int num;
public:
    Temporary(int n) : num(n)
    {
        cout<<"create obj: "<<num<<endl;
    }
    ~Temporary()
    {
        cout<<"destroy obj: "<<num<<endl;
    }
    void ShowTempInfo()
    {
        cout<<"My num is "<<num<<endl;
    }
};
```

```
int main(void)
{
    Temporary(100);
    cout<<"********** after make!"<<endl<<endl;

    Temporary(200).ShowTempInfo();
    cout<<"********** after make!"<<endl<<endl;

    const Temporary &ref=Temporary(300);
    cout<<"********** end of main!"<<endl<<endl;
    return 0;
}
```

참조값이 반환되므로 참조자로 참조 가능!

```
create obj: 100
destroy obj: 100
********** after make!

create obj: 200
My num is 200
destroy obj: 200
********** after make!

create obj: 300
********** end of main!

destroy obj: 300
```

실행결과

클래스 외부에서 객체의 멤버함수를 호출하기 위해 필요한 조건

- 객체에 붙여진 이름
- 객체의 참조값(객체 참조에 사용되는 정보)
- 객체의 주소 값

```
Temporary(200).ShowTempInfo(); → (임시객체의 참조 값).ShowTempInfo();
```

임시객체는 다음 행으로 넘어가면 바로 소멸

참조자에 참조되는 임시객체는 바로 소멸되지 않는다.

IKnowTempObj.cpp

```
#include <iostream>
using namespace std;
class Temporary
{
private:
    int num;
public:
    Temporary(int n) : num(n)
    {
        cout<<"create obj: "<<num<<endl;
    }
    ~Temporary()
    {
        cout<<"destroy obj: "<<num<<endl;
    }
    void ShowTempInfo()
    {
        cout<<"My num is "<<num<<endl;
    }
};
```

```
int main(void)
{
        Temporary(100);
        cout<<"********** after make!"<<endl<<endl;

        Temporary(200).ShowTempInfo();
        cout<<"********** after make!"<<endl<<endl;

        const Temporary &ref=Temporary(300);
        cout<<"********** end of main!"<<endl<<endl;
        return 0;
}
```

Q&A