

08. 함수와 참조, 복사생성자

8.1 객체전달과 참조

8.2 복사생성자

8.1 객체전달과 참조

'값에 의한 호출'로 객체 전달

- 함수를 호출하는 쪽에서 객체 전달
 - 객체 이름만 사용
 - 함수의 매개 변수 객체 생성
 - 매개 변수 객체의 공간이 스택에 할당
 - 호출하는 쪽의 객체가 매개 변수 객체에 그대로 복사됨
 - 매개 변수 객체의 생성자는 호출되지 않음
 - 함수 종료
 - 매개 변수 객체의 소멸자 호출
- 매개 변수 객체의 생성자 소멸자의 비대칭 실행 구조
- 값에 의한 호출 시 매개 변수 객체의 생성자가 실행되지 않는 이유?
 - 호출되는 순간의 실인자 객체 상태를 매개 변수 객체에 그대로 전달하기 위함

'값에 의한 호출' 방식으로 `increase(Circle c)` 함수가 호출되는 과정

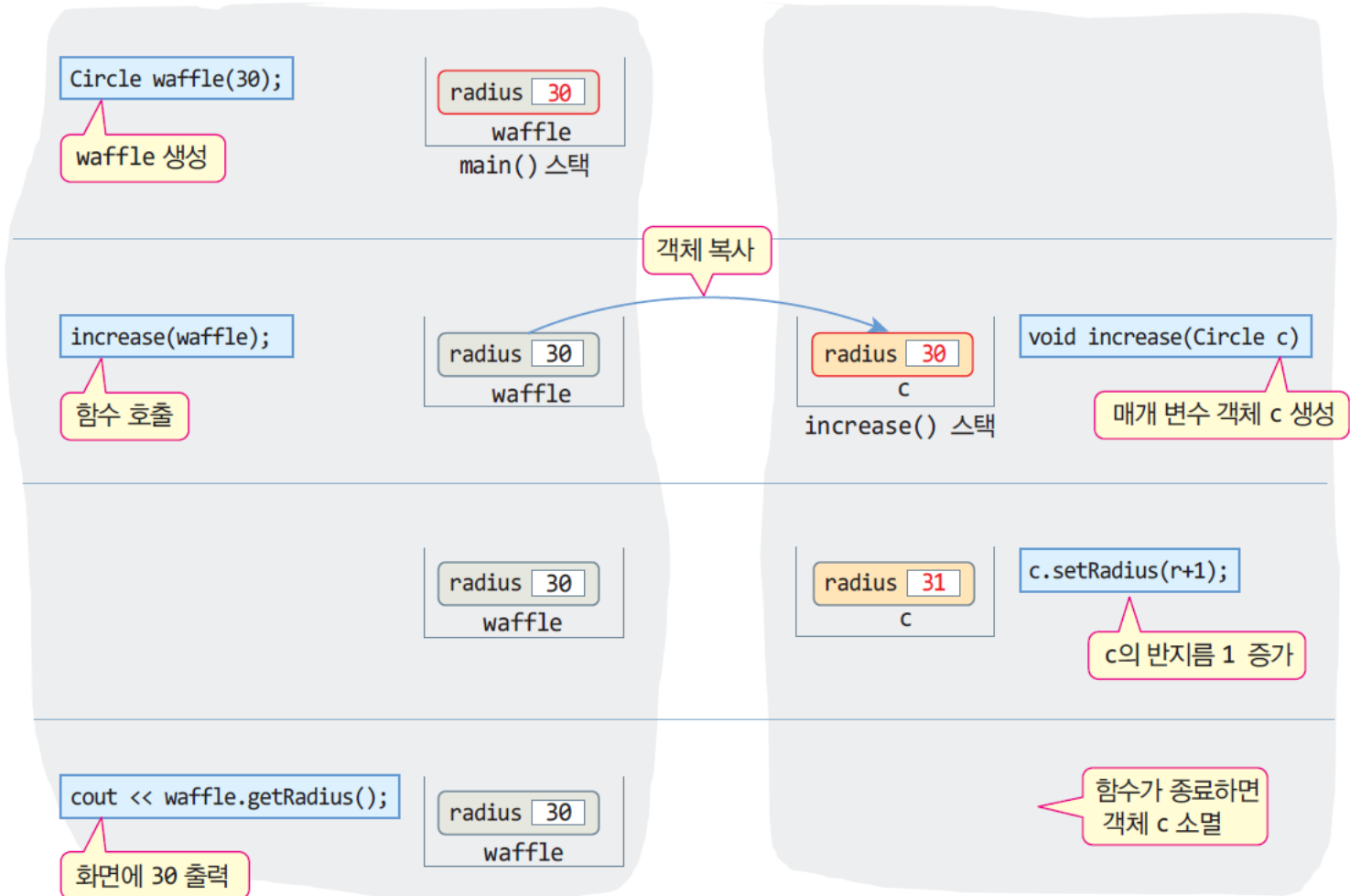
→ 실행 결과

30

```
int main() {  
    Circle waffle(30);  
    increase(waffle);  
    cout << waffle.getRadius() << endl;  
}
```

call by value

```
void increase(Circle c) {  
    int r = c.getRadius();  
    c.setRadius(r+1);  
}
```



```

#include <iostream>
using namespace std;

class Circle {
private:
    int radius;
public:
    Circle();
    Circle(int r);
    ~Circle();
    double getArea() { return 3.14*radius*radius; }
    int getRadius() { return radius; }
    void setRadius(int radius) { this->radius = radius; }
};

Circle::Circle() {
    radius = 1;
    cout << "생성자 실행 radius = " << radius << endl;
}

Circle::Circle(int radius) {
    this->radius = radius;
    cout << "생성자 실행 radius = " << radius << endl;
}

Circle::~~Circle() {
    cout << "소멸자 실행 radius = " << radius << endl;
}

```

```

void increase(Circle c) {
    int r = c.getRadius();
    c.setRadius(r+1);
}

int main() {
    Circle waffle(30);
    increase(waffle);
    cout << waffle.getRadius() << endl;
}

```

waffle의 내용이 그대로 c에 복사

waffle 생성

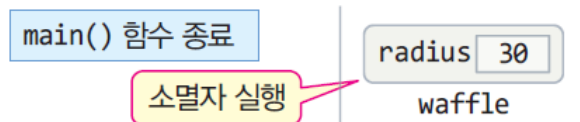
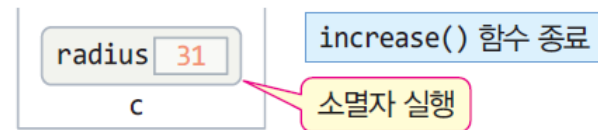
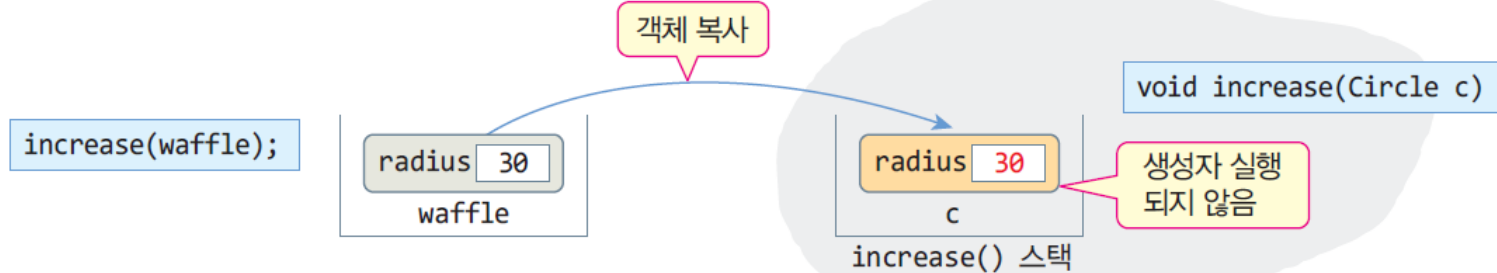
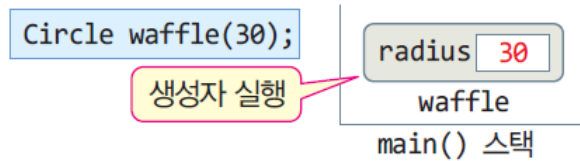
생성자 실행 radius = 30
 소멸자 실행 radius = 31
 30
 소멸자 실행 radius = 30

c의 생성자 실행되지 않았음

c 소멸

waffle 소멸

'값에 의한 호출'시에 생성자와 소멸자의 비대칭 실행



함수에 객체 전달 – ‘주소에 의한 호출’로

■ 함수 호출시 객체의 주소만 전달

- 함수의 매개 변수는 객체에 대한 포인터 변수로 선언
- 함수 호출 시 생성자 소멸자가 실행되지 않는 구조

'주소에 의한 호출'로 `increase(Circle *p)` 함수가 호출되는 과정

31

```
int main() {  
    Circle waffle(30);  
    increase(&waffle);  
    cout << waffle.getRadius();  
}
```

call by address

```
void increase(Circle *p) {  
    int r = p->getRadius();  
    p->setRadius(r+1);  
}
```

Circle waffle(30);

waffle 생성

radius 30

waffle
main() 스택

waffle의 주소가
p에 전달

increase(&waffle);

함수호출

radius 30

waffle

p

increase() 스택

void increase(Circle *p)

매개 변수 포인터 p 생성

radius 31

waffle

p

p->setRadius(r+1);

waffle의 반지름 1 증가

cout << waffle.getRadius();

31이 화면에 출력됨

radius 31

waffle

함수가 종료하면
포인터 p 소멸

객체 치환 및 객체 리턴

■ 객체 치환

- 동일한 클래스 타입의 객체끼리 치환 가능
- 객체의 모든 데이터가 비트 단위로 복사

```
Circle c1(5);  
Circle c2(30);  
c1 = c2; // c2 객체를 c1 객체에 비트 단위 복사. c1의 반지름 30됨
```

- 치환된 두 객체는 현재 내용물만 같을 뿐 독립적인 공간 유지

■ 객체 리턴

```
Circle getCircle() {  
    Circle tmp(30);  
    return tmp; // 객체 tmp 리턴  
}
```

```
Circle c; // c의 반지름 1  
c = getCircle(); // tmp 객체의 복사본이 c에 치환. c의 반지름은 30이 됨
```

객체 리턴

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

class Circle {
    int radius;
public:
    Circle() { radius = 1; }
    Circle(int radius) { this->radius = radius; }
    void setRadius(int radius) { this->radius = radius; }
    double getArea() { return 3.14*radius*radius; }
};

Circle getCircle() {
    Circle tmp(30);
    return tmp; // 객체 tmp를 리턴한다.
}

int main() {
    Circle c; // 객체가 생성된다. radius=1로 초기화된다.
    cout << c.getArea() << endl;

    c = getCircle();
    cout << c.getArea() << endl;
}
```

tmp 객체의 복사본이
리턴된다.

tmp 객체가 c에 복사된다.
c의 radius는 30이 된다.

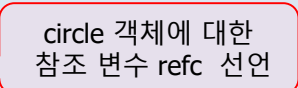
3.14
2826

객체에 대한 참조

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

class Circle {
    int radius;
public:
    Circle() { radius = 1; }
    Circle(int radius) { this->radius = radius; }
    void setRadius(int radius) { this->radius = radius; }
    double getArea() { return 3.14*radius*radius; }
};

int main() {
    Circle circle;
    Circle &refc = circle;
    refc.setRadius(10);
    cout << refc.getArea() << " " << circle.getArea();
}
```



314 314

참조에 의한 호출로 Circle 객체에 참조 전달

```
void increaseCircle(Circle &c) {  
    int r = c.getRadius();  
    c.setRadius(r+1);  
}
```

참조 매개 변수 c

```
int main() {  
    Circle waffle(30);  
    increaseCircle(waffle);  
    cout << waffle.getRadius() << endl;  
}
```

참조에 의한 호출

```
30 생성자 실행 radius = 30  
31
```

waffle 객체 생성

```
소멸자 실행 radius = 31
```

waffle 객체 소멸

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

```
class Circle {  
private:  
    int radius;  
public:  
    Circle();  
    Circle(int r);  
    ~Circle();  
    double getArea() { return 3.14*radius*radius; }  
    int getRadius() { return radius; }  
    void setRadius(int radius) { this->radius = radius; }  
};
```

```
Circle::Circle() {  
    radius = 1;  
    cout << "생성자 실행 radius = " << radius << endl;  
}
```

```
Circle::Circle(int radius) {  
    this->radius = radius;  
    cout << "생성자 실행 radius = " << radius << endl;  
}
```

```
Circle::~~Circle() {  
    cout << "소멸자 실행 radius = " << radius << endl;  
}
```

8.2 복사 생성자

C++에서 얇은 복사와 깊은 복사

■ 얇은 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;  
    .....  
};
```

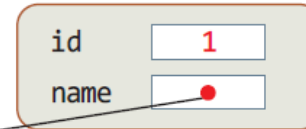
Person 타입 객체, 원본



얕은 복사기



복사본 객체



(a) 얕은 복사

K i t a e \0

name 포인터가 복사되었기 때문에
메모리 공유! - 문제 유발

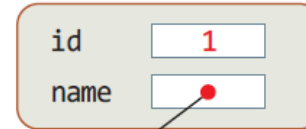
Person 타입 객체, 원본



깊은 복사기



복사본 객체



(b) 깊은 복사

K i t a e \0

K i t a e \0

name 포인터의 메모리도
복사되었음

객체 간의 초기화와 대입

- 같은 클래스의 객체 간에 서로 초기화나 대입이 가능
- 객체 간의 초기화 : 복사 생성자 이용
 - 같은 클래스의 다른 객체와 같은 값을 갖도록 초기화
 - 클래스의 멤버 변수를 1대1로 초기화
- 객체 간의 대입 : 대입 연산자 이용
 - 같은 클래스의 다른 객체의 값을 대입
 - 클래스의 멤버 변수를 1대1로 대입

```
Circle c1(10);  
Circle c2 = c1;    // == Circle c2(c1) : 복사생성자를 통한 초기화  
Circle c3;  
c3 = c1;           // 객체 간의 대입  
                    // c1 객체를 c3 객체에 비트 단위 복사
```


복사 생성자

■ 복사 생성자(copy constructor)란?

- 객체의 복사 생성시 호출되는 특별한 생성자
- 같은 클래스의 객체를 이용해서 초기화하는 생성자

■ 특징

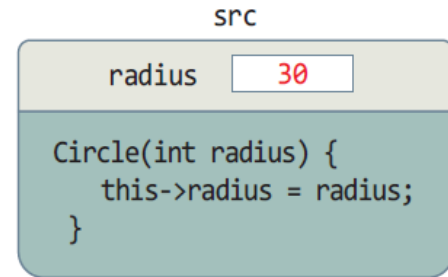
- 한 클래스에 **오직 1개**만 선언 가능
- 모양 : 클래스에 대한 **참조 매개 변수**를 가지는 독특한 **생성자**

복사 생성자의 원형	복사 생성자 호출방법
Circle(Circle& c); Circle(const Circle& c);	Circle c1; Circle c2 = c1; Circle c3(c1);

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

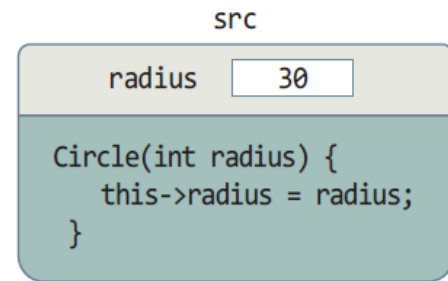
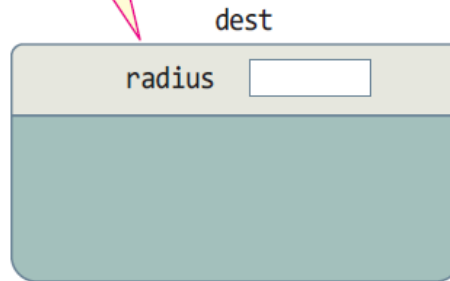
복사 생성 과정

(1) `Circle src(30);`



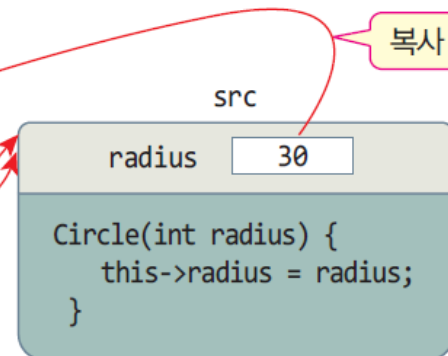
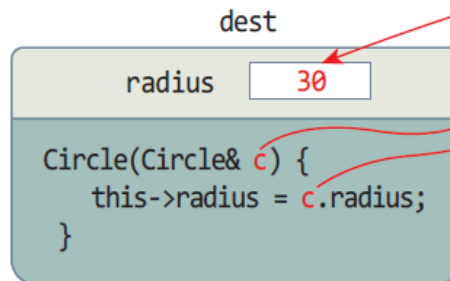
(2) `Circle dest(src);`

dest 객체
공간 할당



전달

(3) dest 객체의 복사 생성자
`Circle(Circle& c)` 실행



복사

Circle의 복사 생성자와 객체 복사

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

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

Circle::Circle(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(Circle& c)

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

디폴트 복사 생성자

- 개발자가 클래스에 복사 생성자를 작성해놓지 않으면
 - 컴파일러가 자동으로 디폴트 복사 생성자 만들어서 삽입

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

복사 생성자
없음

복사 생성자 없는데
컴파일 오류?

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

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

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

얕은 복사 생성자를 사용하여 프로그램이 비정상 종료되는 경우

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

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

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

컴파일러에 의해
디폴트 복사 생성자 삽입

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

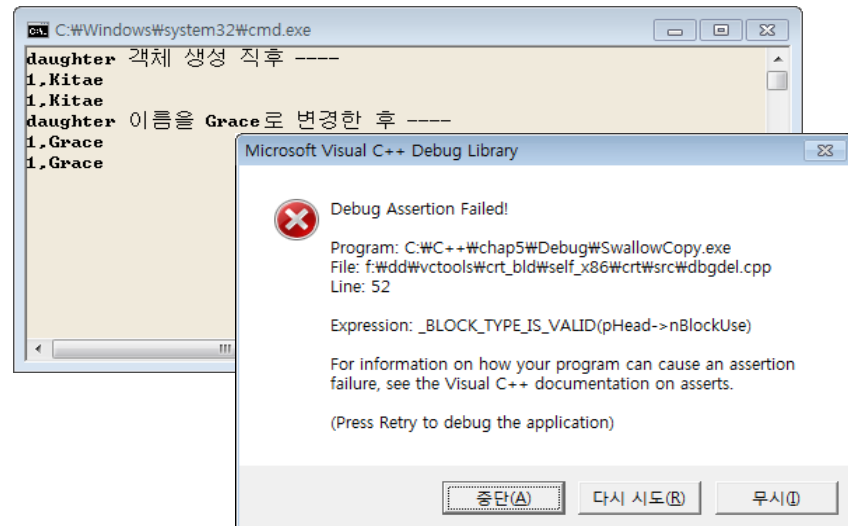
name 메모리 반환

얕은 복사 생성자를 사용하여 프로그램이 비정상 종료되는 경우

```
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  
}
```

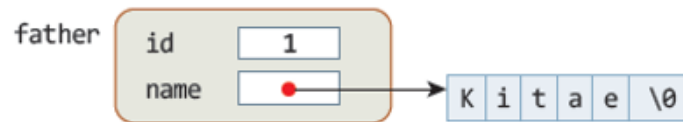
컴파일러가 삽입한
디폴트 복사 생성자 호출

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



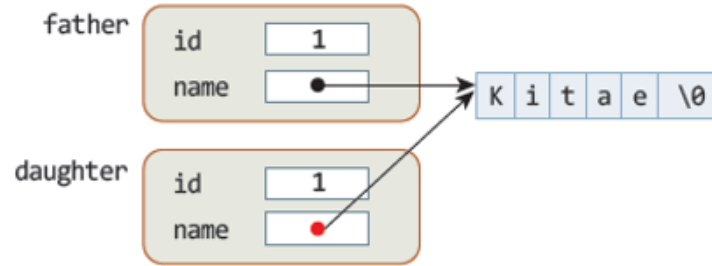
(1) `Person father(1, "Kitae");`

father 객체 생성



(2) `Person daughter(father);`

father를 복사한
daughter 객체 생성



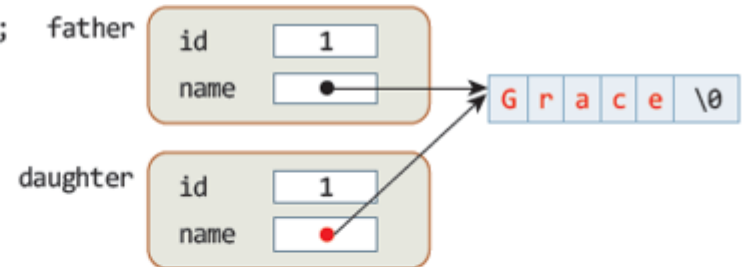
(3) `father.show();`
`daughter.show();`

→ 실행 결과

1,Kitae
1,Kitae

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

daughter의 이름
변경

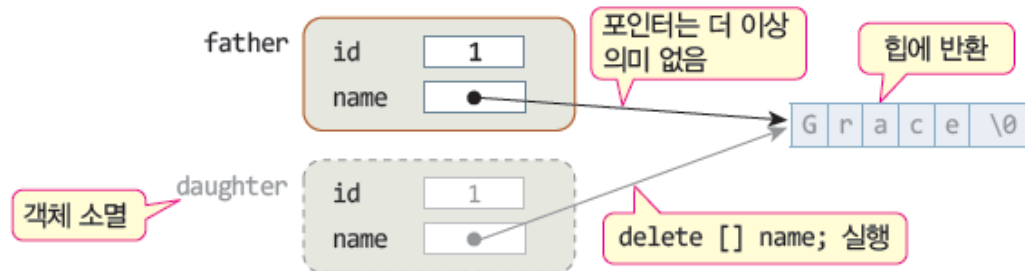


(5) `father.show();`
`daughter.show();`

→ 실행 결과

1,Grace
1,Grace

(6) daughter 객체 소멸



(7) father 객체 소멸



깊은 복사 생성자를 사용

- 깊은 복사를 하도록 새로 구현한 복사 생성자

```
class Person {  
    ...  
  
    Person(const Person& person); // 복사 생성자  
  
};  
  
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  
}
```

묵시적 복사 생성에 의해 복사 생성자가 자동 호출되는 경우

```
void f(Person p) {  
    p.changeName("Kim Young woo")  
}
```

2. '값에 의한 호출'로 객체가 전달될 때.
person 객체의 복사 생성자 호출

```
Person g() {  
    Person mother(2, "Hyun So Bin");  
    return mother;  
}
```

3. 함수에서 객체를 리턴할 때.
mother 객체의 복사본 생성.
복사본의 복사 생성자 호출

```
int main() {  
    Person father(1, "Kim Min Soo");  
    Person mother = father;  
    f(father);  
    g();  
}
```

1. 객체로 초기화하여 객체가 생성될 때.
son 객체의 복사 생성자 호출

복사 생성자 실행 Kim Min Soo
복사 생성자 실행 Kim Min Soo
복사 생성자 실행 Hyun So Bin

다음 수업

■ friend와 연산자중복

- 1_ C++에서의 friend
- 2_ 연산자중복