05

함수와 참조, 복사 생성자

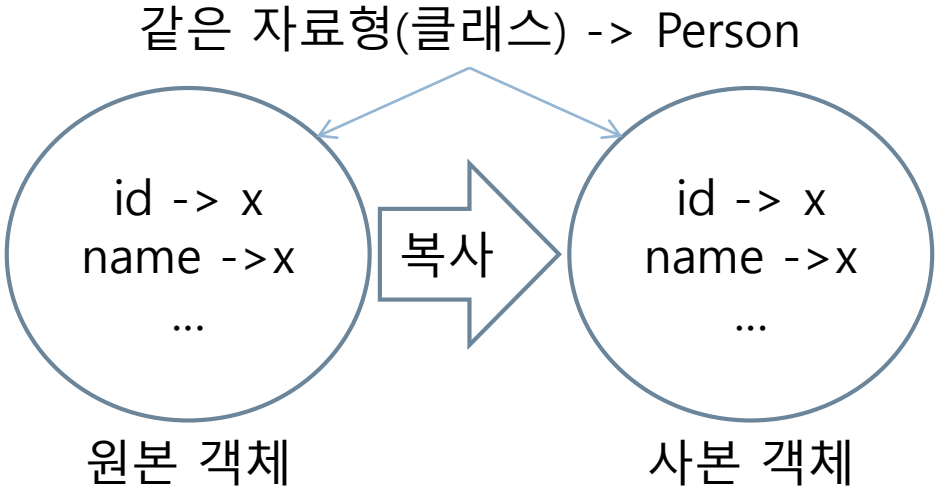
학습 목표

1. 복사생성자의 필요성과 활용을 이해하고, 작성할 수 있다.

객체의 복사

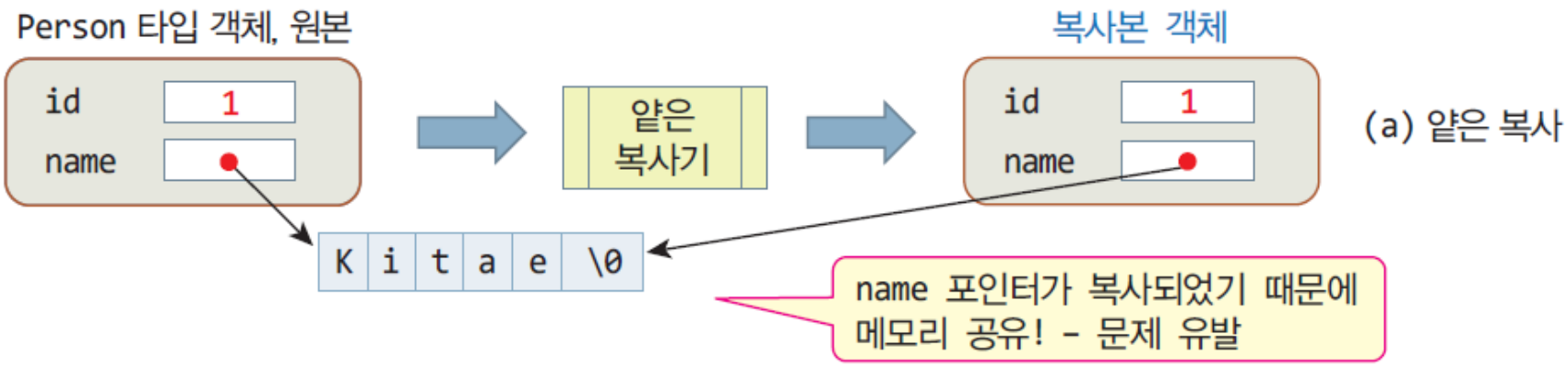
- 객체 원본과 동일한 별개의 사본 객체를 만드는 것
- 같은 자료형(클래스)의 객체들 사이에 멤버를 1:1로 복사함
- 얕은 복사(shallow copy), 깊은 복사(deep copy)
- 대입 연산자, 복사 생성자 통하여 구현 가능

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



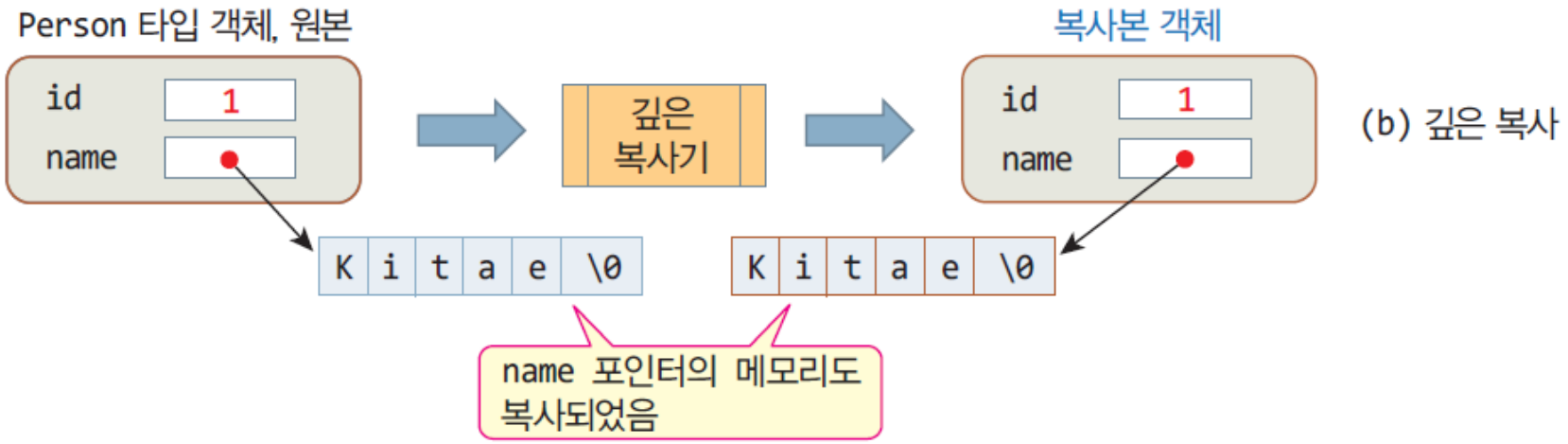
객체의 얕은 복사(shallow copy)

- 객체 복사 시, 객체의 멤버를 1:1로 복사
- 객체의 멤버 변수가 동적 메모리를 가리키는 경우 사본은 원본 객체가 할당 받은 동적 메모리를 공유하는 문제 발생



객체의 깊은 복사(deep copy)

- 객체 복사 시, 객체의 멤버를 1:1대로 복사
- 객체의 멤버 변수가 동적 메모리를 가리키는 경우
  - 사본은 원본이 가진 메모리 크기 만큼 별도로 동적 할당
  - 원본의 동적 메모리에 있는 내용을 사본 동적메모리에 복사
- 사본과 원본은 다른 동적메모리를 가리키므로 메모리를 공유하는 문제 없음



대입 연산에 의한 객체 복사

- 컴파일러에 의해 모든 객체를 생성할 때 디폴트 대입 연산자 함수가 자동으로 삽입됨
- 디폴트 대입연산자는 멤버를 1:1로 얕은 복사함
- 깊은 복사가 필요한 경우는 직접 정의해야 함(7장)

```
Circle c1(5), c2(30);
c1 = c2;            // c2객체를 c1객체에 복사 -> c1의 반지름 30됨
```

복사 생성자 (copy constructor)

- 객체의 복사 생성시 호출되는 특별한 생성자
- 자신이 속한 클래스형의 참조 매개 변수를 가지고 다른 매개변수는 없음 -> 클래스에 오직 한 개만 선언 가능
- 복사 생성자는 보통 생성자와 클래스 내에 중복 선언 가능
- 미리 생성된 같은 클래스 객체로 초기화 하고 싶을 때 사용

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

예제 5–9 Circle의 복사 생성자와 객체 복사


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

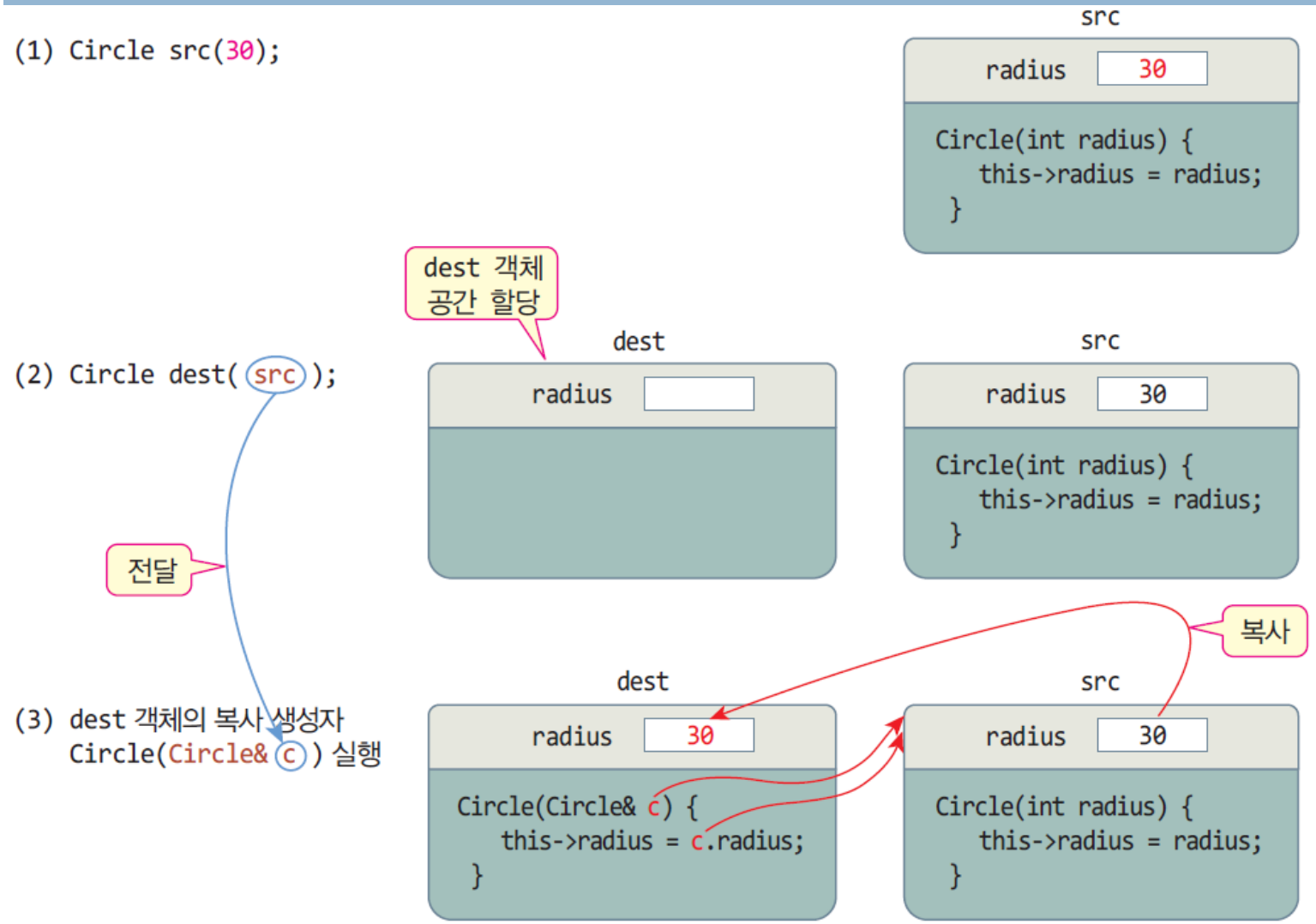
```
#include <iostream>
using namespace std;
class Circle {
private:
	int radius;
public:
	Circle(Circle& c);			// 복사 생성자 선언
	Circle() { radius = 1; }
	Circle(int radius) { this->radius = radius; }
	double getArea() { return 3.14 * radius * radius; }
};
Circle::Circle(Circle& c) {			// 복사 생성자 구현
	this->radius = c.radius;		// 멤버변수 직접 접근가능
	cout << "복사 생성자 실행 radius = " << radius << endl;
}
```

예제 5–9 Circle의 복사 생성자와 객체 복사


```
int main() {
	Circle src(30);		// 생성자 Circle(int radius) 호출
	Circle dest(src);	// 복사 생성자 Circle(Circle& c) 호출
	cout << "원본의 면적 = " << src.getArea() << endl;
	cout << "사본의 면적 = " << dest.getArea() << endl;
	return 0;
}
```

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

예제 5-9 Circle의 복사 생성자와 객체 복사



디폴트 복사 생성자

- 복사 생성자를 선언하지 않으면 컴파일러는 자동으로 디폴트 복사 생성자 삽입 -> 얕은 복사 수행

```
class Circle {
        int radius;
public:
        Circle(int r){ this->radius = r; }
        double getArea(){ return 3.14
        * radius * radius; }
};
int main(void)
{
        Circle src(10); // 생성자
        Circle dest(src); // 복사 생성자
}
```

복사생성자를 선언하지 않았는데 오류 발생하지 않음

```
class Circle {
        int radius;
public:
        Circle(int r){ this->radius = r; }
        Circle(Circle& c);
        double getArea(){ return 3.14 *
        radius * radius; }
};
Circle::Circle(Circle& c) {
        this->radius = c.radius;
}
int main(void)
{
        Circle src(10);
        Circle dest(src); // 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

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

컴파일러가 삽입하는 디폴트 복사 생성자

예제 5-10 얕은 복사 생성자의 문제점


```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
class Person {
	char* name;
	int id;
public:
	Person(int id, const char* name);
	~Person();
	void changeName(const char* name);
	void show() { cout << id << ',' << name << endl; }
};
```

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

컴파일러에 의해 디폴트 복사 생성자 삽입

예제 5–10 얕은 복사 생성자의 문제점


```
Person::Person(int id, const char* name) {    // 생성자
	this->id = id;
	int len = strlen(name);
	this->name = new char[len + 1];
	strcpy(this->name, name);
}
Person::~Person() {                           // 소멸자
	if (name) delete[ ] name;                 // name이 NULL이 아니면 소멸
}
void Person::changeName(const char* name) {
	if (strlen(name) > strlen(this->name)) return; // 에러처리
	strcpy(this->name, name);
}
```

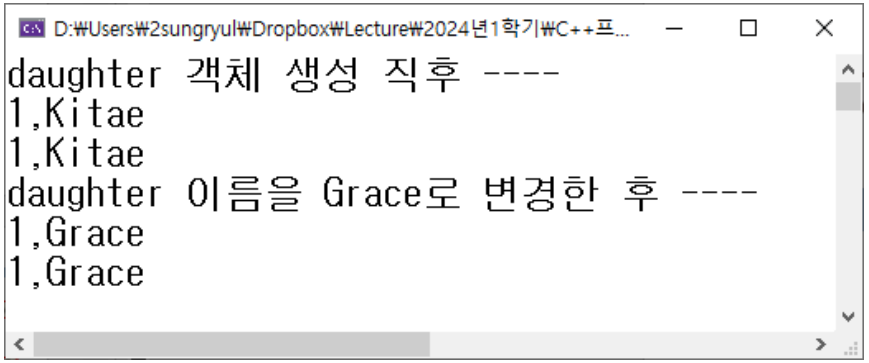
예제 5–10 얕은 복사 생성자의 문제점

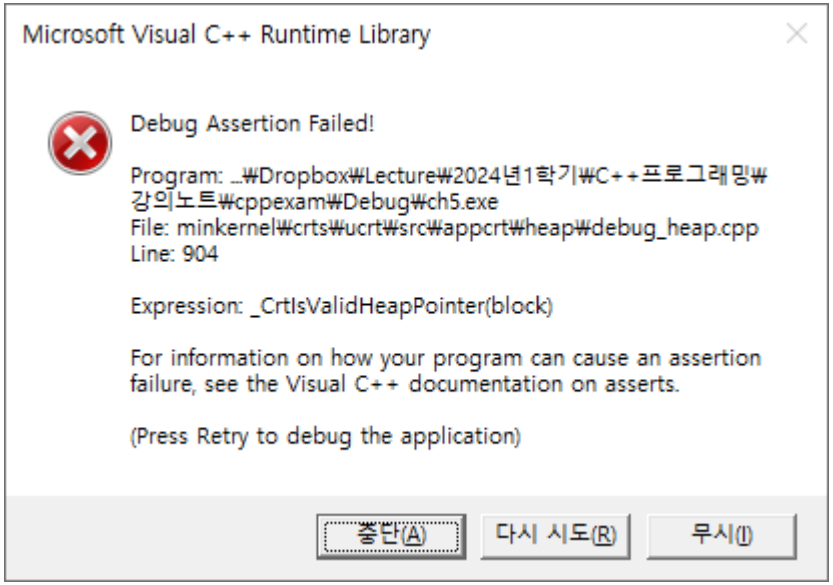

```
int main() {
	Person father(1, "Kitae");		// father 객체 생성자 호출
	Person daughter(father);		// daughter 객체의 복사생성자 호출

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

	cout << "daughter 이름을 Grace로 변경한 후 ----" << endl;
	father.show();				// father 객체 출력
	daughter.show();			// daughter 객체 출력
	return 0;				// daughter, father 객체 소멸
}
```

예제 5-10의 실행 결과



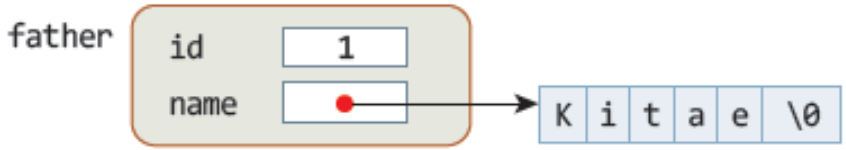


daughter, father 순으로 소멸 -> father가 소멸할 때, 프로그램 비정상 종료됨 -> why?

예제 5-10의 실행 과정

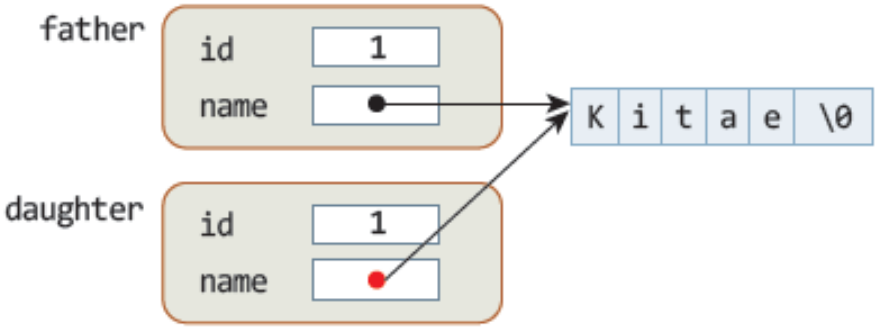
(1) Person father(1, "Kitae");

father 객체 생성



(2) Person daughter(father);

father를 복사한
daughter 객체 생성



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

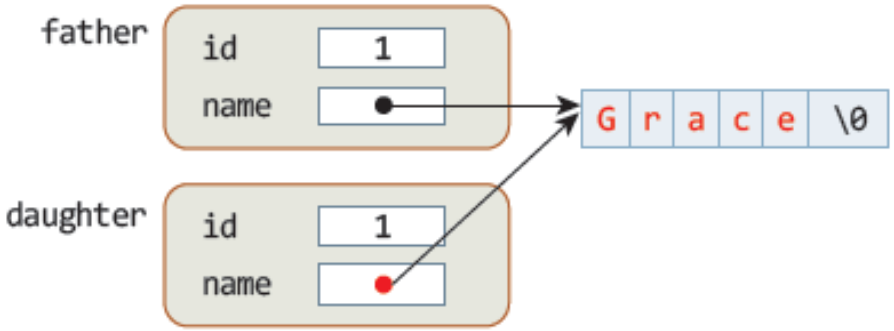
실행 결과

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

예제 5-10의 실행 과정


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

daughter의 이름 변경



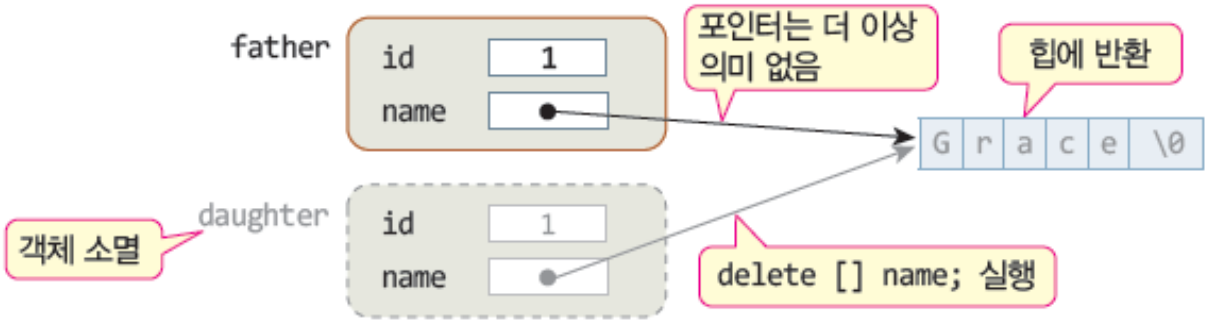

```
(5) father.show();
    daughter.show();
```

실행 결과

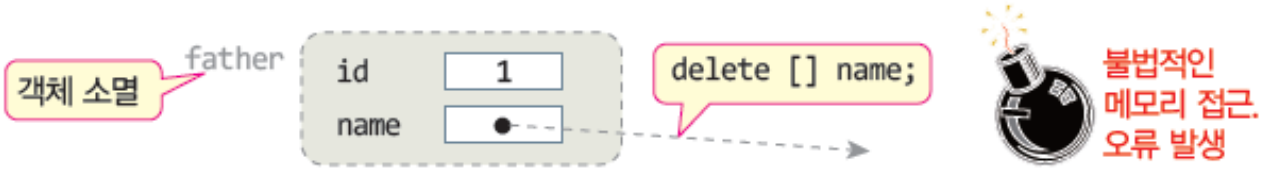
```
1,Grace
1,Grace
```

예제 5-10의 실행 과정

(6) daughter 객체 소멸



(7) father 객체 소멸



이미 반환한 메모리를 다시 반환하는 경우 실행 오류 발생함

예제 5–11 깊은 복사 생성자를 이용


```
#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(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
}
```

예제 5–11 깊은 복사 생성자를 이용


```
Person::Person(Person& person) {                // 복사 생성자
        this->id = person.id;
        int len = strlen(person.name);
        this->name = new char[len + 1];         // 새로운 메모리를 동적할당
        strcpy(this->name, person.name);
        cout << "복사 생성자 실행. 원본 객체의 이름 " << this->name << endl;
}
Person::~Person() {
        if (name) delete[ ] name;
}
void Person::changeName(const char* name) {
        if (strlen(name) > strlen(this->name)) return;
        strcpy(this->name, name);
}
```

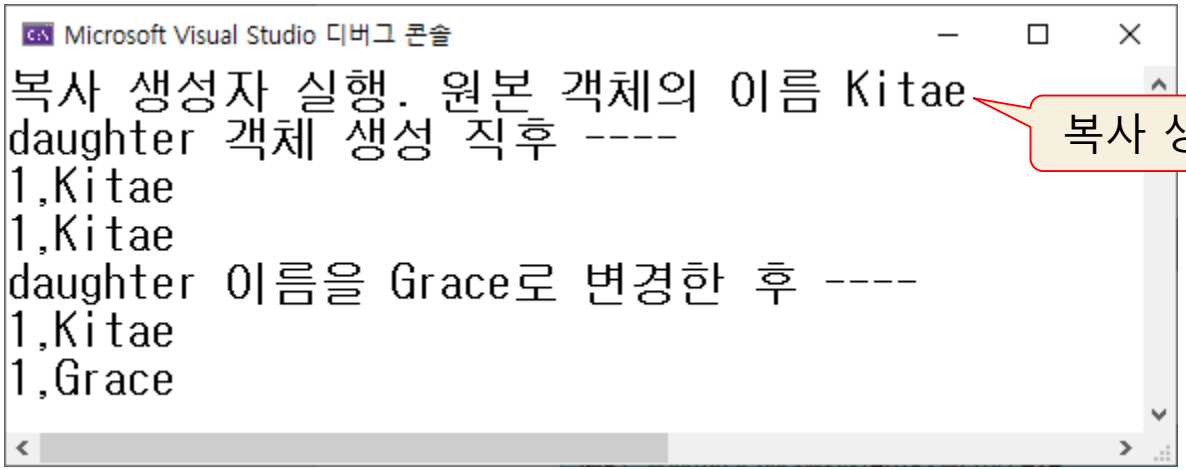
예제 5–11 깊은 복사 생성자를 이용


```
int main() {
	Person father(1, "Kitae");		// father 객체 생성자 호출
	Person daughter(father);		// daughter 객체의 복사생성자 호출

	cout << "daughter 객체 생성 직후 ----" << endl;
	father.show();				// father 객체 출력
	daughter.show();			// daughter 객체 출력
	daughter.changeName("Grace"); // daughter의 이름을 "Grace"로 변경

	cout << "daughter 이름을 Grace로 변경한 후 ----" << endl;
	father.show();				// father 객체 출력
	daughter.show();			// daughter 객체 출력
	return 0;				// daughter, father 객체 소멸
}
```

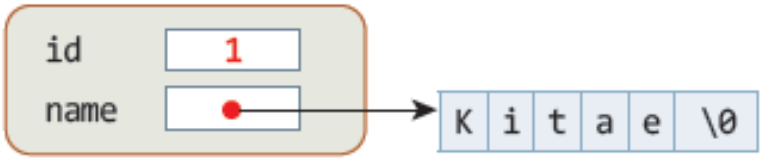
예제 5-11의 실행 결과


Microsoft Visual Studio 디버그 콘솔
복사 생성자 실행. 원본 객체의 이름 Kitae
daughter 객체 생성 직후 ----
1,Kitae
1,Kitae
daughter 이름을 Grace로 변경한 후 ----
1,Kitae
1,Grace
복사 생성자에서 출력한 내용

예제 5-11의 실행 과정

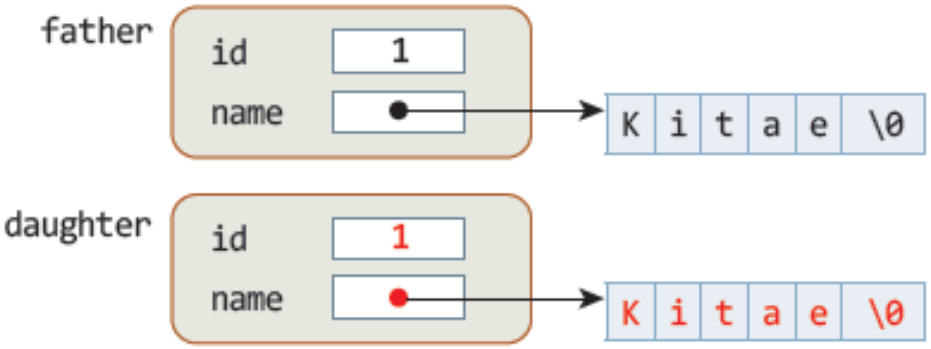
(1) Person father(1, "Kitae");

father 객체 생성



(2) Person daughter(father);

father를 복사한 daughter 객체 생성



실행 결과

복사 생성자 실행 Kitae

예제 5-11의 실행 과정

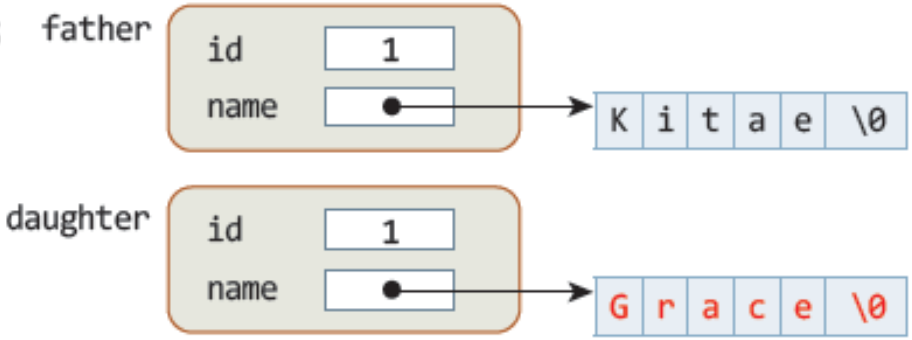

```
(3) father.show();
    daughter.show();
```

실행 결과

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

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

daughter의 이름 변경



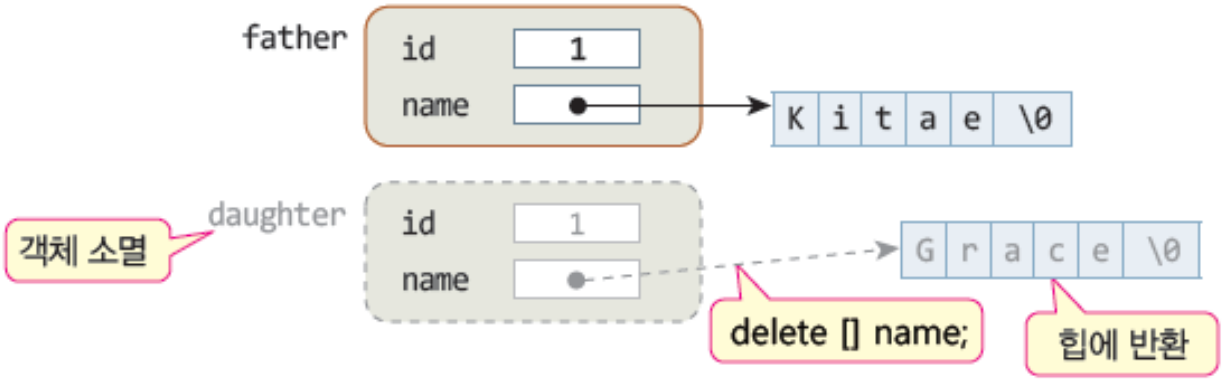

```
(5) father.show();
    daughter.show();
```

실행 결과

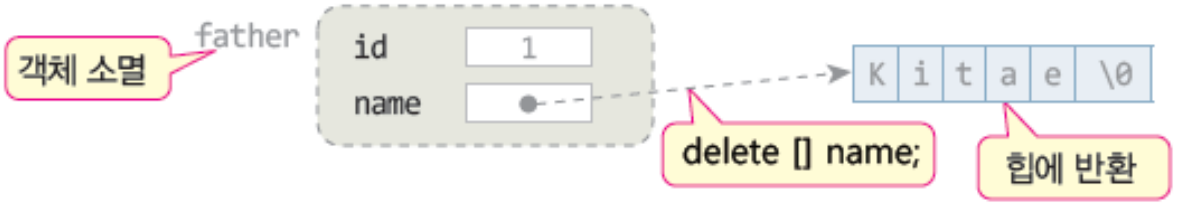
```
1,Kitae
1,Grace
```

예제 5-11의 실행 과정

(6) daughter 객체 소멸



(7) father 객체 소멸



father, daughter 객체의 소멸자가 각자의 동적 메모리를 반환하므로 정상 동작함

깊은 복사가 필요한 경우

- 멤버변수가 동적 할당된 메모리를 가리키는 경우 반드시 깊은 복사 생성자를 작성해야 함
- 멤버변수가 동적메모리와 관련 없다면 얕은 복사로 충분함
- 디폴트 복사 생성자는 얕은 복사만 수행

실습과제1

- 지금까지 배운 내용기준으로 컴파일러가 자동으로 추가하는 멤버함수는 어떤 것이 있는가?(4가지)
- 대입연산과 얕은 복사생성자의 차이는 무엇인가?

실습과제2

- 예제 5-11에서 문자열 저장을 위하여 char 배열을 사용하는데 이것을 string 클래스를 이용한 코드로 변경 하시오.
- 멤버변수, 멤버함수 수정이 필요한 곳을 모두 변경할 것
- string 클래스를 이용하면 문자열크기를 자동으로 조절하므로 동적할당을 사용할 필요가 없으므로 깊은 복사생성자를 작성할 필요 없음

실습과제3

- 교재 274페이지 11-(1), (2), (3)번을 푸시오.
- 예제 5-11번과 유사함
- Book java = cpp; -> Book java(cpp); 와 같음 -> 복사생성자 호출됨

실습과제4

- 교재 274페이지 11-(4) 번을 푸시오.

과제제출방법

- 소스코드, 라인단위의 주석, 실행결과를 포함하는 pdf파일을 작성한 후 eclass 과제 게시판에 업로드, **반드시 하나의 pdf파일로 업로드할 것**
- 기한 : 과제 게시판에 마감시간 참조
- 실행결과를 캡쳐할 때 글자를 알아보기 쉽게 확대해서 캡쳐할 것.
- 소스코드의 첫 부분은 아래처럼 제목,날짜,작성자(학번,이름)를 작성할 것

```
// ************************************************
//    제   목 :  정수 4개의 평균을 구하는 프로그램
//    날   짜 :  2023년 9월10일
//    작성자 :  15010101 홍길동
// ************************************************

// 소스코드 작성
```