

5장 함수와 참조, 복사 생성자

- 1) 함수 호출 시 객체 전달
- 2) 객체 치환 및 리턴
- 3) 참조와 함수
- 4) 복사 생성자



함수의 인자 전달

함수의 인자 전달 방식 리뷰

- 인자 전달 방식

- 1) 값에 의한 호출, call by value

- ▶ 함수가 호출되면 매개 변수가 스택에 생성됨
 - ▶ 호출하는 코드에서 값을 넘겨줌
 - ▶ 호출하는 코드에서 넘어온 값이 매개 변수에 복사됨

- 2) 주소에 의한 호출, call by address

- ▶ 함수의 매개 변수는 포인터 타입
 - ▶ 함수가 호출되면 포인터 타입의 매개 변수가 스택에 생성됨
 - ▶ 호출하는 코드에서는 명시적으로 주소를 넘겨줌
 - ▶ 기본 타입 변수나 객체의 경우, 주소 전달
 - ▶ 배열의 경우, 배열의 이름
 - ▶ 호출하는 코드에서 넘어온 주소 값이 매개 변수에 저장됨

함수의 인자 전달 방식 리뷰



(a) 값에 의한 호출



(b) 주소에 의한 호출

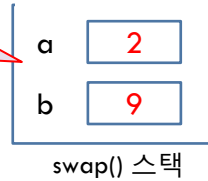
```
#include <iostream>
using namespace std;
```

```
void swap(int a, int b) {
    int tmp;

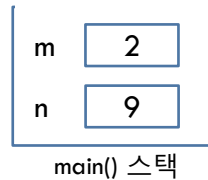
    tmp = a;
    a = b;
    b = tmp;
}
```

```
int main() {
    int m=2, n=9;
    swap(m, n);
    cout << m << " " << n;
}
```

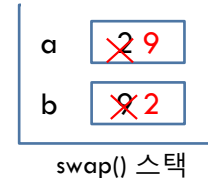
a, b에
m, n의
값 복사



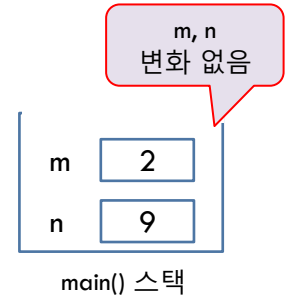
(1) swap() 호출 전



(2) swap() 호출 직후



(3) swap() 실행



(4) swap() 리턴 후

2 9

값에 의한 호출

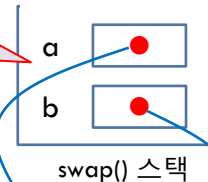
```
#include <iostream>
using namespace std;
```

```
void swap(int *a, int *b) {
    int tmp;

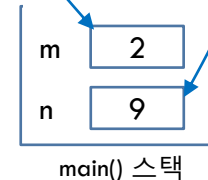
    tmp = *a;
    *a = *b;
    *b = tmp;
}
```

```
int main() {
    int m=2, n=9;
    swap(&m, &n);
    cout << m << " " << n;
}
```

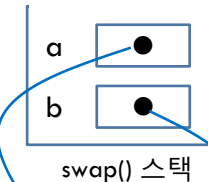
a, b에
m, n의
주소 전달



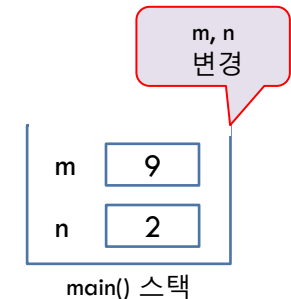
(1) swap() 호출 전



(2) swap() 호출 직후



(3) swap() 실행



(4) swap() 리턴 후

9 2

주소에 의한 호출

“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(복사 생성자가 수행)
- 함수 종료
 - ▶ 매개 변수 객체의 소멸자 호출
- 값에 의한 호출 시 매개 변수 객체의 생성자가 실행되지 않는 이유?
 - ▶ 호출되는 순간의 실 인자 객체 상태를 매개 변수 객체에 그대로 전달하기 위함

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);
}
```

Circle waffle(30);

waffle 생성

radius 30

waffle

main() 스택

객체 복사(객체 복사를 위해 복사 생성자 호출)

increase(waffle);

함수 호출

radius 30

waffle

radius 30

c

increase() 스택

void increase(Circle c)

매개 변수 객체 c 생성

radius 30

waffle

radius 31

c

c.setRadius(r+1);

c의 반지름 1 증가

cout << waffle.getRadius() ;

화면에 30 출력

radius 30

waffle

함수가 종료하면
객체 c 소멸

예제 5-1 “값에 의한 호출”시 매개 변수의 생성자 실행되지 않음

```
#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

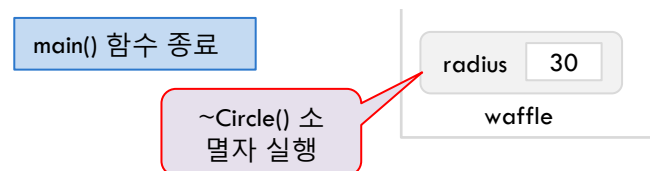
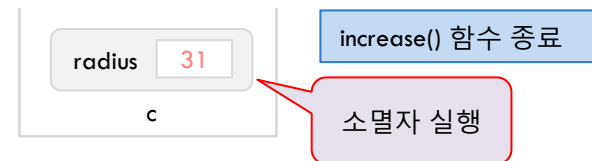
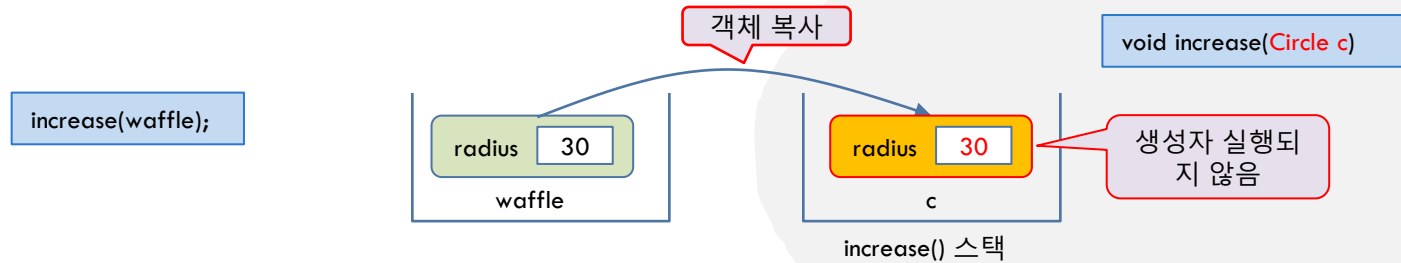
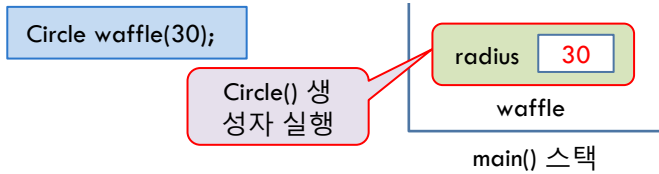
c의 생성자 실행되지 않았음

c 소멸

waffle 소멸

소멸자 실행 radius = 30

“값에 의한 호출”시에 생성자와 소멸자의 비대칭 실행



함수에 객체 전달 – “주소에 의한 호출”로

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

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

increase(&waffle);

함수 호출

radius 30

waffle

waffle의
주소가 p
에 전달

p

void increase(Circle *p)

매개 변수 p 생성

increase() 스택

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 단위로 복사한다.
```

- ▶ 치환된 두 객체는 현재 내용물만 같을 뿐 독립적인 공간 유지

- 객체 리턴(복사 생성자 수행)

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

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

예제 5-3 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. c
의 radius는 30이 된다.

3.14
2826

참조란?



메뚜기는 유재석의 별명



참조(reference)란 가리킨다는 뜻으로,
이미 존재하는 객체나 변수에 대한 별명

참조의 활용

- 참조 변수
- 참조에 의한 호출
- 참조 리턴

참조 변수

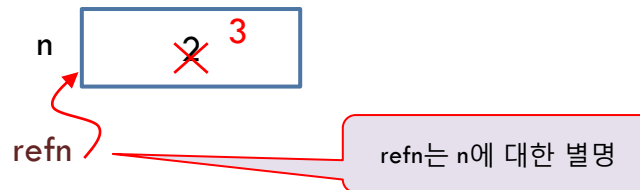
- 참조 변수 선언
 - ▶ 참조자 &의 도입
 - ▶ 이미 존재하는 변수에 대한 다른 이름(별명)을 선언
 - ▶ 참조 변수는 이름만 생기며
 - ▶ 참조 변수에 새로운 공간을 할당하지 않는다.
 - ▶ 초기화로 지정된 기존 변수를 공유한다.

```
int n=2;  
int &refn = n; // 참조 변수 refn 선언. refn은 n에 대한 별명
```

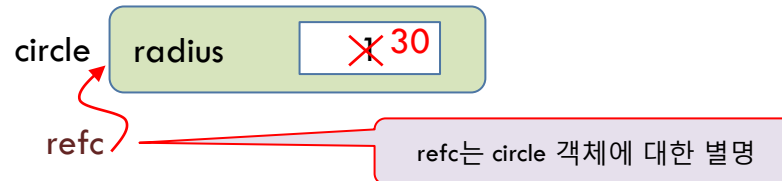
```
Circle circle;  
Circle &refc = circle; // 참조 변수 refc 선언. refc는 circle에 대한 별명
```

참조 변수 선언 및 사용 사례

```
int n = 2;  
int &refn = n;  
  
refn = 3;
```



```
Circle circle;  
Circle &refc = circle;  
  
refc.setRadius(30);
```



refc->setRadius(30);으로 하면 안 됨

예제 5-3 기본 자료형에 대한 참조

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

int main() {
    cout << "i" << '\t' << "n" << '\t' << "refn" << endl;
    int i = 1;
    int n = 2;
    int &refn = n; // 참조 변수 refn 선언. refn은 n에 대한 별명
    n = 4;
    refn++; // refn=5, n=5
    cout << i << '\t' << n << '\t' << refn << endl;

    refn = i; // refn=1, n=1
    refn++; // refn=2, n=2
    cout << i << '\t' << n << '\t' << refn << endl;

    int *p = &refn; // p는 n의 주소를 가짐
    *p = 20; // refn=20, n=20
    cout << i << '\t' << n << '\t' << refn << endl;
}
```

참조 변수 refn 선언

참조에 대한
포인터 변수 선언

i	n	refn
1	5	5
1	2	2
1	20	20

예제 5-4 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();
}
```

circle 객체에 대한
참조 변수 refc 선언

314 314

참조에 의한 호출

- 참조를 가장 많이 활용하는 사례
- call by reference라고 부름
- 함수 형식
 - ▶ 함수의 매개 변수를 참조 타입으로 선언
 - ▶ 참조 매개 변수(reference parameter)라고 부름
 - ▶ 참조 매개 변수는 실 인자 변수를 참조함
 - ▶ 참조매개 변수의 이름만 생기고 공간이 생기지 않음
 - ▶ 참조 매개 변수는 실 인자 변수 공간 공유
 - ▶ 참조 매개 변수에 대한 조작은 실 인자 변수 조작 효과

참조에 의한 호출 사례

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

```
void swap(int &a, int &b) {
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
}
```

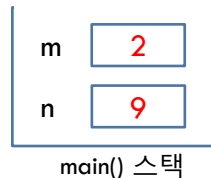
```
int main() {
    int m=2, n=9;
    swap(m, n);
    cout << m << ' ' << n;
}
```

참조 매개 변수 a, b

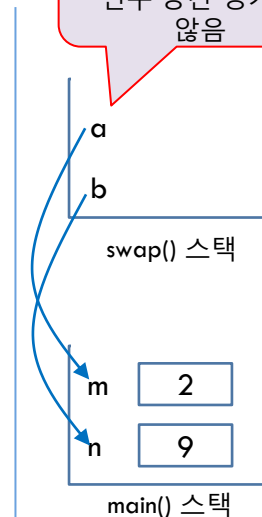
참조 매개 변수를
보통 변수처럼 사용

함수가 호출되면
m, n에 대한 참조
변수 a, b가 생긴
다.

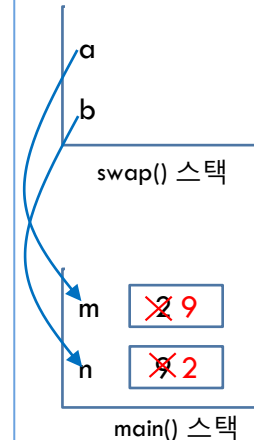
a, b는 m, n의 별명.
a, b 이름만 생성.
변수 공간 생기지
않음



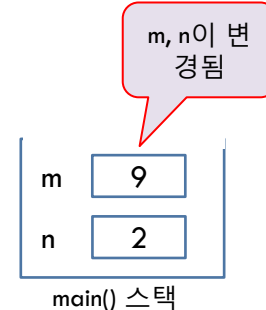
(1) swap() 호출 전



(2) swap() 호출 직후



(3) swap() 실행



(4) swap() 리턴 후

9 2

예제 5-5 기본 자료형을 참조 매개 변수로 사용

참조 매개 변수를 통해 평균을 리턴하고 리턴문을 통해서 함수의 성공 여부를 리턴하도록 `average()` 함수를 작성하라

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

```
bool average(int a[], int size, int& avg) {
    if(size <= 0)
        return false;
    int sum = 0;
    for(int i=0; i<size; i++)
        sum += a[i];
    avg = sum/size;
    return true;
}
```

참조 매개 변수 `avg`에 평균 값 전달

`avg`에 평균이 넘어오고, `average()`는 `true` 리턴

`avg`의 값은 의미없고, `average()`는 `false` 리턴

```
int main() {
    int x[] = {0,1,2,3,4,5};
    int avg;
    if(average(x, 6, avg)) cout << "평균은 " << avg << endl;
    else cout << "매개 변수 오류" << endl;

    if(average(x, -2, avg)) cout << "평균은 " << avg << endl;
    else cout << "매개 변수 오류 " << endl;
}
```

평균은 2
매개 변수 오류

예제 5-6 Circle 객체를 참조 매개 변수로 사용

참조 매개 변수 c

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

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

참조에 의한 호출

생성자 실행 radius = 30
31

소멸자 실행 radius = 31

waffle 객체 생성

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 }  
};
```

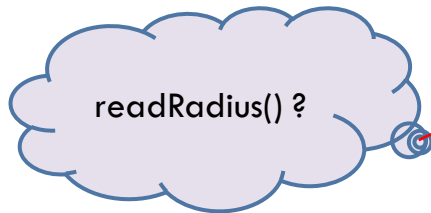
```
Circle::Circle() {  
    radius = 1;  
    cout << "생성자 실행 radius = " << radius << endl;  
}
```

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

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

예제 5-7(실습) 참조 매개 변수를 가진 함수 만들기 연습

키보드로부터 반지름 값을 읽어
Circle 객체에 반지름을 설정하는
readRadius() 함수를 작성하라.



```
#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 donut;
    readRadius(donut);
    cout << "donut의 면적 = " << donut.getArea() << endl;
}
```

정수 값으로 반지름을 입력하세요>>3
donut의 면적 = 28.26

예제 5-7 정답

```
void readRadius(Circle &c) {  
    int r;  
    cout << "정수 값으로 반지름을 입력하세요>>";  
    cin >> r; // 반지름 값 입력  
    c.setRadius(r); // 객체 c에 반지름 설정  
}
```


참조 리턴

- C 언어의 함수 리턴

- ▶ 함수는 반드시 값만 리턴
 - ▶ 기본 타입 값 : int, char, double 등
 - ▶ 포인터 값

- C++의 함수 리턴

- ▶ 함수는 값 외에 참조 리턴 가능
- ▶ 참조 리턴
 - ▶ 변수 등과 같이 현존하는 공간에 대한 참조 리턴
 - ▶ 변수의 값을 리턴하는 것이 아님
- ▶ 참조 리턴된 값은 경우에 따라 lvalue가 되기도 하고 rvalue가 되기도 한다.

값을 리턴하는 함수 vs. 참조를 리턴하는 함수

문자
리턴

```
char c = 'a';

char get() { // char 리턴
    return c; // 변수 c의 문자('a') 리턴
}

char a = get(); // a = 'a'가 됨

get() = 'b'; // 컴파일 오류
```

char 타입
의 공간에
대한 참조
리턴

```
char c = 'a';

char& find() { // char 타입의 참조 리턴
    return c; // 변수 c에 대한 참조 리턴
}

char a = find(); // a = 'a'가 됨

char &ref = find(); // ref는 c에 대한 참조
ref = 'M'; // c = 'M'

find() = 'b'; // c = 'b'가 됨
```

find()가 리턴한 공
간에 'b' 문자 저장

(a) 문자 값을 리턴하는 get()

(b) char 타입의 참조(공간)을 리턴하는 find()

예제 5-8 간단한 참조 리턴 사례

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

```
char& find(char s[], int index) {
    return s[index]; // 참조 리턴
}
```

s[index] 공간의 참조 리턴

```
int main() {
    char name[] = "Mike";
    cout << name << endl;
```

find()가 리턴한 위치에 문자 'S' 저장

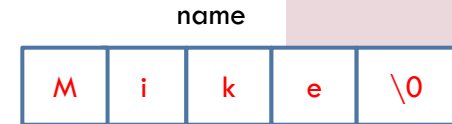
```
    find(name, 0) = 'S'; // name[0]='S'로 변경
    cout << name << endl;
```

```
    char& ref = find(name, 2);
    ref = 't'; // name = "Site"
    cout << name << endl;
```

ret는 name[2] 참조

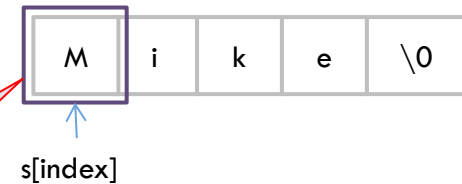
Mike
Sike
Site

(1) char name = "Mike";

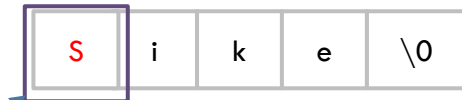


(2) return s[index];

공간에 대한 참조, 즉 이름의 이름 리턴



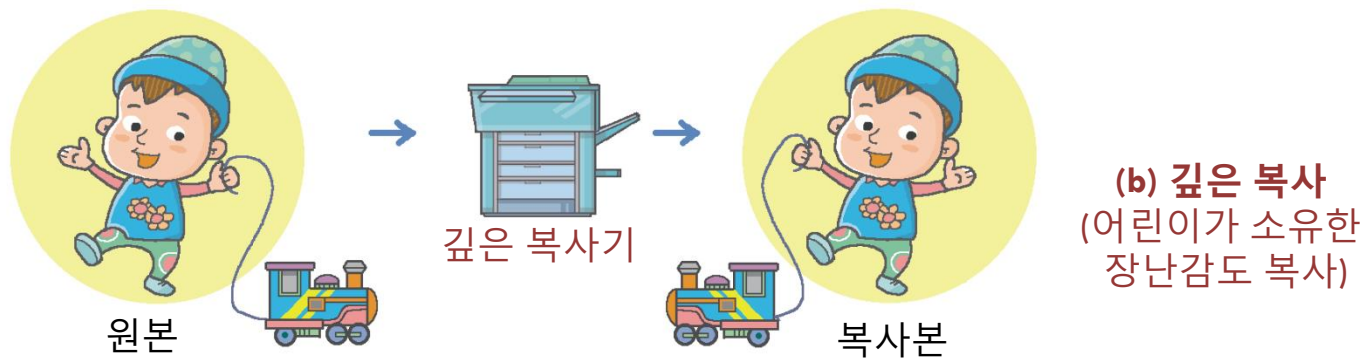
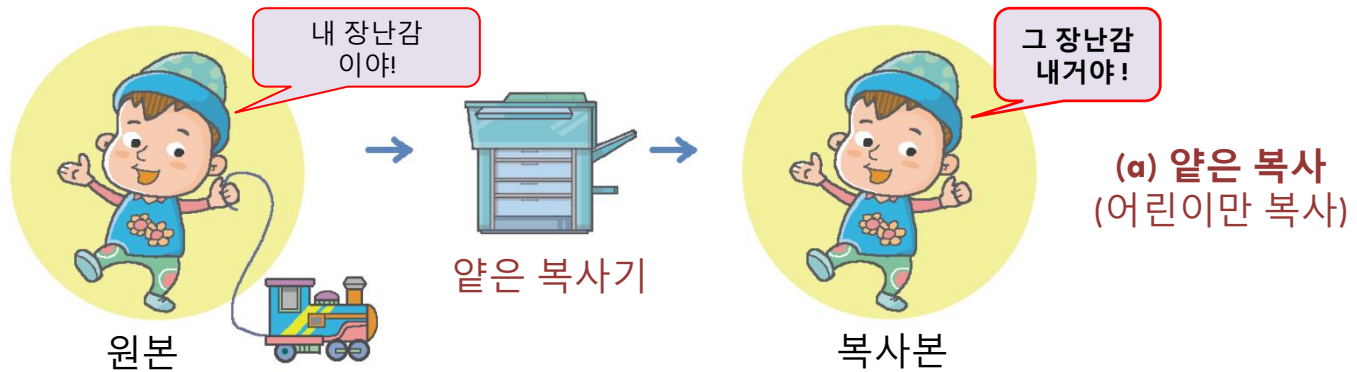
(3) find(name, 0) = 'S';



(4) ret = 't';



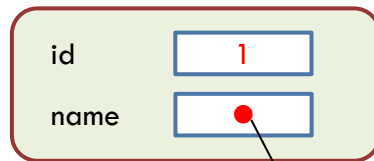
얇은 복사와 깊은 복사



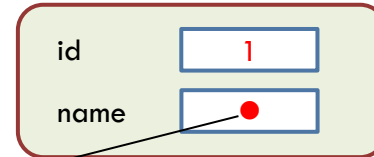
C++에서 객체의 복사

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

Person 타입 객체, 원본



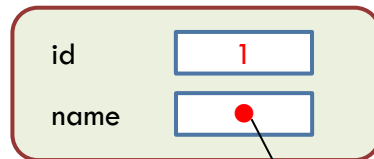
복사본 객체



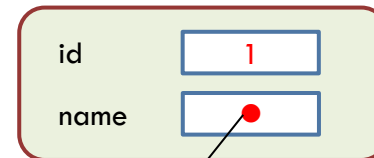
(a) 얕은 복사

name 포인터가 복사되었기 때문에 메모리 공유!! - 문제 유발

Person 타입 객체, 원본



복사본 객체



(b) 깊은 복사

name 포인터의 메모리도 각각 복사되었음

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 1:1로 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**



복사 생성자

복사 생성자

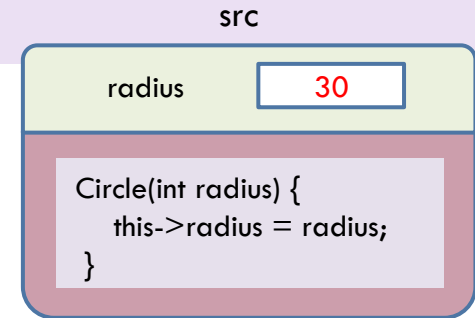
- 복사 생성자(copy constructor)란?
 - ▶ 객체의 복사 생성시 호출되는 특별한 생성자
- 특징
 - ▶ 한 클래스에 **오직 한 개**만 선언 가능
 - ▶ 모양
 - ▶ 클래스에 대한 **참조 매개 변수**를 가지는 독특한 **생성자**
- 복사 생성자 선언

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

자기 클래스에 대한
참조 매개 변수

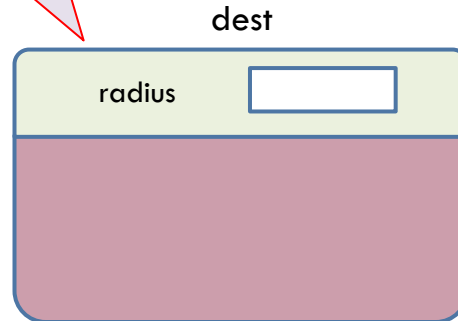
복사 생성 과정

(1) Circle src(30);

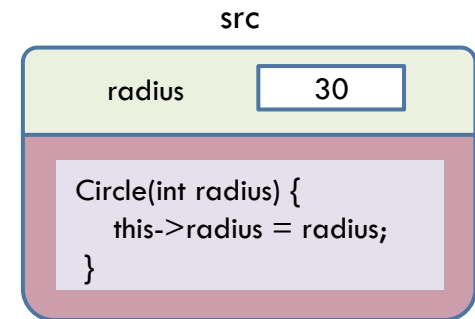


(2) Circle dest(src);

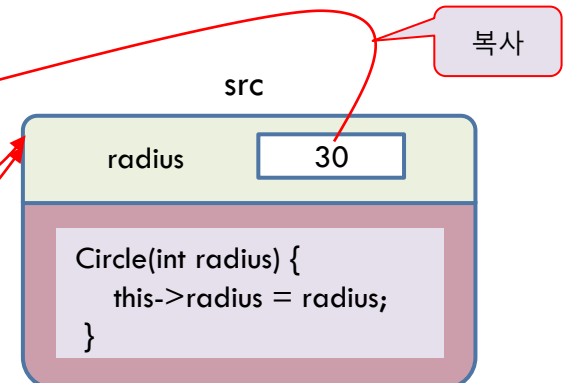
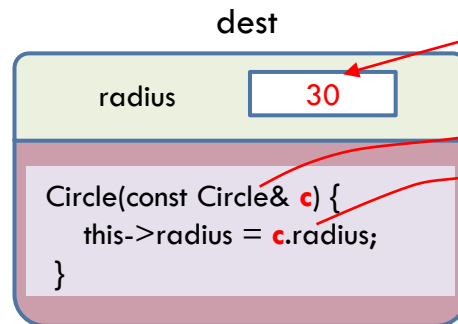
dest 객체
공간 할당



전달



(3) dest 객체의 복사생성자
Circle(const Circle& c) 실행



예제 5-9 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(Circle& c)

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

디폴트 복사 생성자

- 복사 생성자가 선언되어 있지 않는 클래스
 - ▶ 컴파일러가 자동으로 디폴트 복사 생성자를 생성

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

복사 생성자
없음

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

복사 생성자 없는데
컴파일 오류?

```
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;  
}
```

디폴트 복사 생성자는 얇은 복사를 수행한다.

예제 5-10 얇은 복사 생성자를 사용하여 프로그램이 비정상 종료되는 경우-1

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

name 메모리 반환

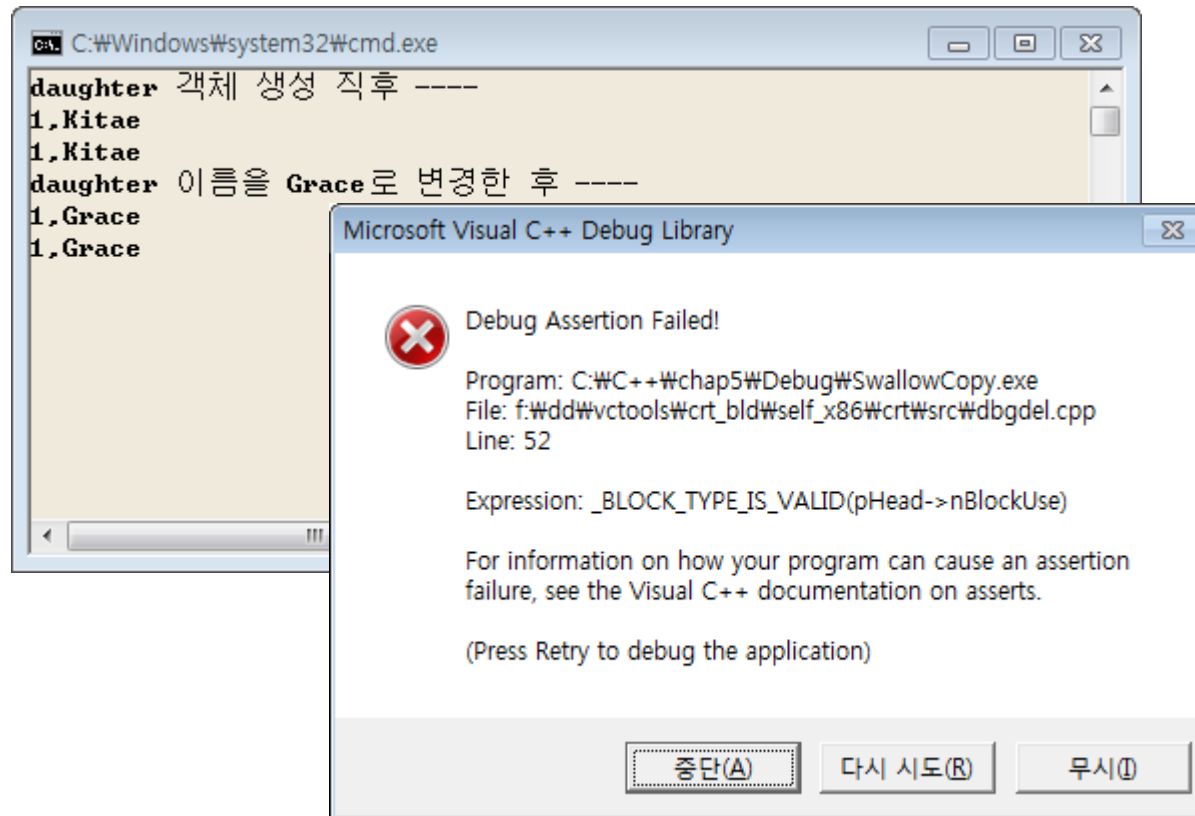
예제 5-10 얇은 복사 생성자를 사용하여 프로그램이 비정상 종료되는 경우-2

```
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 5-10의 실행 결과



예제 5-10의 실행 과정

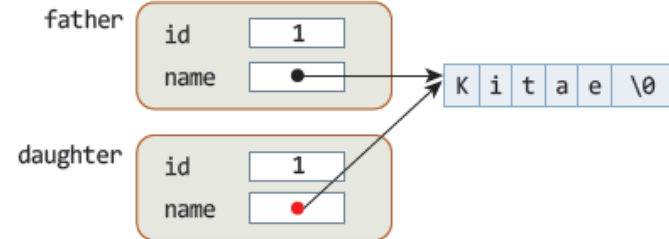
(1) `Person father(1, "Kitae");`

father 객체 생성



(2) `Person daughter(father);`

father를 복사한
daughter 객체 생성



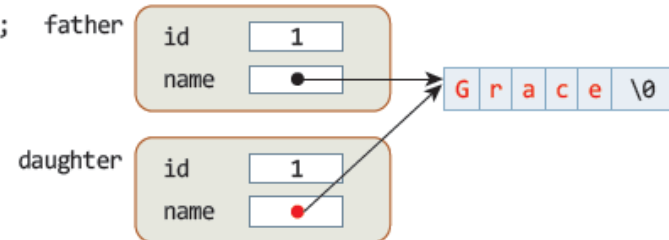
(3) `father.show();`
`daughter.show();`

⇒ 실행 결과

1,Kitae
1,Kitae

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

daughter의 이름
변경

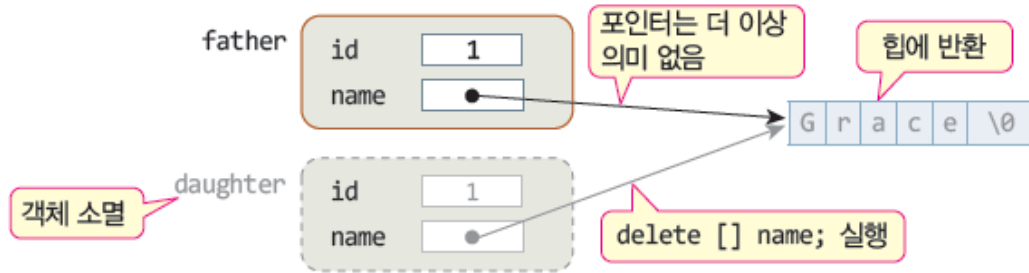


(5) `father.show();`
`daughter.show();`

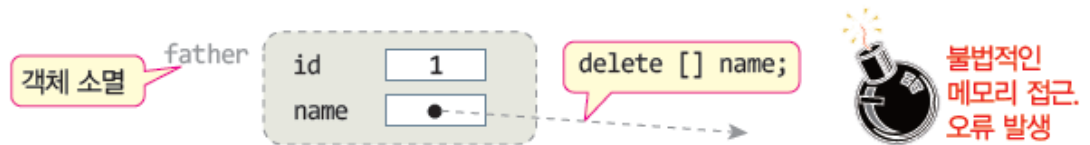
⇒ 실행 결과

1,Grace
1,Grace

(6) daughter 객체 소멸



(7) father 객체 소멸



예제 5-11 깊은 복사 생성자를 가진 정상적 인 Person 클래스

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

class Person { // Person 클래스 선언
    char* name;
    int id;
public:
    Person(int id, char* name); // 생성자
    Person(const Person& person); // 복사 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(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(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

예제 5-11 깊은 복사 생성자를 가진 정상적 인 Person 클래스

```
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 순
으로 소멸

예제 5-11의 실행 결과

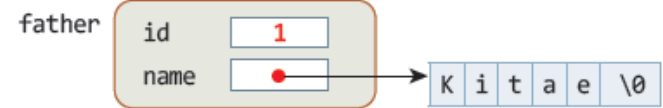
```
C:\Windows\system32\cmd.exe
복사 생성자 실행. 원본 객체의 이름 Kitae
daughter 객체 생성 직후 ----
1,Kitae
1,Kitae
daughter 이름을 Grace로 변경한 후 ----
1,Kitae
1,Grace
계속하려면 아무 키나 누르십시오 . . .
```

복사 생성자에서 출력한 내용

예제 5-11의 실행 과정

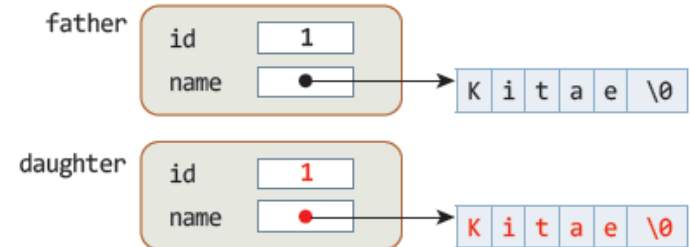
(1) `Person father(1, "Kitae");`

father 객체 생성



(2) `Person daughter(father);`

father를 복사한
daughter 객체 생성



→ 실행 결과

복사 생성자 실행 Kitae

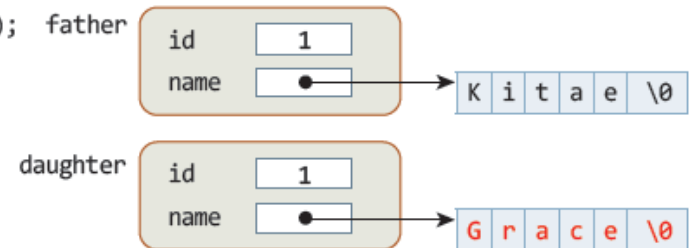
(3) `father.show();`
`daughter.show();`

→ 실행 결과

1,Kitae
1,Kitae

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

daughter의 이름
변경

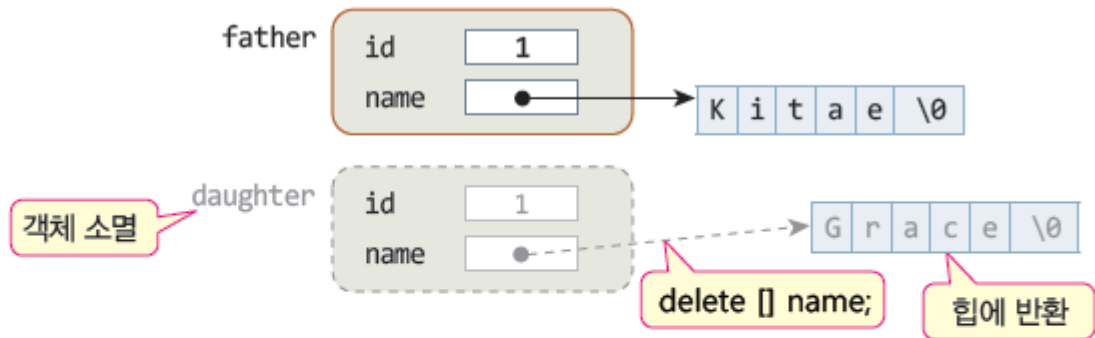


→ 실행 결과

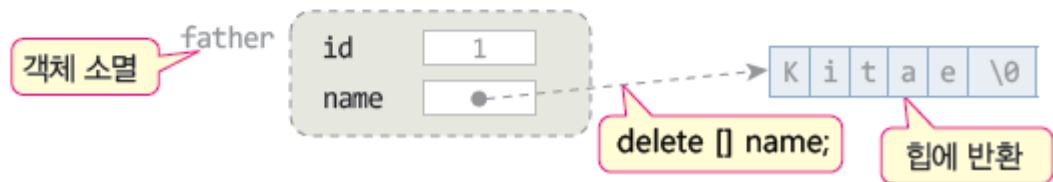
1,Kitae
1,Grace

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

(6) daughter 객체 소멸



(7) father 객체 소멸



묵시적 복사 생성

- 컴파일러가 복사 생성자를 자동으로 호출하는 경우

- 객체로 추기화하여 객체가 생성될 때

```
Person father(1, "Hongkilsoo");  
Person son1 = father; //복사 생성자 호출  
Person son2(father);  //복사 생성자 호출
```

- “값에 의한 호출”로 객체가 전달될 때

```
void changeName(Person person)  
{person.changeName("Hongkildong"); }
```

```
changeName ( son ); //함수 호출 시 객체를 매개변수로 전달
```

- 함수가 객체를 리턴할 때

```
Person getPerson() {  
    Person tmp(2, "HongYounghee");  
    return tmp;  
}
```

```
getPerson(); //함수 호출 결과 반환된 결과를 m에 치환
```

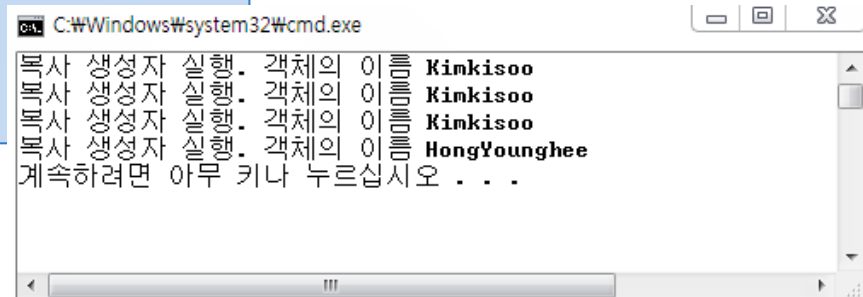
예제 5-12 묵시적 복사 생성에 의해 복사 생성자가 자동 호출되는 경우

```
void chageName( Person person){
    person.changeName("Hongkildong");
}

Person getPerson() {
    Person tmp(2, "HongYounghee");
    return tmp;
}

int main() {
    Person father(1, "Kimkisoo");
    Person son1 = father; //복사 생성자 호출
    Person son2(father);  //복사 생성자 호출
    chageName( son2 ); //복사생성자 호출
    getPerson(); //복사생성자 호출

    return 0;
}
```

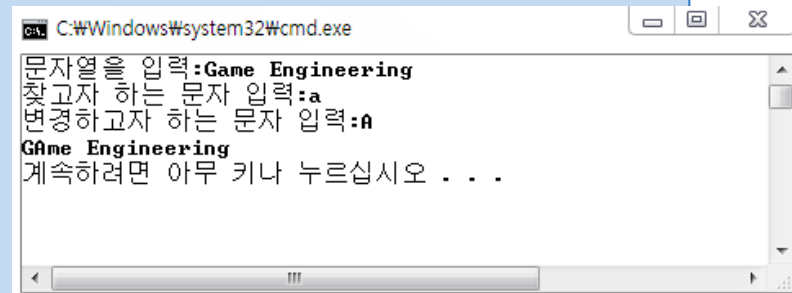


```
C:\Windows\system32\cmd.exe
복사 생성자 실행. 객체의 이름 Kimkisoo
복사 생성자 실행. 객체의 이름 Kimkisoo
복사 생성자 실행. 객체의 이름 Kimkisoo
복사 생성자 실행. 객체의 이름 HongYounghee
계속하려면 아무 키나 누르십시오 . . .
```


연습문제 1 – find함수 구현

- find() 함수의 원형은 다음과 같다. 문자열 a에서 문자 c가 처음 나오는 곳을 찾아, 문자 c가 있는 공간에 대한 참조를 리턴 한다. 만약 문자 c를 찾을 수 없다면 success 참조 매개 변수에 false를 설정한다. 물론 찾게 되면 success에 true를 설정한다. Main()함수가 수행되도록 find()를 작성하시오.

```
int main() {  
    char s[N];  
    bool b = false;  
    char ch, newch;  
    cout << "문자열을 입력:";  
    cin.getline( s, N, '\n');  
    cout << "찾고자 하는 문자 입력:";  
    cin >> ch;  
    char& loc = find(s, ch, b);  
    if(b == false) {  
        cout << "M을 발견할 수 없다" << endl;  
        return 0;  
    }  
    cout << "변경하고자 하는 문자 입력:";  
    cin >> newch;  
    loc = newch ;  
    cout << s << endl;  
}
```



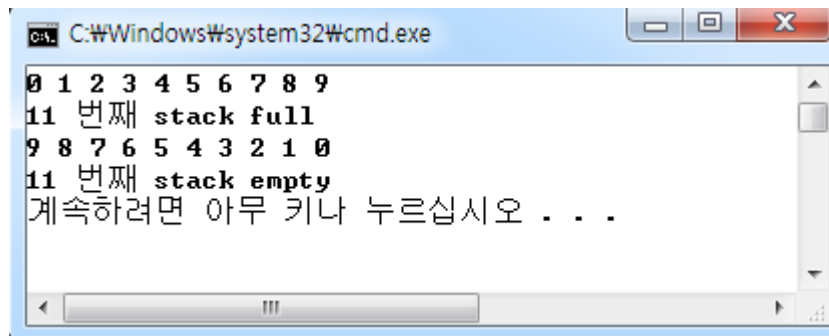
```
C:\Windows\system32\cmd.exe  
문자열을 입력:Game Engineering  
찾고자 하는 문자 입력:a  
변경하고자 하는 문자 입력:A  
Game Engineering  
계속하려면 아무 키나 누르십시오 . . .
```

연습문제 2 -MyIntStack 클래스

- 다음과 같이 선언된 정수를 저장하는 스택 클래스 MyIntStack을 구현하시오. MyIntStack 스택에 저장할 수 있는 정수의 최대 개수는 10이다.

```
class MyIntStack {
    int p[10];
    int tos; // 스택의 꼭대기를 가리키는 인덱스
public:
    MyIntStack();
    bool push(int n); // 정수 n 푸시. 꽉 차 있으면 false, 아니면 true 리턴
    bool pop(int &n); // 팝하여 n에 저장.스택이 비어 있으면 false, 아니면 true 리턴
};
```

- MyIntStack클래스를 사용하는 main()코드와 실행결과는 다음과 같다.



```
C:\Windows\system32\cmd.exe
0 1 2 3 4 5 6 7 8 9
11 번째 stack full
9 8 7 6 5 4 3 2 1 0
11 번째 stack empty
계속하려면 아무 키나 누르십시오 . . .
```

연습문제 2 -MyIntStack 클래스

- MyIntStack클래스를 사용하는 main()코드와 실행결과는 다음과 같다.

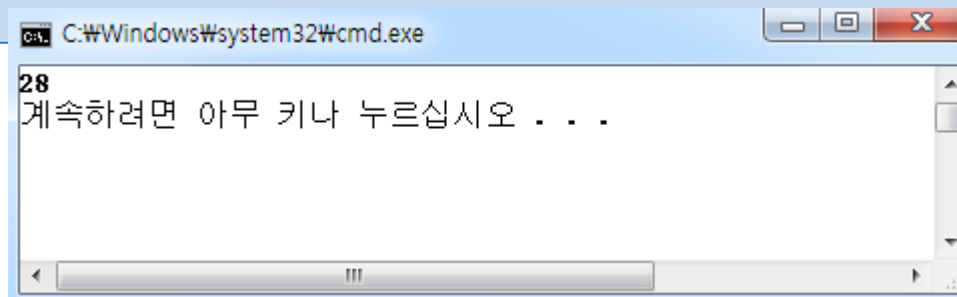
```
int main() {  
    MyIntStack a;  
    for(int i=0; i<11; i++) { // 11개를 푸시한다.  
        if(a.push(i)) cout << i << ' '; // 푸시된 값 에코  
        else cout << endl << i+1 << " 번째 stack full" << endl;  
    }  
    int n;  
    for(int i=0; i<11; i++) { // 11개를 팝한다.  
        if(a.pop(n)) cout << n << ' '; // 팝 한 값 출력  
        else cout << endl << i+1 << " 번째 stack empty";  
    }  
    cout << endl;  
}
```

연습문제 3 – Accumulator 클래스

- 클래스 Accumulator는 add()함수를 통해 계속 값을 누적하는 클래스로서, 다음과 같이 선언된다. Accumulator 클래스를 구현하시오.

```
class Accumulator {  
    int value;  
public:  
    Accumulator(int value) { this->value = value; }  
    Accumulator& add(int n);  
    int get() { return value; }  
};
```

```
int main() {  
    Accumulator acc(10);  
    acc.add(5).add(6).add(7); // acc의 value 멤버가 28이 된다.  
    cout << acc.get() << endl; // 28 출력  
}
```



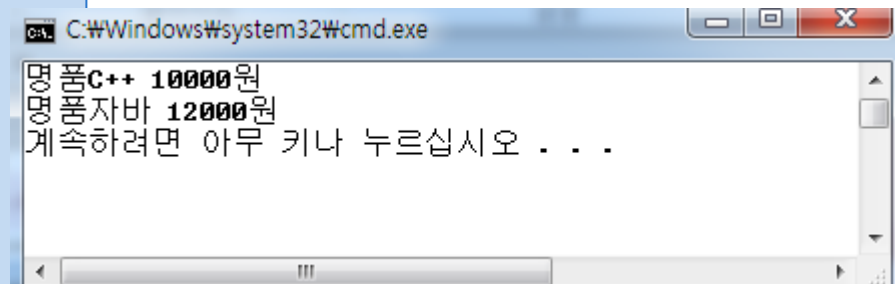
연습문제 4 – Book 클래스

- 책의 이름과 가격을 저장하는 다음 Book 클래스에 대해 물음에 답하여라.

```
class Book {  
    char *title; // 제목 문자열  
    int price; // 가격  
  
public:  
    Book(char* title, int price);  
    ~Book();  
    void set(char* title, int price);  
    void show() { cout << title << ' ' << price << "원" << endl; }  
};
```

- 1) Book 클래스의 생성자, 소멸자, set() 함수를 구현하라.
- 2) 위의 클래스에는 복사 생성자가 정의되어 있지 않으므로 디폴트 복사 생성자가 생성되고 디폴트 생성자만 있을 때 아래 main() 함수는 실행 오류를 발생시킨다. 실행 오류가 발생하지 않도록 깊은 복사 생성자를 추가해 보자.

```
int main() {  
    Book cpp("명품C++", 10000);  
    Book java = cpp;  
    java.set("명품자바", 12000);  
    cpp.show();  
    java.show();  
}
```



```
C:\Windows\system32\cmd.exe  
명품C++ 10000원  
명품자바 12000원  
계속하려면 아무 키나 누르십시오 . . .
```