

05

함수와 참조, 복사생성자

■ 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

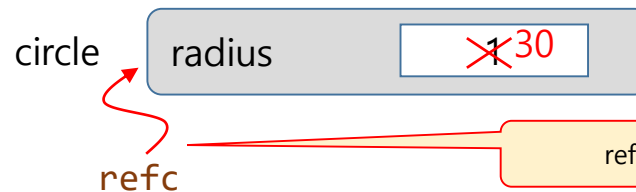
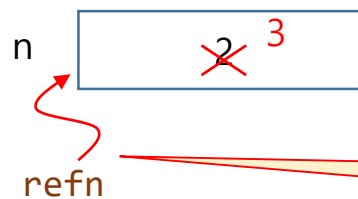
이미 존재하는
변수 n

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

이미 존재하는
클래스 객체 circle

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

refc->setRadius(30);으로 하면 안 됨



예제 5-3 기본 타입 변수에 대한 참조

4

```
#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

```
#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

- 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 실인자 변수를 참조함
 - 참조매개 변수의 이름만 생기고 공간이 생기지 않음
 - 참조 매개 변수는 실인자 변수 공간 공유
 - 참조 매개 변수에 대한 조작은 실인자 변수 조작 효과

참조에 의한 호출 사례

7

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

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

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

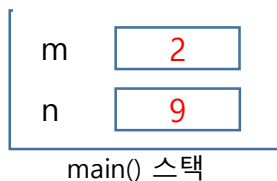
int main() {
    int m=2, n=9;
    swap(m, n);
    cout << m << ' ' << n;
}
```

참조 매개 변수 a, b

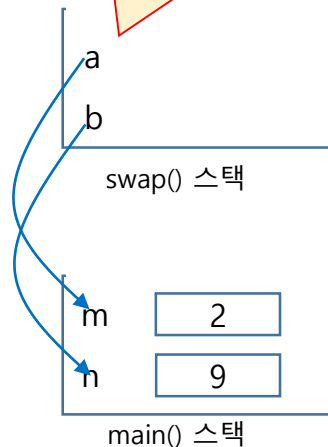
참조 매개 변수를 보통 변수처럼 사용

함수가 호출되면 m, n에 대한 참조 변수 a, b가 생긴다.

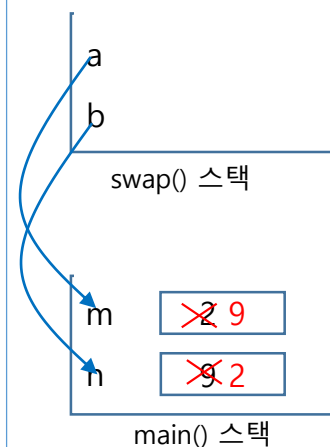
9 2



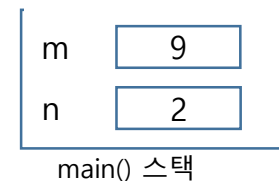
(1) swap() 호출 전



(2) swap() 호출 직후



(3) swap() 실행



(4) swap() 리턴 후

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

m, n이 변경 됨

■ 다음 코드에 어떤 문제가 있을까?

- average() 함수의 작동
 - 계산에 오류가 있으면 0 리턴, 아니면 평균 리턴
- 만일 average()가 리턴 한 값이 0이라면?
 - 평균이 0인 거야? 아니면 오류가 발생한 거야?

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

호출

호출

예제 5-5에서 해결

```
int x[]={1,2,3,4};  
int avg = average(x, 4);
```

// avg는 2

흠, 평균이 2군. 알았어!

```
int x[]={1,2,3,4};  
int avg = average(x, -1);
```

// avg는 0

평균이 0인 거야,
아니면 오류가 난 거야?

예제 5-5 참조 매개 변수로 평균 리턴하기

9

참조 매개 변수를 통해 평균을 리턴하고
리턴문을 통해서도 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 avg에 평균 값 전달

main의 avg공간에 (average의) avg의 별명이
붙으면서 똑같은 메모리 공간에 접근
int **avg**; (main) → avarage(x,6,**avg**)
int &avg = avg;

```
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;
}
```

avg에 평균이 넘어오고,
average()는 true 리턴

avg의 값은 의미없고,
average()는 false 리턴

평균은 2
매개 변수 오류

예제 5-6 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

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

waffle 객체 생성

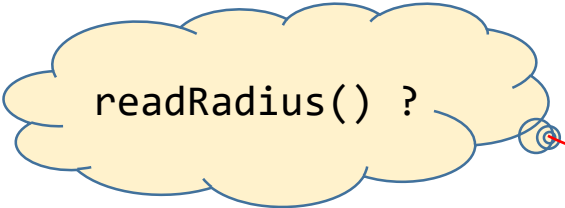
waffle 객체 소멸

```
#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

11

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



readRadius() ?

```
#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

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

■ 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

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
```

(a) 문자 값을 리턴하는 get()

char 타입의 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
```

(b) char 타입의 참조(공간)을 리턴하는 find()

find()가 리턴한 공간에
'b' 문자 저장

예제 5-8 간단한 참조 리턴 사례

15

```
#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 'm' 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;
```

ref는 name[2] 참조

Mike
Sike
Site

(1) `char name[] = "Mike";`

M	i	k	e	\0
---	---	---	---	----

 name

(2) `return s[index];`

공간에 대한
참조, 즉 이름
의 이름 리턴

M	i	k	e	\0
---	---	---	---	----

s[index]

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

S	i	k	e	\0
---	---	---	---	----

(4) `ref = 't';`

S	i	t	e	\0
---	---	---	---	----

■ 인자 전달 방식

- 값에 의한 호출, call by value
 - 함수가 호출되면 매개 변수가 스택에 생성됨
 - 호출하는 코드에서 값을 넘겨줌
 - 호출하는 코드에서 넘어온 값이 매개 변수에 복사됨
- 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



(a) 값에 의한 호출



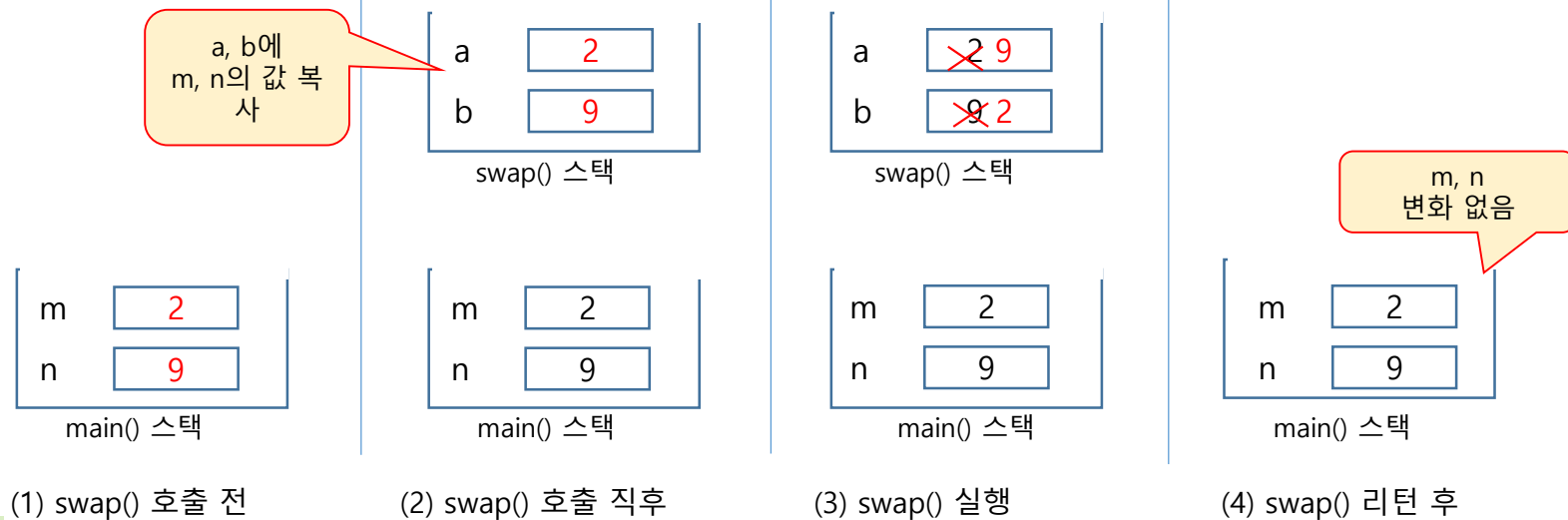
(b) 주소에 의한 호출

```
#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;
}
```



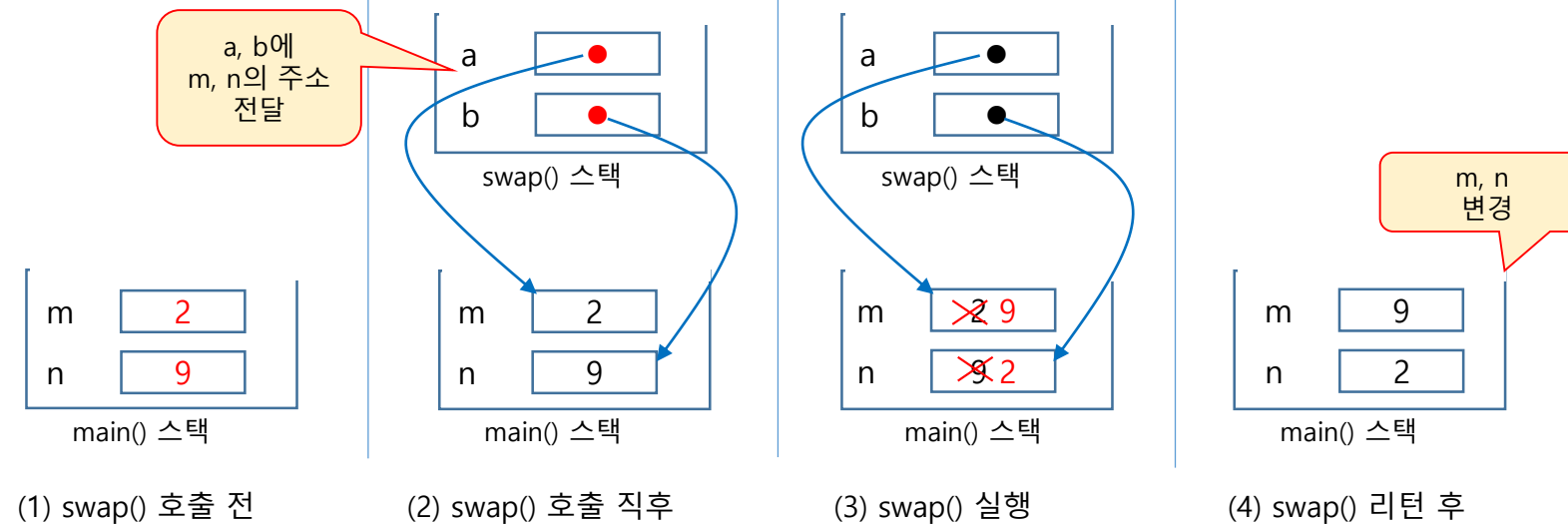
값에 의한 호출

```
#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;
}
```



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**
 - 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

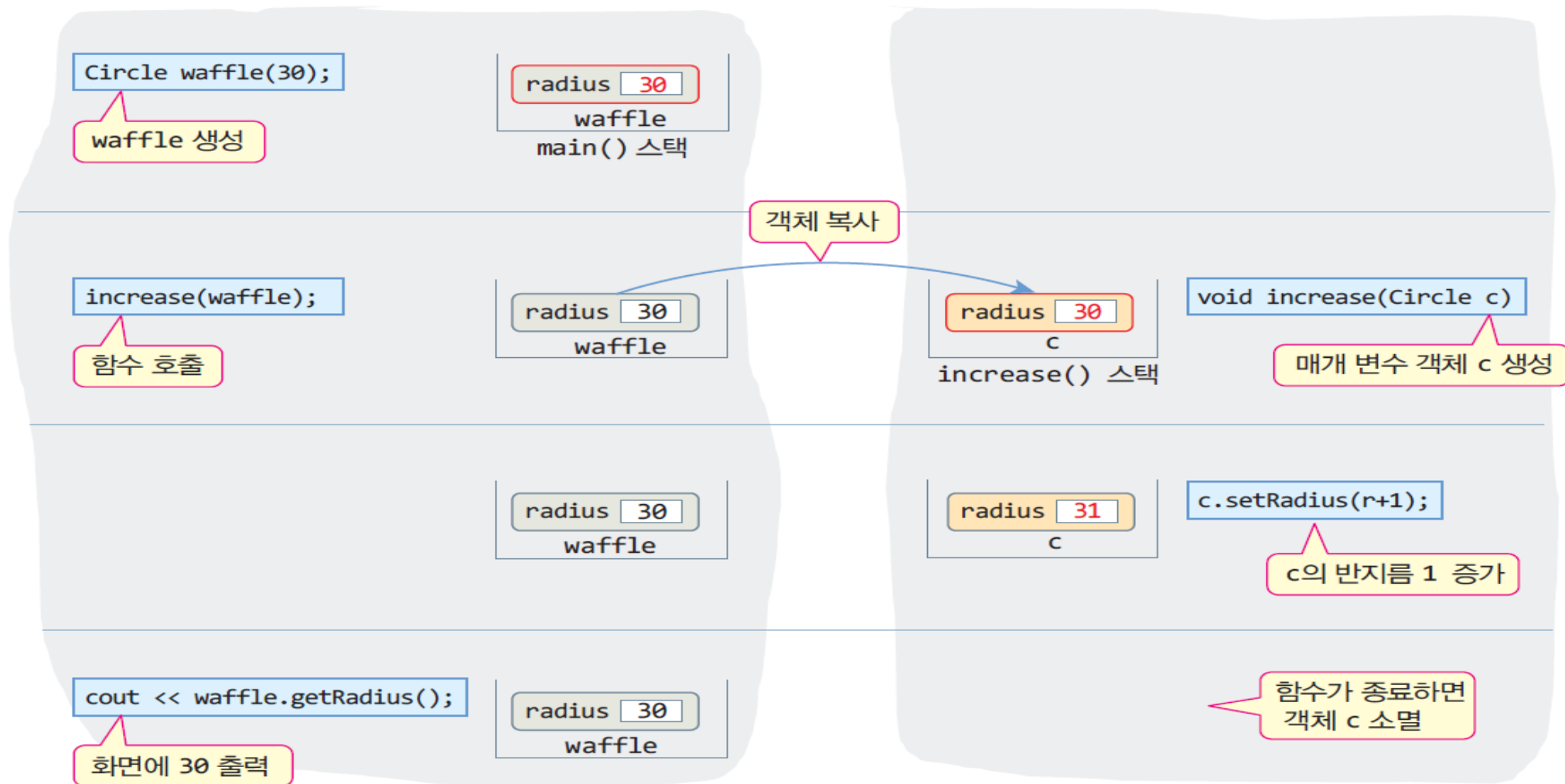
⇒ 실행 결과

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 c
에 복사

waffle 생성

생성자 실행 radius = 30
소멸자 실행 radius = 31
30

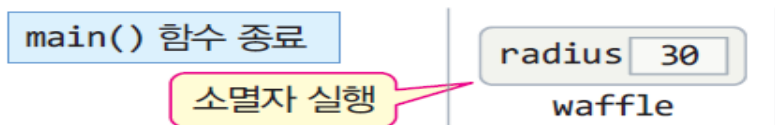
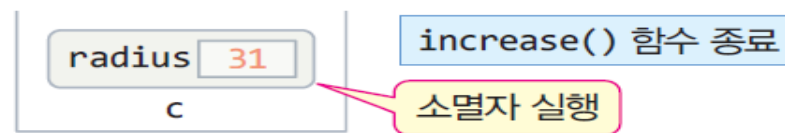
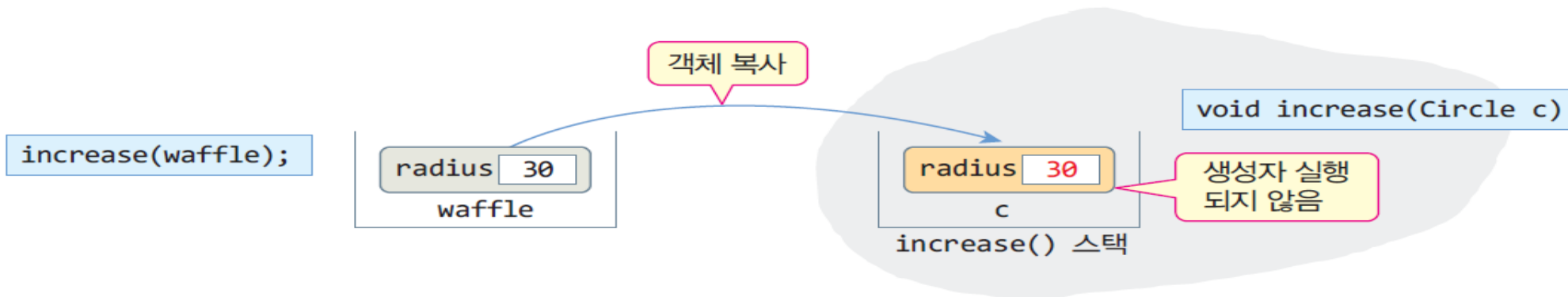
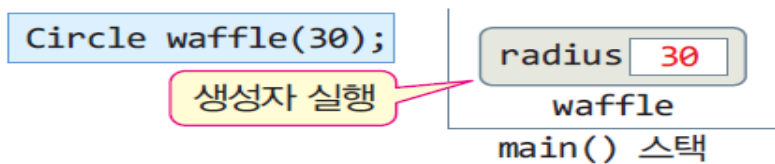
waffle 소멸

소멸자 실행 radius = 30

c의 생성자
실행되지 않았음

c 소멸

'값에 의한 호출' 시에 생성자와 소멸자의 비대칭 실행



■ 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

increase(&waffle);

함수호출

radius 30

waffle

waffle의 주소가
p에 전달

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

- 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
```

```
#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



메뚜기는 유재석의 별명

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



참조 활용

- 참조 변수
- 참조에 의한 호출
- 참조 리턴

■ 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

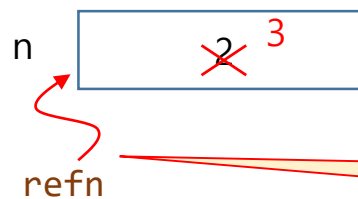
이미 존재하는
변수 n

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

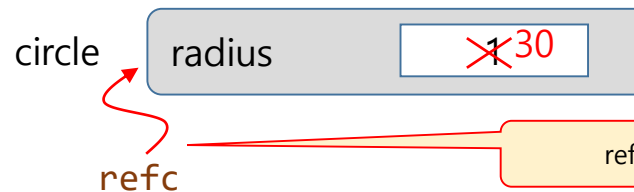
이미 존재하는
클래스 객체 circle

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

refc->setRadius(30);으로 하면 안 됨



refn는 n에 대한 별명



refc는 circle 객체에 대한 별명

예제 5-3 기본 타입 변수에 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

```
#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

- 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 실인자 변수를 참조함
 - 참조매개 변수의 이름만 생기고 공간이 생기지 않음
 - 참조 매개 변수는 실인자 변수 공간 공유
 - 참조 매개 변수에 대한 조작은 실인자 변수 조작 효과


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

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

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

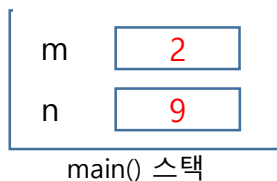
int main() {
    int m=2, n=9;
    swap(m, n);
    cout << m << ' ' << n;
}
```

참조 매개 변수 a, b

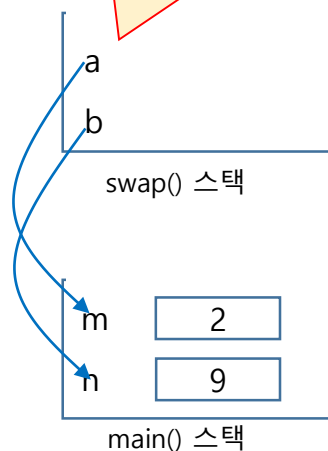
참조 매개 변수를 보통 변수처럼 사용

함수가 호출되면 m, n에 대한 참조 변수 a, b가 생긴다.

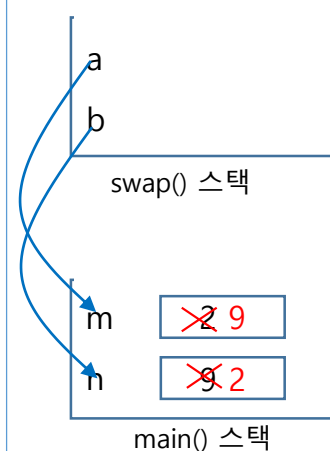
9 2



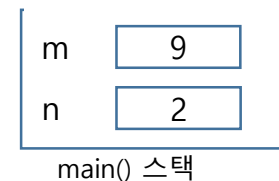
(1) swap() 호출 전



(2) swap() 호출 직후



(3) swap() 실행



(4) swap() 리턴 후

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

m, n이 변경 됨

■ 다음 코드에 어떤 문제가 있을까?

- average() 함수의 작동
 - 계산에 오류가 있으면 0 리턴, 아니면 평균 리턴
- 만일 average()가 리턴 한 값이 0이라면?
 - 평균이 0인 거야? 아니면 오류가 발생한 거야?

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

호출

호출

예제 5-5에서 해결

```
int x[]={1,2,3,4};  
int avg = average(x, 4);
```

// avg는 2

흠, 평균이 2군. 알았어!

```
int x[]={1,2,3,4};  
int avg = average(x, -1);
```

// avg는 0

평균이 0인 거야,
아니면 오류가 난 거야?

예제 5-5 참조 매개 변수로 평균 리턴하기

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 avg에 평균 값 전달

main의 avg공간에 (average의) avg의 별명이
붙으면서 똑같은 메모리 공간에 접근
int **avg**; (main) → avarage(x,6,**avg**)
int &avg = avg;

```
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;
}
```

avg에 평균이 넘어오고,
average()는 true 리턴

avg의 값은 의미없고,
average()는 false 리턴

평균은 2
매개 변수 오류

예제 5-6 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

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

waffle 객체 생성

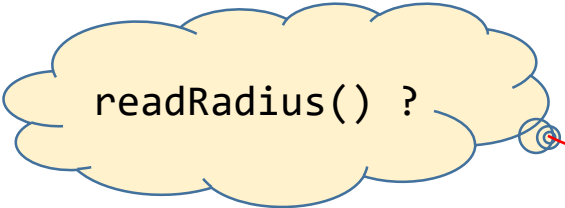
waffle 객체 소멸

```
#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

37

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



readRadius() ?

```
#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

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

■ 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

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
```

(a) 문자 값을 리턴하는 get()

char 타입의 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
```

(b) char 타입의 참조(공간)을 리턴하는 find()

find()가 리턴한 공간에
'b' 문자 저장

예제 5-8 간단한 참조 리턴 사례

41

```
#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 'm' 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;
```

ref는 name[2] 참조

Mike
Sike
Site

(1) `char name[] = "Mike";`

M	i	k	e	\0
---	---	---	---	----

 name

(2) `return s[index];`

공간에 대한
참조, 즉 이름
의 이름 리턴

M	i	k	e	\0
---	---	---	---	----

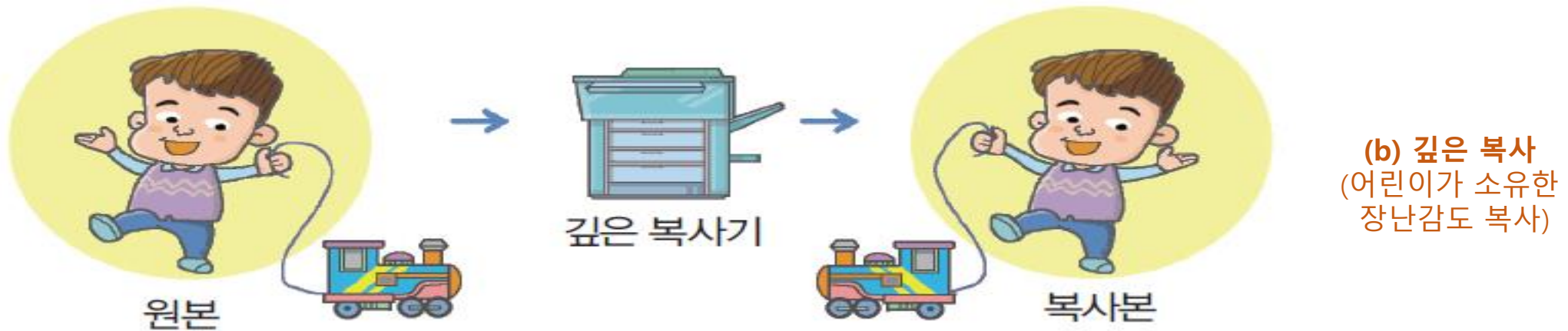
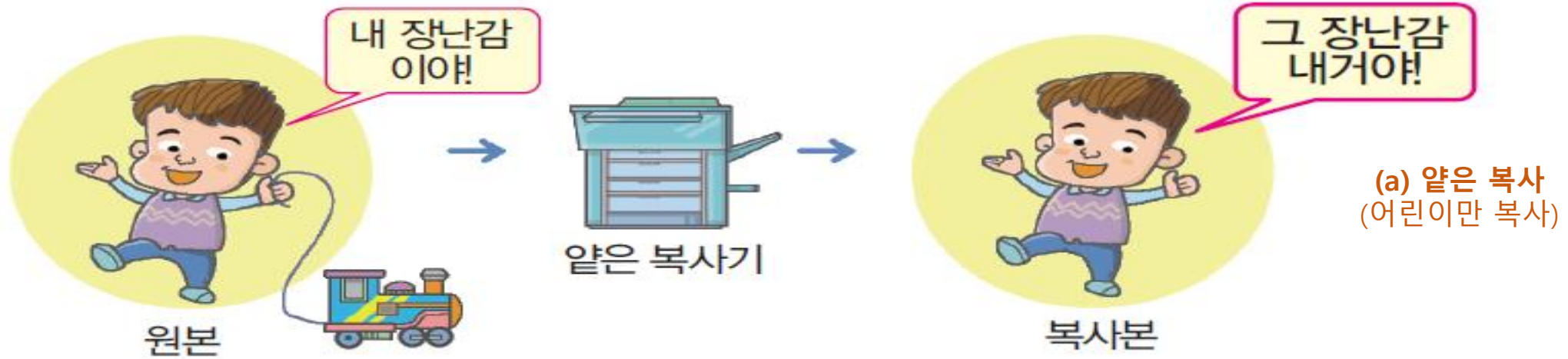
s[index]

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

S	i	k	e	\0
---	---	---	---	----

(4) `ref = 't';`

S	i	t	e	\0
---	---	---	---	----



■ 얇은 복사(shallow copy)

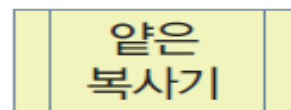
- 객체 복사 시, 객체의 멤버를 1:1로 복사
- 객체의 멤버 변수에 동적 메모리가 할당된 경우
 - 사본은 원본 객체가 할당 받은 메모리를 공유하는 문제 발생

■ 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

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

Person 타입 객체, 원본



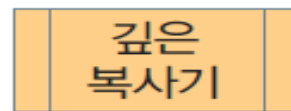
복사본 객체



(a) 얕은 복사

name 포인터가 복사되었기 때문에
메모리 공유! - 문제 유발

Person 타입 객체, 원본



복사본 객체



(b) 깊은 복사

name 포인터의 메모리도
복사되었음

■ 복사 생성자(copy constructor)란?

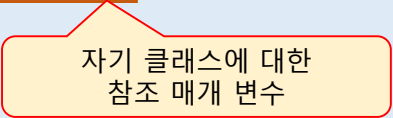
- 객체의 복사 생성시 호출되는 특별한 생성자

■ 특징

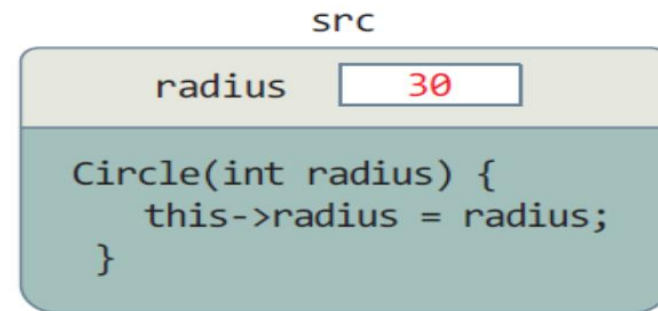
- 한 클래스에 오직 한 개만 선언 가능
- 복사 생성자는 보통 생성자와 클래스 내에 중복 선언 가능
- 모양
 - 클래스에 대한 참조 매개 변수를 가지는 독특한 생성자

■ 복사 생성자 선언

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

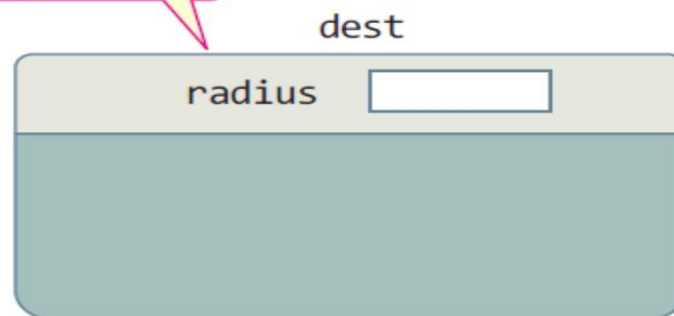


(1) `Circle src(30);`

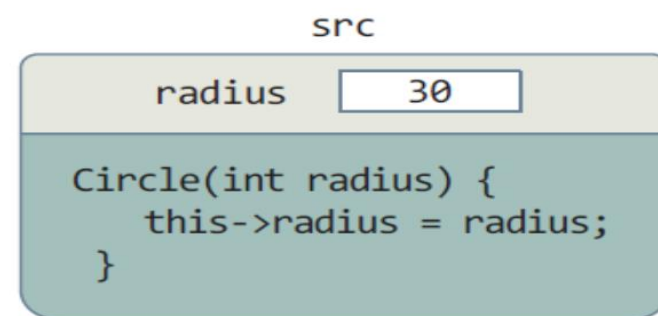


(2) `Circle dest(src);`

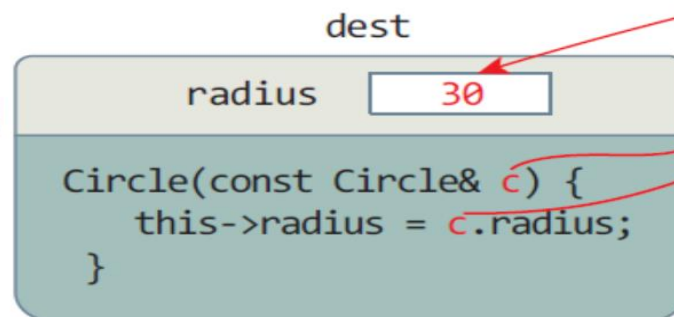
dest 객체
공간 할당



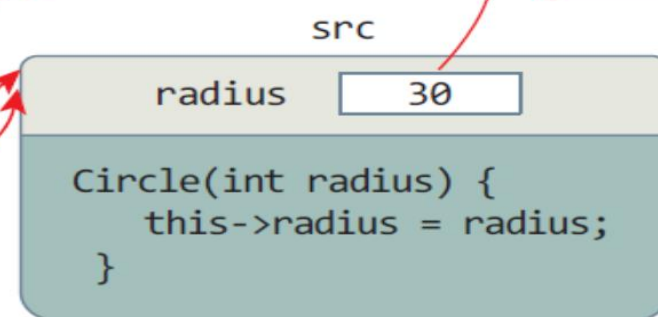
전달



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



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 생성자가 선언되어 있지 않는 클래스

- 컴파일러는 자동으로 디폴트 복사 생성자 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(const Circle&) 호출
```

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

디폴트 복사 생성자


```
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;  
}
```

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

50

```
#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

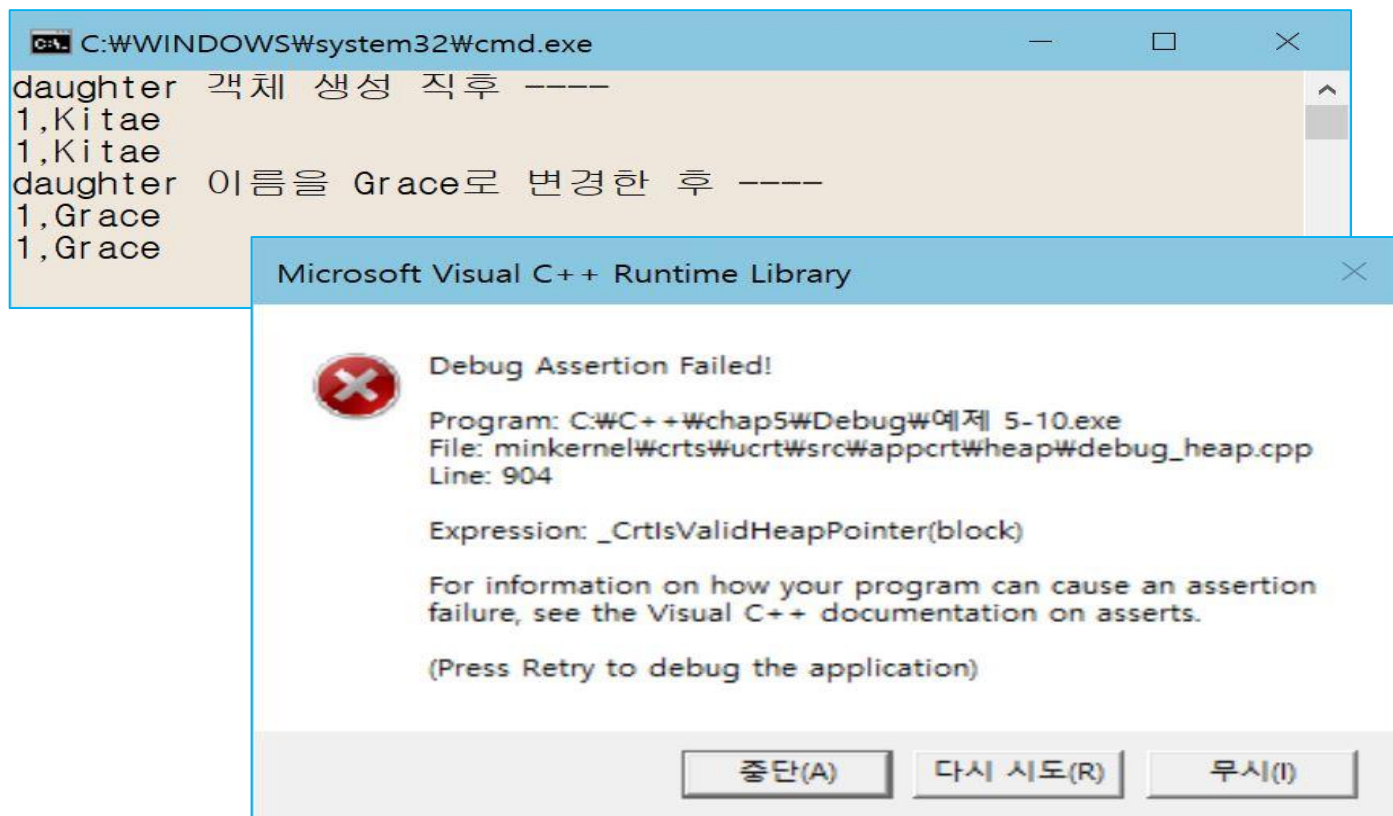
    daughter.changeName("Grace");        // (4) daughter의 이름을 "Grace"로 변경
    cout << "daughter 이름을 Grace로 변경한 후 ----" << endl;
    father.show();                       // (5) father 객체 출력
    daughter.show();                     // (5) daughter 객체 출력

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

```

컴파일러가 삽입한
디폴트 복사 생성자 호출

daughter, father 순으로 소멸.
father가 소멸할 때, 프로그램 비정상 종
료됨



예제 5-10의 실행 과정

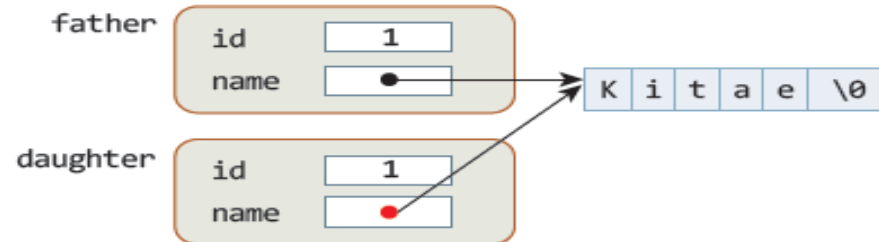
(1) `Person father(1, "Kitae");`

father 객체 생성



(2) `Person daughter(father);`

father를 복사한
daughter 객체 생성



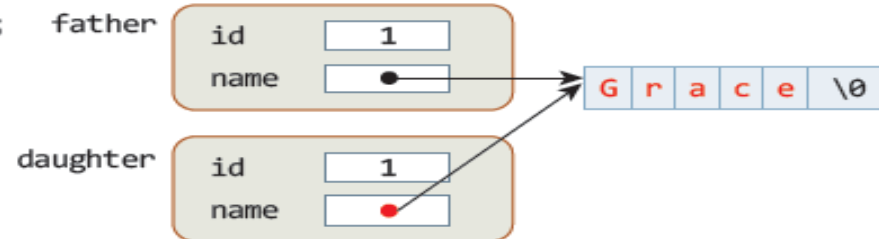
(3) `father.show();`
`daughter.show();`

⇒ 실행 결과

1,Kitae
1,Kitae

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

daughter의 이름
변경

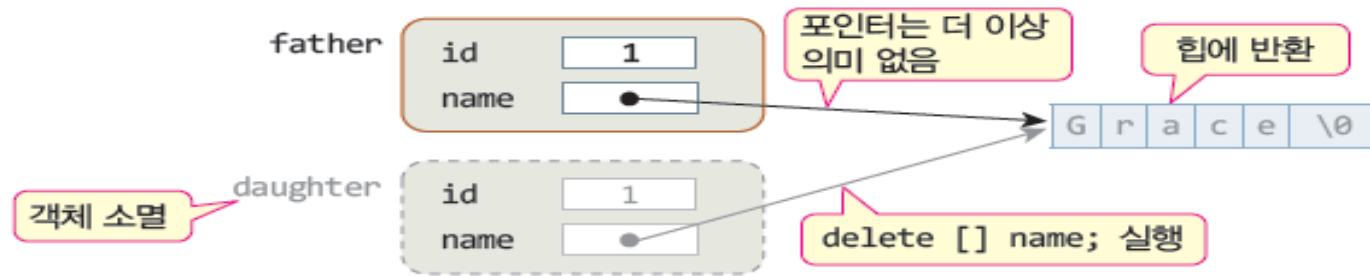


(5) `father.show();`
`daughter.show();`

⇒ 실행 결과

1,Grace
1,Grace

(6) daughter 객체 소멸



(7) father 객체 소멸



예제 5-11 깊은 복사 생성자를 가진 정상적인 Person 클래스

55

```
#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



```
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

58

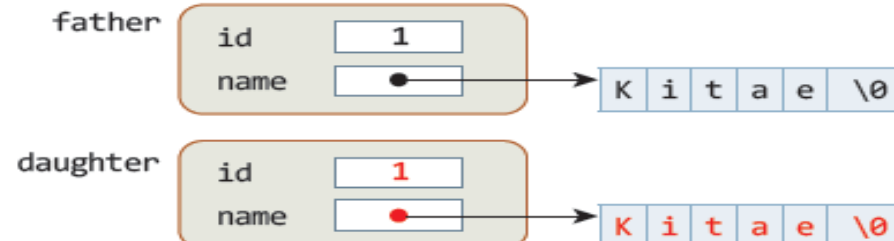
(1) `Person father(1, "Kitae");`

father 객체 생성



(2) `Person daughter(father);`

father를 복사한
daughter 객체 생성



⇒ 실행 결과

복사 생성자 실행 Kitae

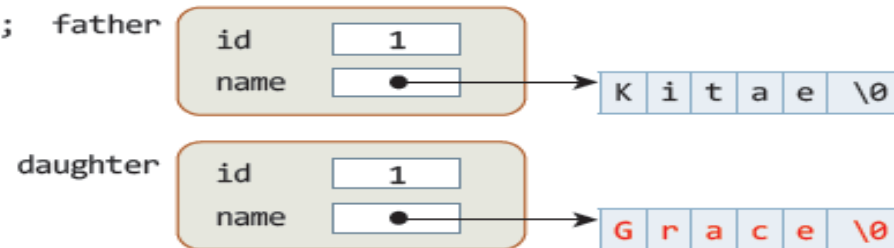
(3) `father.show();`
`daughter.show();`

⇒ 실행 결과

1,Kitae
1,Kitae

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

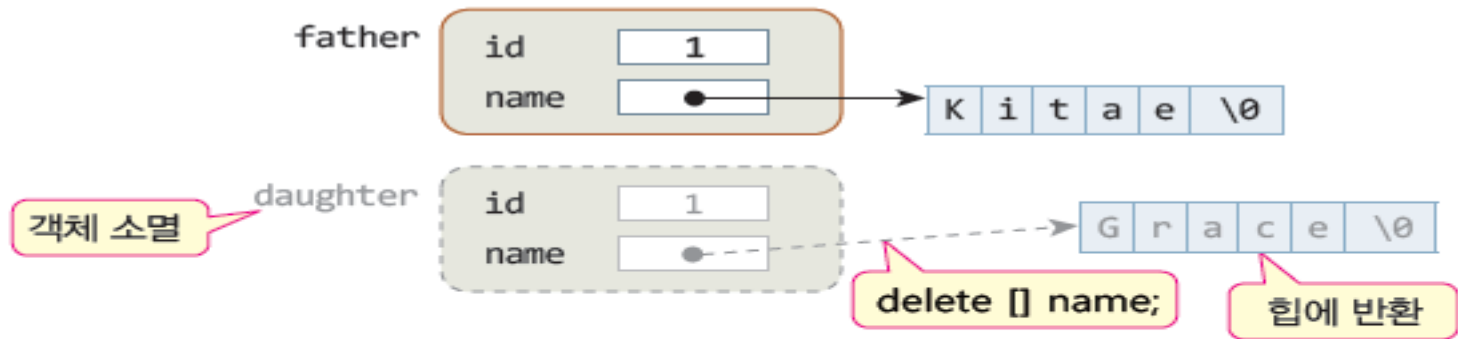
daughter의 이름
변경



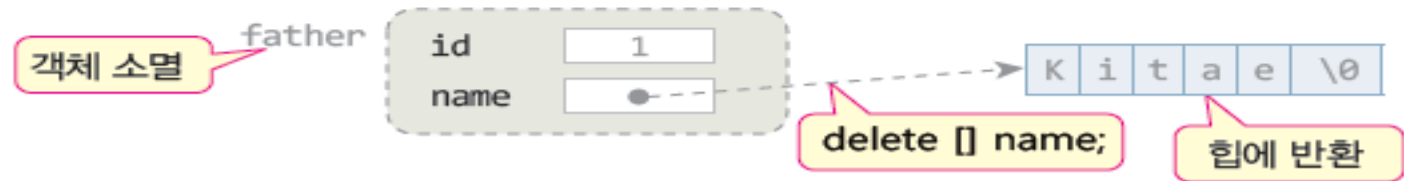
⇒ 실행 결과

1,Kitae
1,Grace

(6) daughter 객체 소멸



(7) father 객체 소멸



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

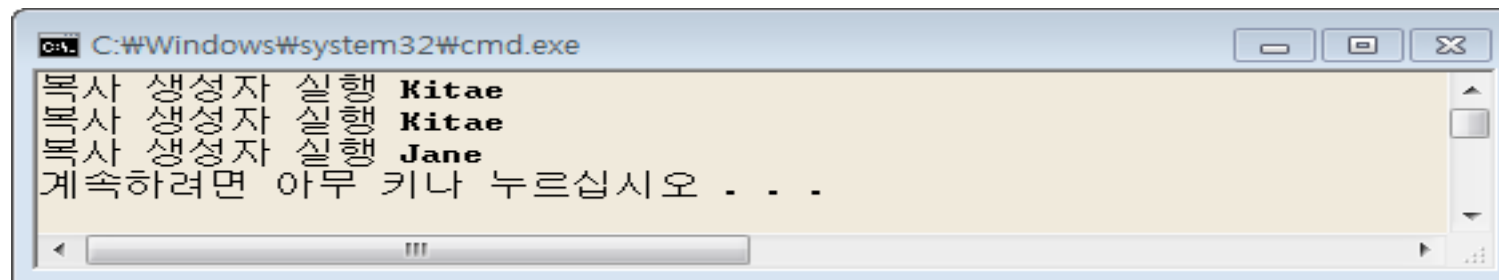
2. '값에 의한 호출'로 객체가 전달될 때.
person 객체의 복사 생성자 호출

```
Person g() {  
    Person mother(2, "Jane");  
    return mother;  
}
```

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

```
int main() {  
    Person father(1, "Kitae");  
    Person son = father;  
    f(father);  
    g();  
}
```

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



```
C:\Windows\system32\cmd.exe  
복사 생성자 실행 Kitae  
복사 생성자 실행 Kitae  
복사 생성자 실행 Jane  
계속하려면 아무 키나 누르십시오 . . .
```