4장. 복사 생성자

박 종 혁 교수
(서울과기대 컴퓨터공학과)

Tel: 970-6702
Email: jhpark1@seoultech.ac.kr

목차

1. 복사 생성자 개요
2. “얕은 복사”와 “깊은 복사”
3. 복사 생성자의 호출시점

1. 복사 생성자 개요

복사 생성자란?

- 복사 생성자 (copy constructor)
  - 선언되는 객체와 같은 자료형의 객체를 매개변수로 전달하는 생성자
  - 매개변수는 참조자(& 선언)로 전달
    - 참조자가 아니면 무한루프에 빠짐
  - 전달되는 매개변수는 대개 const 선언

복사 생성자

- 특징
  - 한 클래스에 오직 한 개만 선언 가능
  - 복사 생성자는 보통 생성자와 클래스 내에 중복 선언 가능
  - 모양
    - 클래스에 대한 참조 매개 변수를 가지는 독특한 생성자

- 복사 생성자 선언

```
class Circle {
    ..........
    Circle(const Circle& c); // 복사 생성자 선언
    ..........
};

Circle::Circle(c          ){ // 복사 생성자
구현
    ..........
}
```

자기 클래스에 대한 참조 매개 변수

C++ 스타일의 초기화

C 스타일 초기화

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

C++ 스타일 초기화

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

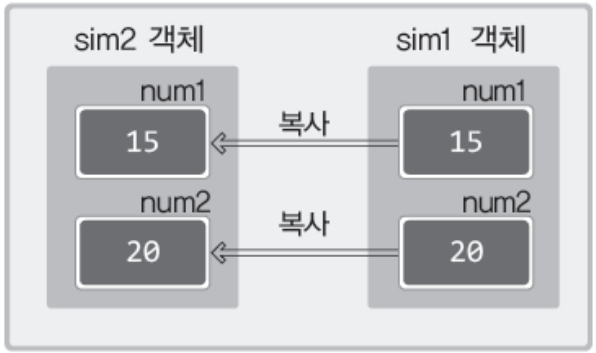
이렇듯, 다음 두 문장은 실제로 동일한 문장으로 해석된다.

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;
}
```

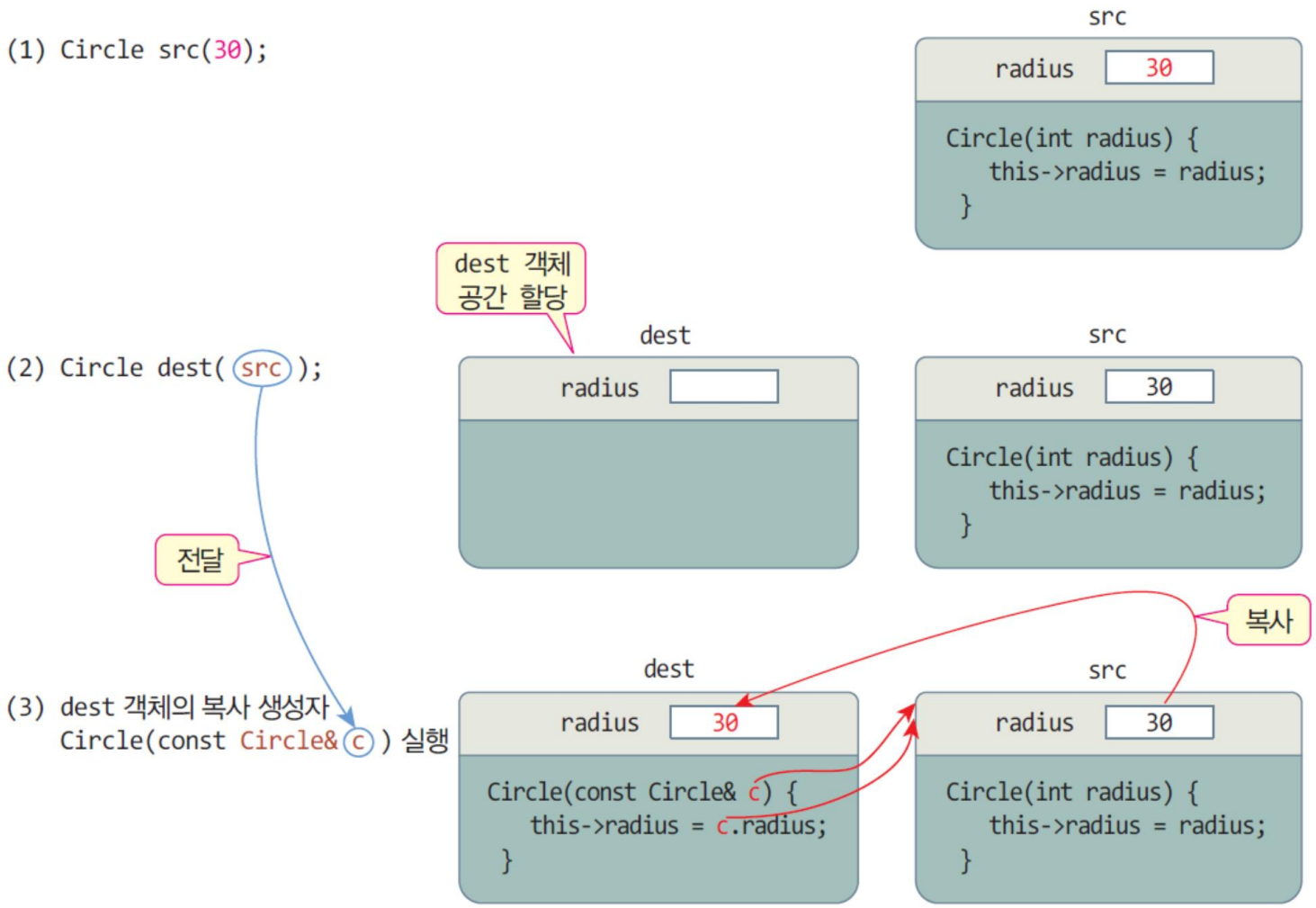
예제- Circle의 복사 생성자와 객체 복사

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

class Circle {
private:
    int radius;
public:
    Circle(const Circle& c); // 복사 생성자 선언
    Circle() { radius = 1; }
    Circle(int radius) { this->radius = radius; }
    double getArea() { return 3.14*radius*radius; }
};

Circle::Circle(const Circle& c) { // 복사 생성자 구현
    this->radius = c.radius;
    cout << "복사 생성자 실행 radius = " << radius << endl;
}

int main() {
    Circle src(30); // src 객체의  보통 생성자 호출
    Circle dest(src); // dest 객체의 복사 생성자 호출

    cout << "원본의 면적 = " << src.getArea() << endl;
    cout << "사본의 면적 = " << dest.getArea() << endl;
}
```

dest 객체가 생성될 때 Circle(const Circle& c)

```
복사 생성자 실행 radius = 30
원본의 면적 = 2826
사본의 면적 = 2826
```

복사 생성 과정


(1) Circle src(30);
src
radius 30
Circle(int radius) {
this->radius = radius;
}
dest 객체
공간 할당
dest
radius
(2) Circle dest(src);
src
radius 30
Circle(int radius) {
this->radius = radius;
}
전달
복사
dest
radius 30
Circle(const Circle& c) {
this->radius = c.radius;
}
(3) dest 객체의 복사 생성자
Circle(const Circle& c) 실행
src
radius 30
Circle(int radius) {
this->radius = radius;
}

디폴트 복사 생성자

- 복사 생성자 정의 생략 시 자동으로 삽입되는 복사 생성자
- 매개변수로 전달되는 객체의 멤버변수를 선언되는 객체의 멤버변수로 복사

디폴트 복사 생성자

- 복사 생성자가 선언되어 있지 않는 클래스
  - 컴파일러는 자동으로 디폴트 복사 생성자 삽입

```
class Circle {
    int radius;
pubic:
    Circle(int r);
    double getArea();
};
```

복사 생성자 없음

복사 생성자 없는데 컴파일 오류?

```
Circle dest(src); // 복사 생성. Circle(const Circle&) 호출
```

```
Circle::Circle(const Circle& c) {
    this->radius = c.radius;
    // 원본 객체 c의 각 멤버를 사본(this)에 복사한다.
}
```

디폴트 복사 생성자

디폴트 복사 생성자 사례

```
class Book {
    double price; // 가격
    int pages;          // 페이지수
    char *title;        // 제목
    char *author;       // 저자이름
public:
    Book(double pr, int pa, char* t, char* a;);
    ~Book()
};
```

복사 생성자가 없는 Book 클래스

컴파일러가 삽입하는
디폴트 복사 생성자

```
Book(const Book& book) {
    this->price = book.price;
    this->pages = book.pages;
    this->title = book.title;
    this->author = book.author;
}
```

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.

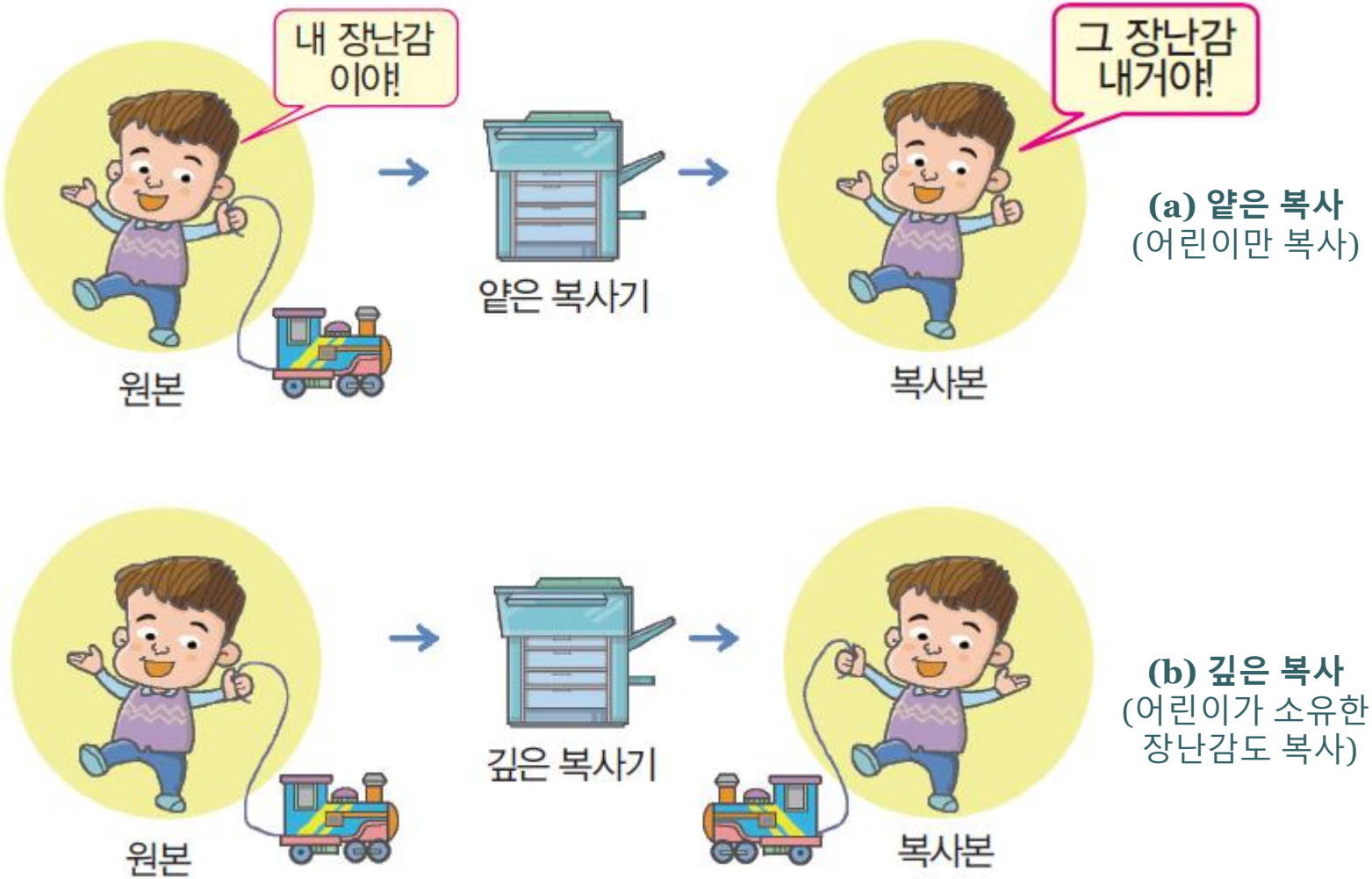
```
explicit SoSimple(const SoSimple &copy)
            : num1(copy.num1), num2(copy.num2)
{
    // empty!!
}
```

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

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

2. ‘깊은 복사’와 ‘얕은 복사’

얕은 복사와 깊은 복사

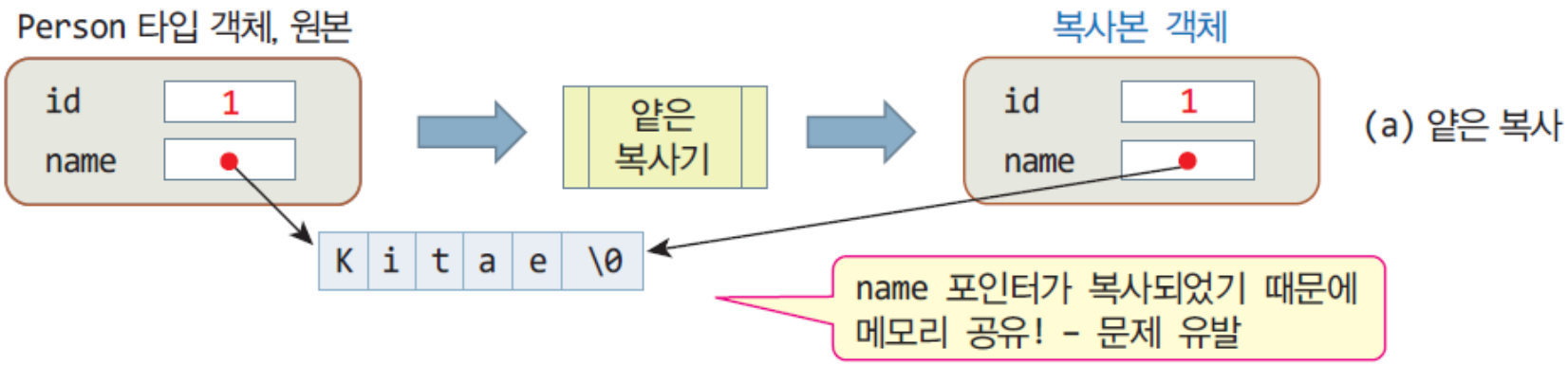

내 장난감 이야!
얕은 복사기
그 장난감 내거야!
원본
복사본
(a) 얕은 복사
(어린이만 복사)
깊은 복사기
원본
복사본
(b) 깊은 복사
(어린이가 소유한 장난감도 복사)

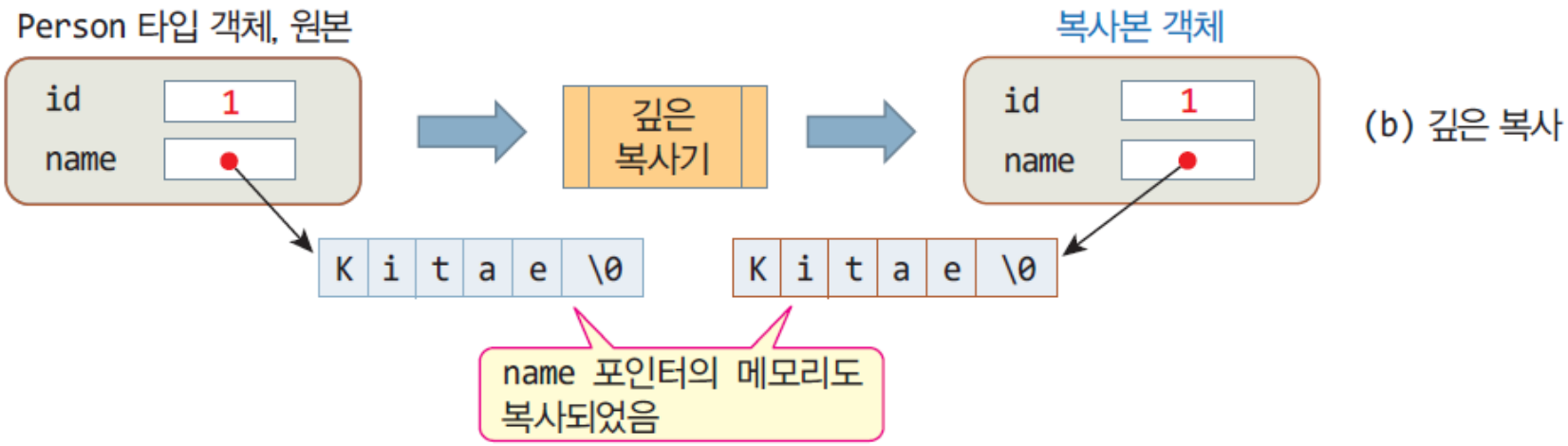
C++에서 얕은 복사와 깊은 복사

- 얕은 복사(shallow copy)
  - 객체 복사 시, 객체의 멤버를 1:1로 복사
  - 객체의 멤버 변수에 동적 메모리가 할당된 경우
    - 사본은 원본 객체가 할당 받은 메모리를 공유하는 문제 발생
- 깊은 복사(deep copy)
  - 객체 복사 시, 객체의 멤버를 1:1대로 복사
  - 객체의 멤버 변수에 동적 메모리가 할당된 경우
    - 사본은 원본이 가진 메모리 크기 만큼 별도로 동적 할당
    - 원본의 동적 메모리에 있는 내용을 사본에 복사
  - 완전한 형태의 복사
    - 사본과 원본은 메모리를 공유하는 문제 없음

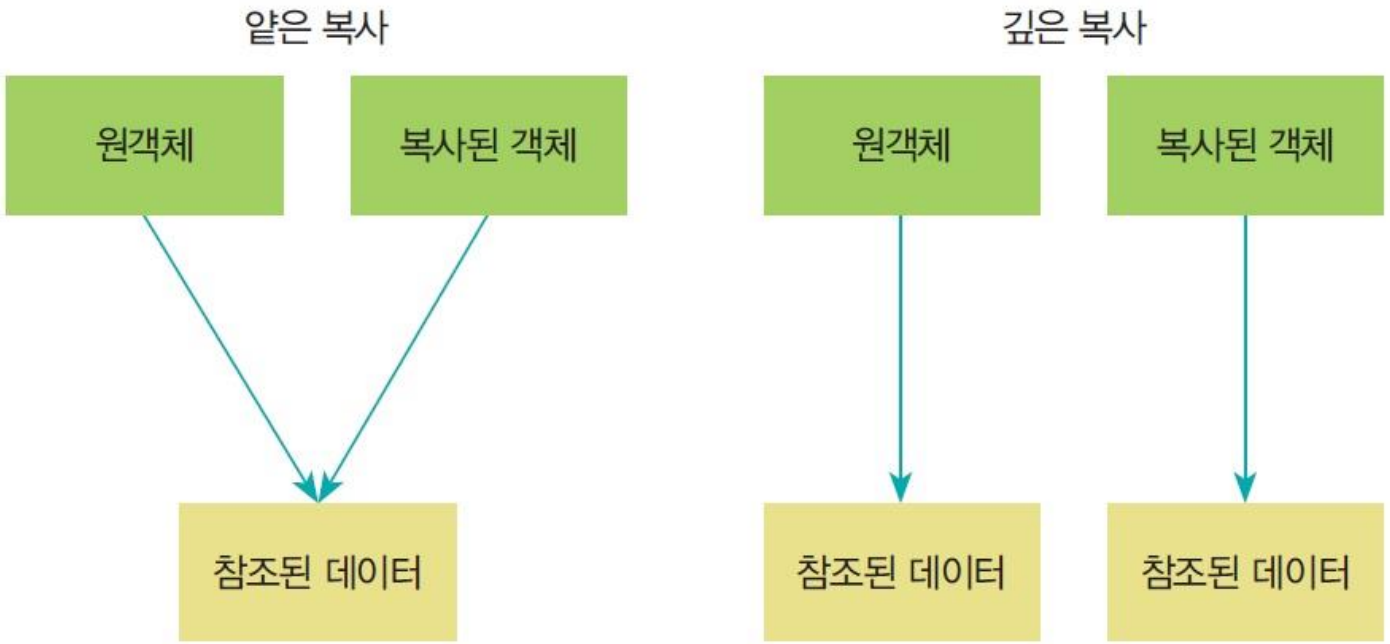
C++에서 객체의 복사

```
class Person {
    int id;
    char *name;
    ..................
};
```





깊은 복사와 얕은 복사



예제- 얕은 복사 생성자를 사용하여 프로그램이 비정상 종료되는 경우

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

class Person { // Person 클래스 선언
    char* name;
    int id;
public:
    Person(int id, const char* name); // 생성자
    ~Person(); // 소멸자
    void changeName(const char *name);
    void show() { cout << id << ',' << name << endl; }
};

Person::Person(int id, const char* name) { // 생성자
    this->id = id;
    int len = strlen(name); // name의 문자 개수
    this->name = new char [len+1]; // name 문자열 공간 할당
    strcpy(this->name, name); // name에 문자열 복사
}

Person::~Person() {// 소멸자
    if(name) // 만일 name에 동적 할당된 배열이 있으면
        delete [] name; // 동적 할당 메모리 소멸
}

void Person::changeName(const char* name) { // 이름 변경
    if(strlen(name) > strlen(this->name))
        return;
    strcpy(this->name, name);
}
```

컴파일러에 의해
디폴트 복사 생성자 삽입

```
Person::Person(const Person& p) {
    this->id = p.id;
    this->name = p.name;
}
```

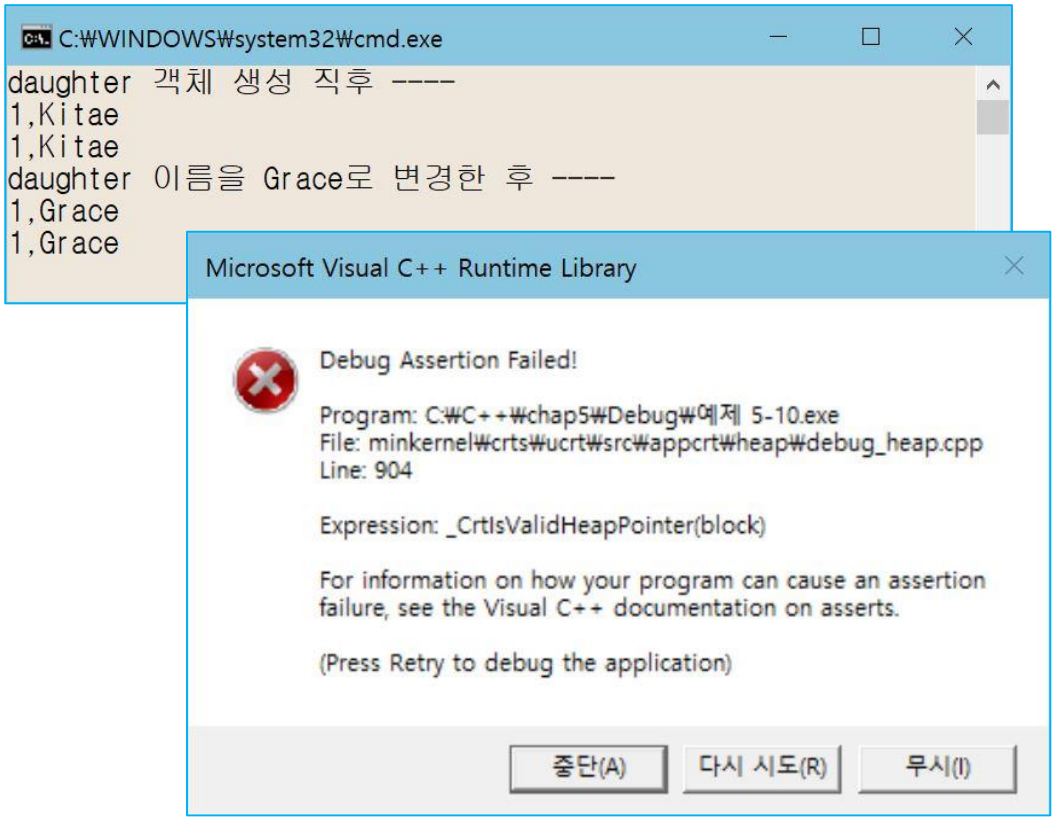
name 메모리 반환

```
int main() {
    Person father(1, "Kitae");          // (1) father 객체 생성
    Person daughter(father);            // (2) daughter 객체 복사 생성. 복사생성자호출

    cout << "daughter 객체 생성 직후 ----" << endl;
    father.show();                      // (3) father 객체 출력
    daughter.show();                    // (3) daughter 객체 출력

    daughter.changeName("Grace"); // (4) daughter의 이름을 "Grace"로 변경
    cout << "daughter 이름을 Grace로 변경한 후 ----" << endl;
    father.show();                      // (5) father 객체 출력
    daughter.show();                    // (5) daughter 객체 출력

    return 0;                           // (6), (7) daughter, father 객체 소멸
}
```

컴파일러가 삽입한
디폴트 복사 생성자 호출

daughter, father 순으로 소멸.
father가 소멸할 때, 프로그램
비정상 종료됨

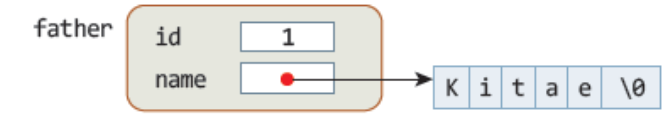
예제 실행 결과


C:₩WINDOWS₩system32₩cmd.exe
daughter 객체 생성 직후 ----
1,Kitae
1,Kitae
daughter 이름을 Grace로 변경한 후 ----
1,Grace
1,Grace
Microsoft Visual C++ Runtime Library
Debug Assertion Failed!
Program: C:₩C++₩chap5₩Debug₩예제 5-10.exe
File: minkernel₩crts₩ucrt₩src₩appcrt₩heap₩debug_heap.cpp
Line: 904
Expression: _CrtIsValidHeapPointer(block)
For information on how your program can cause an assertion failure, see the Visual C++ documentation on asserts.
(Press Retry to debug the application)
중단(A)
다시 시도(R)
무시(I)

실행 과정

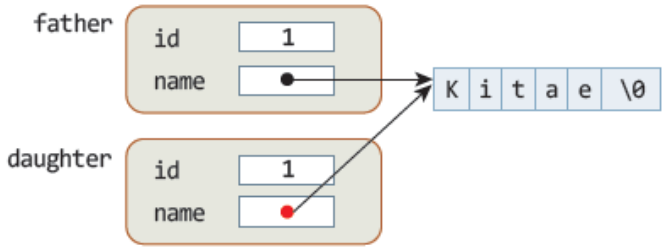
(1) `Person father(1, "Kitae");`

father 객체 생성



(2) `Person daughter(father);`

father를 복사한 daughter 객체 생성



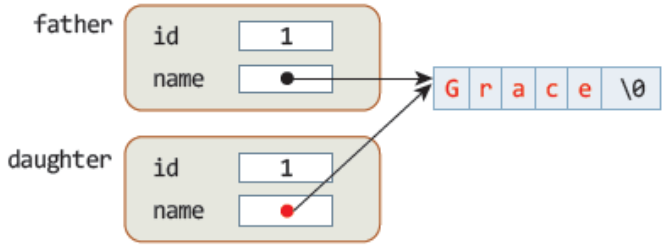
(3)
```
father.show();
daughter.show();
```

실행 결과

```
1,Kitae
1,Kitae
```

(4) `daughter.changeName("Grace");`

daughter의 이름 변경

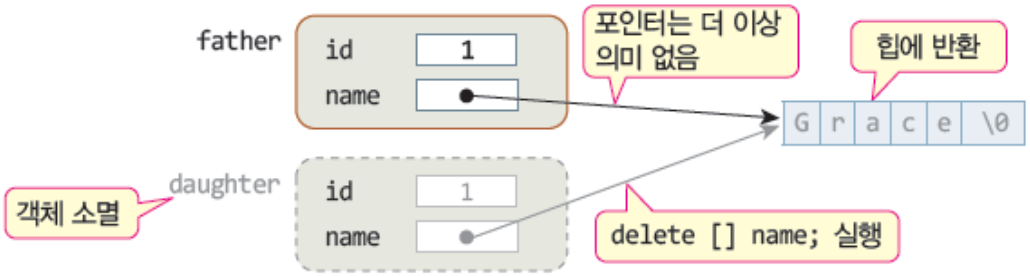


(5)
```
father.show();
daughter.show();
```

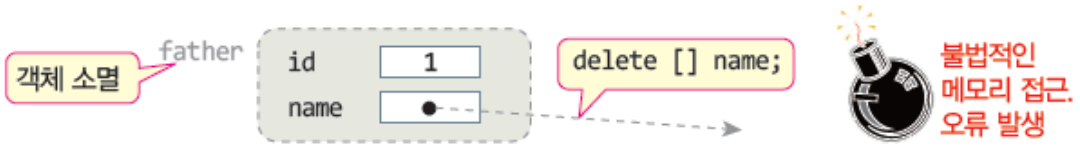
실행 결과

```
1,Grace
1,Grace
```

(6) daughter 객체 소멸



(7) father 객체 소멸



예제- 깊은 복사 생성자를 가진 정상적인 Person 클래스

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

class Person { // Person 클래스 선언
    char* name;
    int id;
public:
    Person(int id, const char* name); // 생성자
    Person(const Person& person); // 복사 생성자
    ~Person(); // 소멸자
    void changeName(const char *name);
    void show() { cout << id << ',' << name << endl; }
};

Person::Person(int id, const char* name) { // 생성자
    this->id = id;
    int len = strlen(name); // name의 문자 개수
    this->name = new char [len+1]; // name 문자열 공간 할당
    strcpy(this->name, name); // name에 문자열 복사
}

Person::Person(const Person& person) { // 복사 생성자
    this->id = person.id; // id 값 복사
    int len = strlen(person.name);// name의 문자 개수
    this->name = new char [len+1]; // name을 위한 공간 할당
    strcpy(this->name, person.name); // name의 문자열 복사
    cout << "복사 생성자 실행. 원본 객체의 이름 " << this->name << endl;
}

Person::~Person() {// 소멸자
    if(name) // 만일 name에 동적 할당된 배열이 있으면
        delete [] name; // 동적 할당 메모리 소멸
}

void Person::changeName(const char* name) { // 이름 변경
    if(strlen(name) > strlen(this->name))
        return; // 현재 name에 할당된 메모리보다 긴 이름으로 바꿀 수 없다.
    strcpy(this->name, name);
}
```

id 복사

name 복사

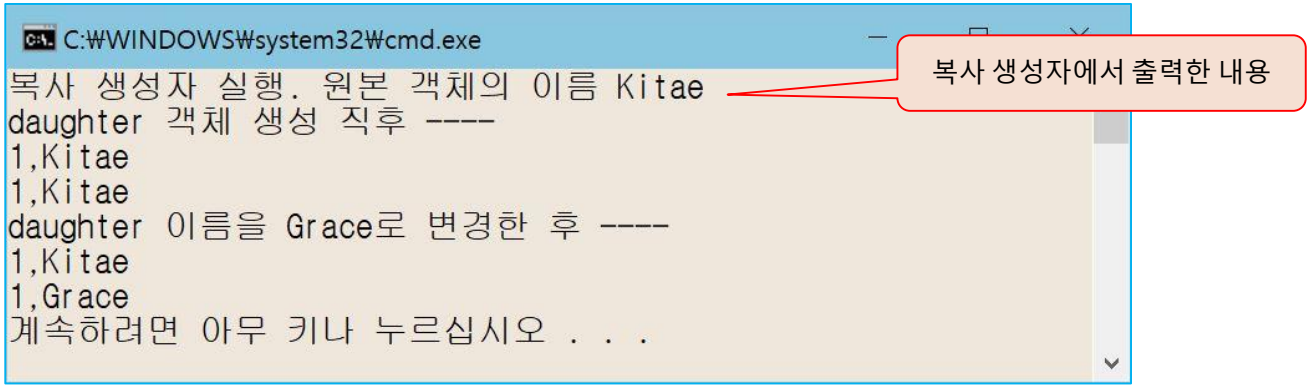
name 메모리 반환

```
int main() {
    Person father(1, "Kitae");          // (1) father 객체 생성
    Person daughter(father);            // (2) daughter 객체 복사 생성. 복사생성자호출

    cout << "daughter 객체 생성 직후 ----" << endl;
    father.show();                      // (3) father 객체 출력
    daughter.show();                    // (3) daughter 객체 출력

    daughter.changeName("Grace"); // (4) daughter의 이름을 "Grace"로 변경
    cout << "daughter 이름을 Grace로 변경한 후 ----" << endl;
    father.show();                      // (5) father 객체 출력
    daughter.show();                    // (5) daughter 객체 출력

    return 0;                           // (6), (7) daughter, father 객체 소멸
}
```

Person에 작성된 깊은 복사 생성자 호출

daughter, father 순으로 소멸

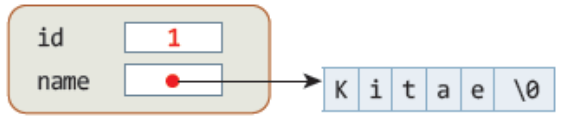
예제 실행 결과


C:₩WINDOWS₩system32₩cmd.exe
복사 생성자 실행. 원본 객체의 이름 Kitae
daughter 객체 생성 직후 ----
1,Kitae
1,Kitae
daughter 이름을 Grace로 변경한 후 ----
1,Kitae
1,Grace
계속하려면 아무 키나 누르십시오 . . .
복사 생성자에서 출력한 내용

실행 과정

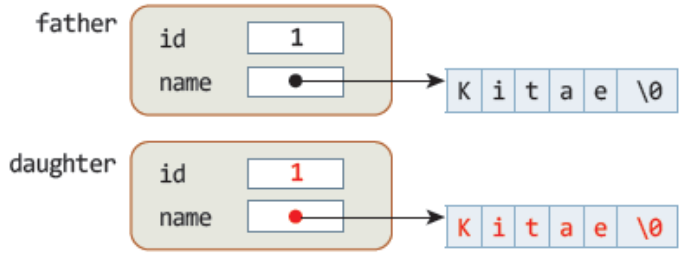
(1) Person father(1, "Kitae");

father 객체 생성



(2) Person daughter(father);

father를 복사한 daughter 객체 생성



실행 결과

```
복사 생성자 실행 Kitae
```

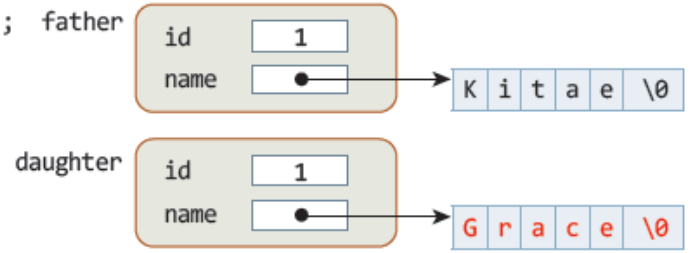
(3)
```
father.show();
daughter.show();
```

실행 결과

```
1,Kitae
1,Kitae
```

(4) daughter.changeName("Grace");

daughter의 이름 변경

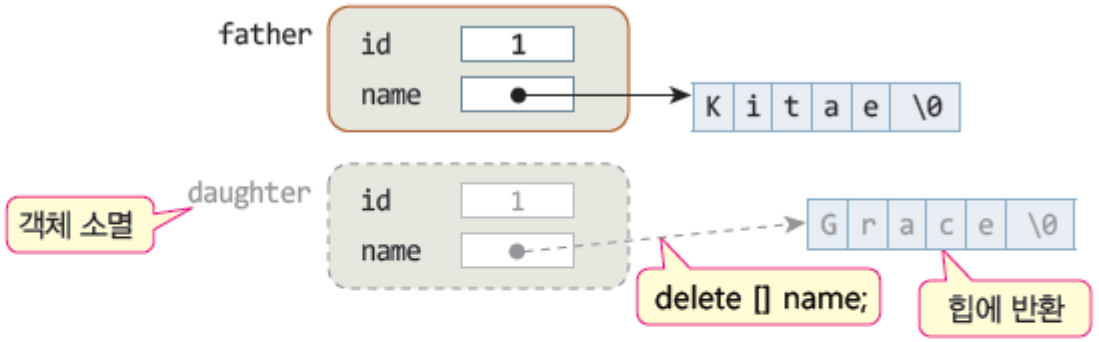


(5)
```
father.show();
daughter.show();
```

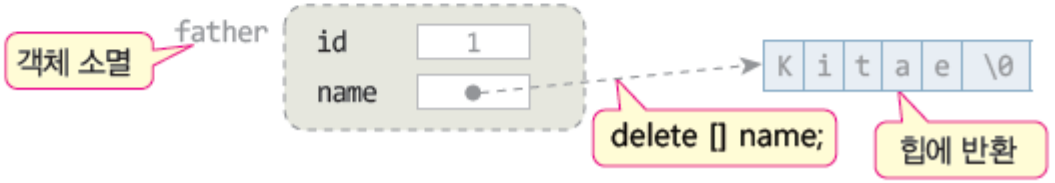
실행 결과

```
1,Kitae
1,Grace
```

(6) daughter 객체 소멸



(7) father 객체 소멸



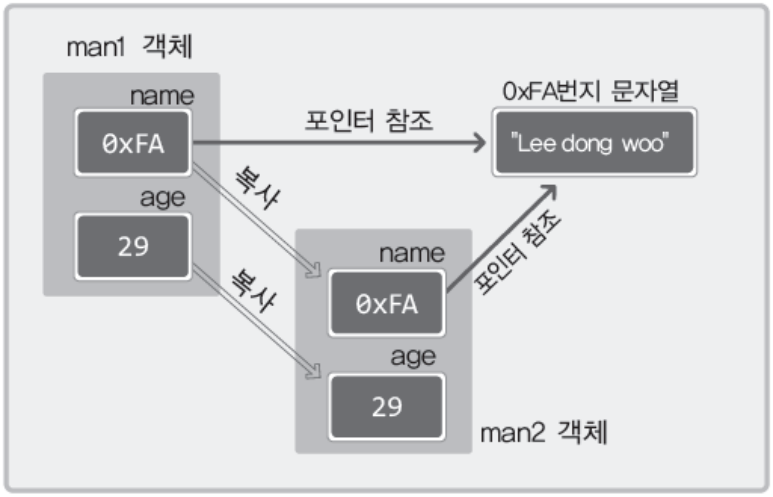
얕은 복사 예(2)
- 디폴트 복사 생성자의 문제점

```
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!

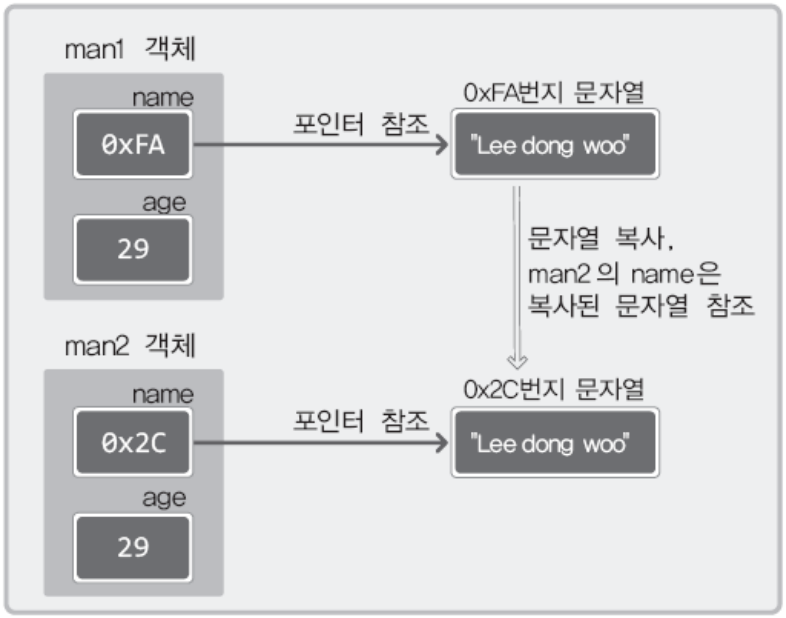
ShallowCopyError.cpp

```
#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;
}
```

해결안➔ 깊은 복사 : 복사 생성자 변경

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

깊은 복사를 구성하는 복사 생성자!!!



3. 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) & (3)

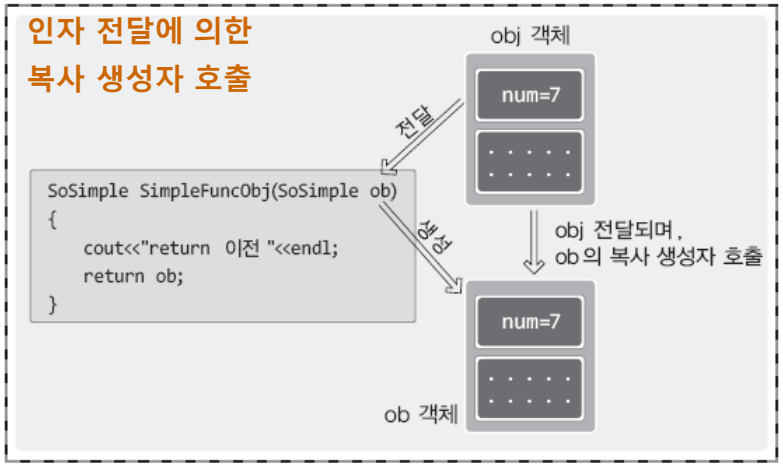
```
class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n)
    { }
    SoSimple(const SoSimple& copy) : num(copy.num)
    {
        cout<<"Called SoSimple(const SoSimple& copy)"<<endl;
    }
    SoSimple& AddNum(int n)
    {
        num+=n;
        return *this;
    }
    void ShowData()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

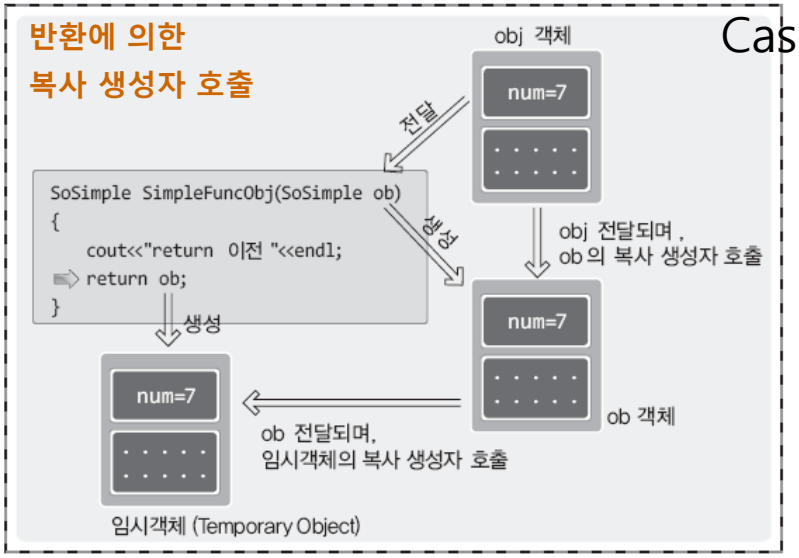
```
Called SoSimple(const SoSimple& copy)
return 이전
Called SoSimple(const SoSimple& copy)
num: 37
num: 7
```

실행결과

Case (2)



Case (3)



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;
}
```

참고문헌

- 뇌를 자극하는 C++ 프로그래밍, 이현창, 한빛미디어, 2011
- 열혈 C++ 프로그래밍(개정판), 윤성우, 오렌지미디어, 2012
- C++ ESPRESSO, 천인국 저, 인피니티북스, 2011
- 명품 C++ Programming, 황기태 , 생능출판사, 2018
- 어서와 C++는 처음이지, 천인국, 인피니티북스, 2018

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