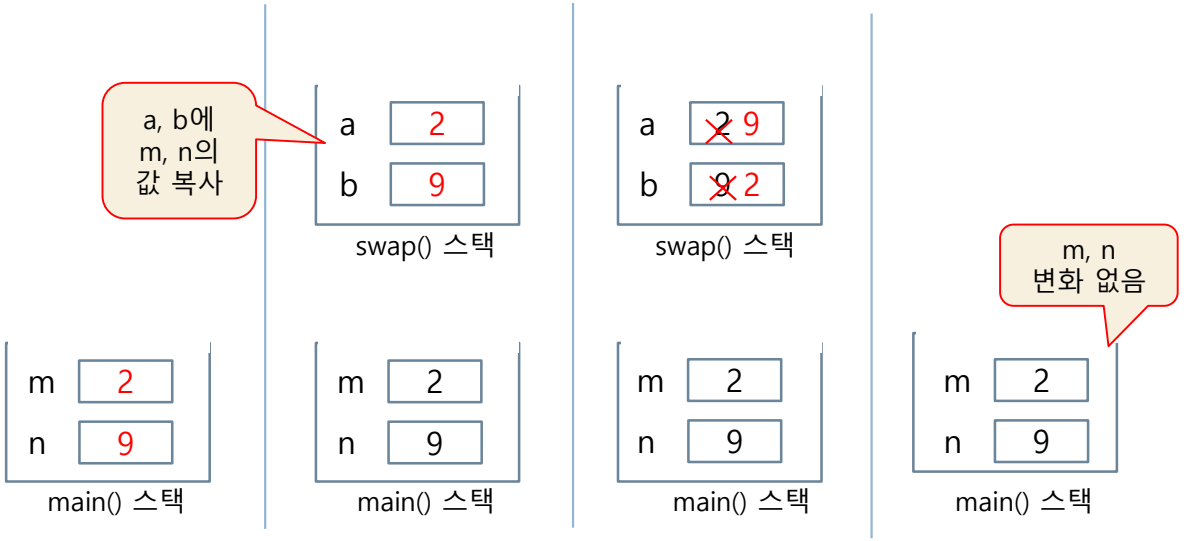
Ch11 함수와 참조

```
#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;
}
```

2 9

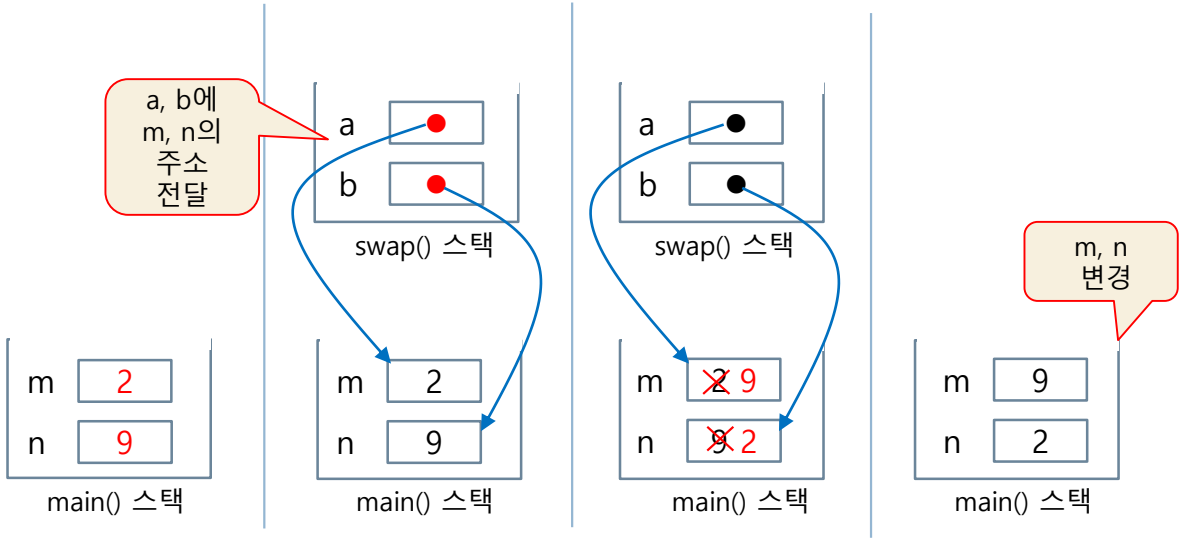


(1) swap() 호출 전 (2) swap() 호출 직후 (3) swap() 실행 (4) swap() 리턴 후

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;
}
```

9 2

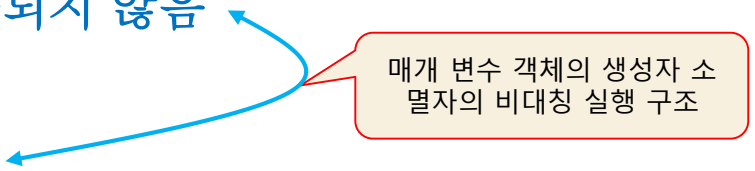


(1) swap() 호출 전 (2) swap() 호출 직후 (3) swap() 실행 (4) swap() 리턴 후

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**



- *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

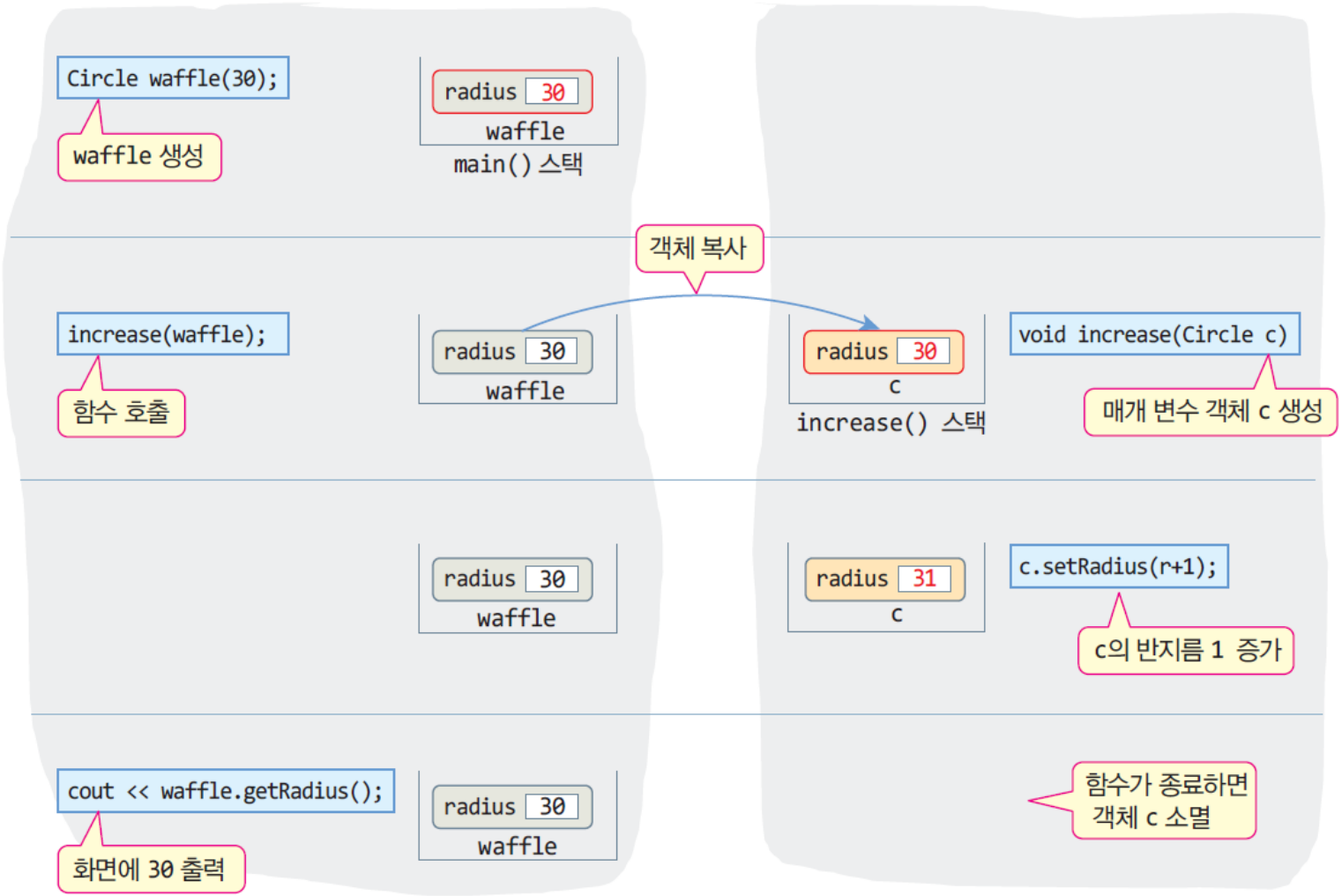
실행 결과

```
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);
}
```



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();
    int getRadius();
    void setRadius(int radius);
};

Circle::Circle() {
    radius = 1;
    cout << "Circle() radius = " << radius << endl;
}
Circle::Circle(int radius) {
    this->radius = radius;
    cout << "Circle(int radius) radius = "  << radius << endl;
}
double Circle::getArea() {
    return 3.14 * radius * radius;
}
int Circle::getRadius() {
    return radius;
}
void Circle::setRadius(int radius) {
    this->radius = radius;
}
Circle::~Circle() {
    cout << "~Circle() 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

```
Circle(int radius) radius = 30
~Circle() radius = 31
30
~Circle() radius = 30
```

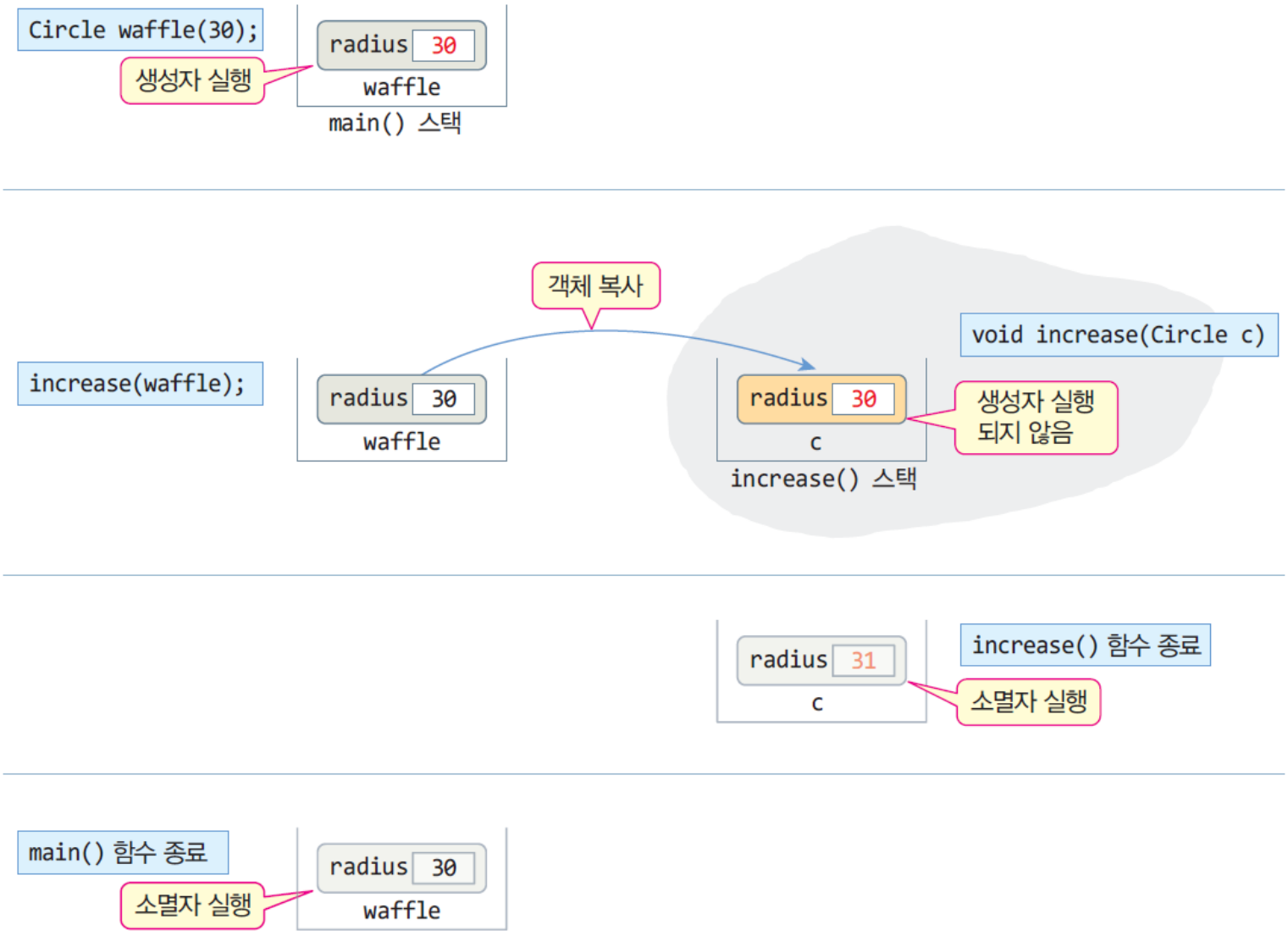
waffle 생성

c의 생성자 실행되지 않았음

waffle 소멸

c 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


```
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

함수가 종료하면 포인터 p 소멸

cout << waffle.getRadius();

31이 화면에 출력됨

radius 31

waffle

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
  - 객체의 복사본 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
```

예제 5-2 객체 리턴


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

class Circle {
    int radius;
public:
    Circle();
    Circle(int radius);
    void setRadius(int radius);
    double getArea();
};

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

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

참조에 의한 호출 사례

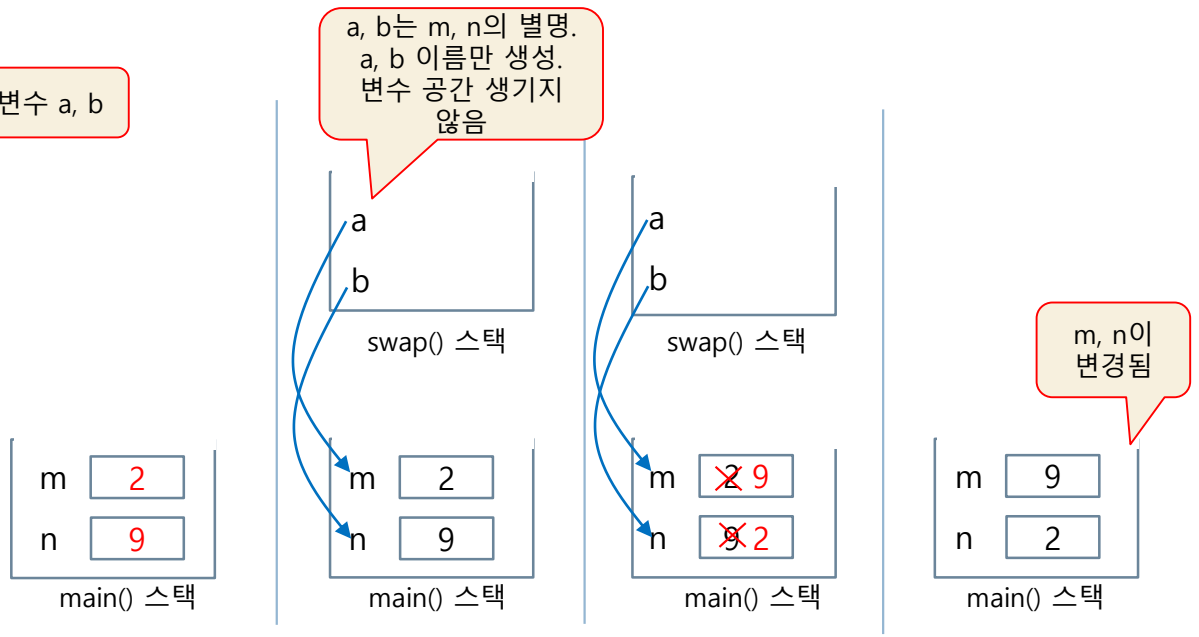

```
#include <iostream>
using namespace std;

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 생긴다.

9 2



예제 5-6 참조에 의한 호출로 Circle 객체에 참조 전달

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();
    int getRadius();
    void setRadius(int radius);
};

Circle::Circle() {
    radius = 1;
    cout << "Circle() radius = " << radius << endl;
}
Circle::Circle(int radius) {
    this->radius = radius;
    cout << "Circle(int radius) radius = " << radius << endl;
}
double Circle::getArea() {
    return 3.14 * radius * radius;
}
int Circle::getRadius() {
    return radius;
}
void Circle::setRadius(int radius) {
    this->radius = radius;
}
Circle::~Circle() {
    cout << "~Circle() radius = " << radius << endl;
}
```

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

예제 5–8 간단한 참조 리턴 사례

ex5-8-0-ValueReturn.cpp

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

// int 형을 반환하는 함수
int change(int value) {
    int i = 0;
    i = value;          // 지역변수에 저장
    return i;           // 지역변수 리턴
}

int main() {
    // change 함수를 호출하여 전달된 값을 받음
    int intval = change(100);

    // 결과 출력
    cout << " 저장된 값: " << intval << endl;

    return 0;
}
```

```
100
```

ex5-8-1-ReferenceReturn.cpp

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

// int 형 포인터를 반환하는 함수
int* change(int value) {
    int* i = new int;  // 동적 메모리 할당
    *i = value;         // 메모리에 값 저장
    return i;           // 포인터 반환
}

int main() {
    // change 함수를 호출하여 int 값을 저장한 포인터를 받음
    int* intval = change(100);

    // 결과 출력
    cout << "저장된 값: " << *intval << endl;

    // 메모리 해제
    delete intval;


    return 0;
}
```

```
100
```

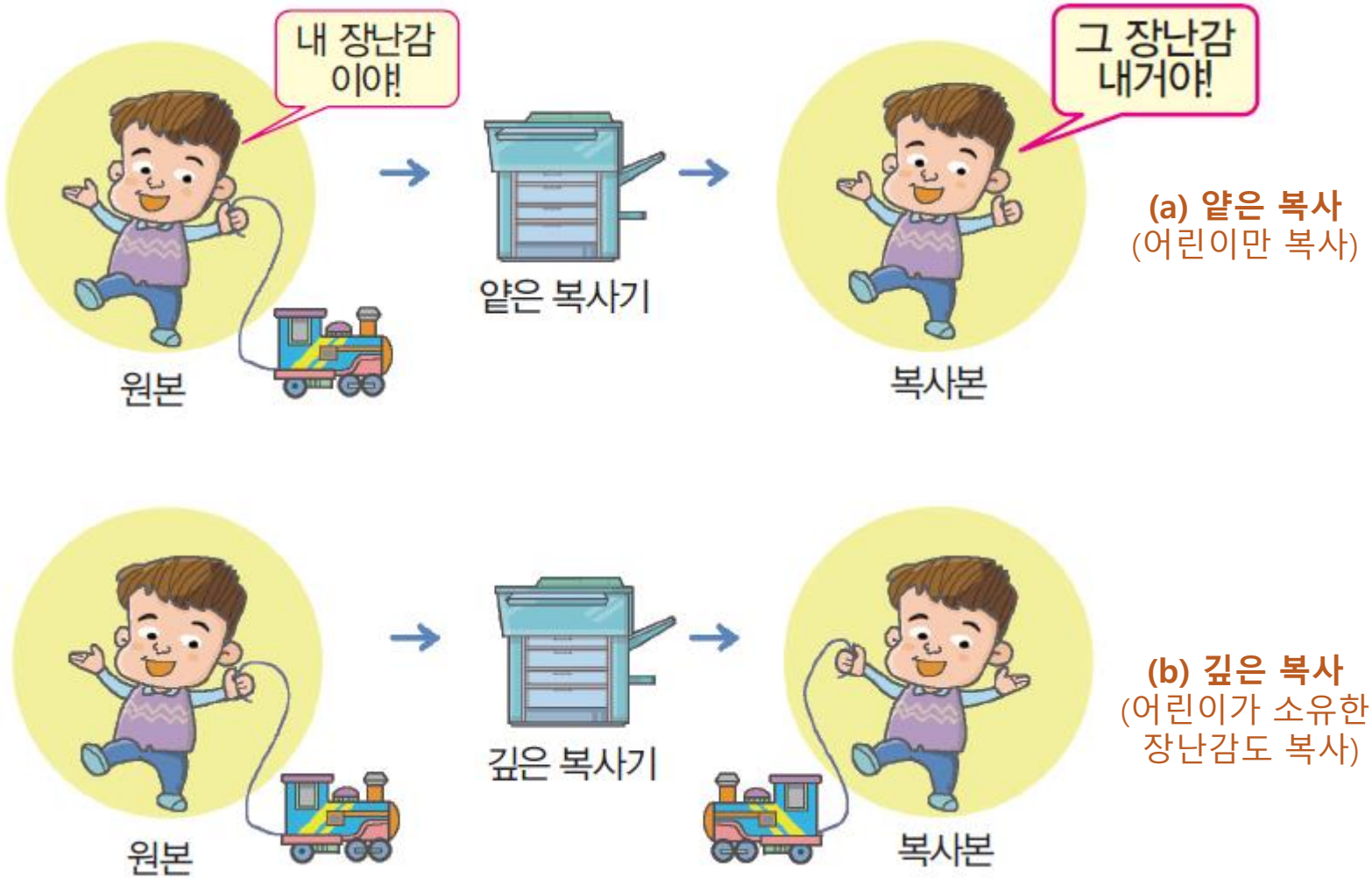
```
#include <iostream>
using namespace std;

class Circle {
    int radius;
public:
    Circle();
    Circle(int r);
    ~Circle();
    double getArea();
    int getRadius();
    void setRadius(int radius);
};

Circle::Circle() {
    radius = 1;
    cout << "Circle() radius = " << radius << endl;
}
Circle::Circle(int radius) {
    this->radius = radius;
    cout << "Circle(int radius) radius = " << radius << endl;
}
double Circle::getArea() {
    return 3.14 * radius * radius;
}
int Circle::getRadius() {
    cout << "Circle 객체의 반지름: " << radius << endl;
    return radius;
}
void Circle::setRadius(int radius) {
    this->radius = radius;
}
Circle::~Circle() {
    //cout << "~Circle() radius = " << radius << endl;
}
```

조

ex5-8-ReferenceReturn.cpp

```
// 1. 값으로 객체 리턴
Circle createCircleByValue(int r) {
    Circle temp(r);  // 임시 객체 생성
    return temp;     // 값으로 리턴 (복사 발생)
}

// 2. 포인터로 객체 리턴
Circle* createCircleByPointer(int r) {
    Circle* ptr = new Circle(r);  // 동적 객체 생성
    return ptr;                   // 포인터 반환
}

// 3. 참조로 객체 리턴
Circle& createCircleByReference(Circle& circle) {
    return circle;   // 참조로 반환
}

int main() {
    int r;
    cout << "=== 값으로 객체 리턴 ===" << endl;
    Circle c1 = createCircleByValue(5);
    r=c1.getRadius();

    cout << "₩n=== 포인터로 객체 리턴 ===" << endl;
    Circle* c2 = createCircleByPointer(10);
    r = c2->getRadius();
    delete c2;  // 동적 할당된 객체는 반드시 해제

    cout << "₩n=== 참조로 객체 리턴 ===" << endl;
    Circle c3(15);
    Circle& c4 = createCircleByReference(c3);
    r = c4.getRadius();

    return 0;
}
```

얕은 복사와 깊은 복사

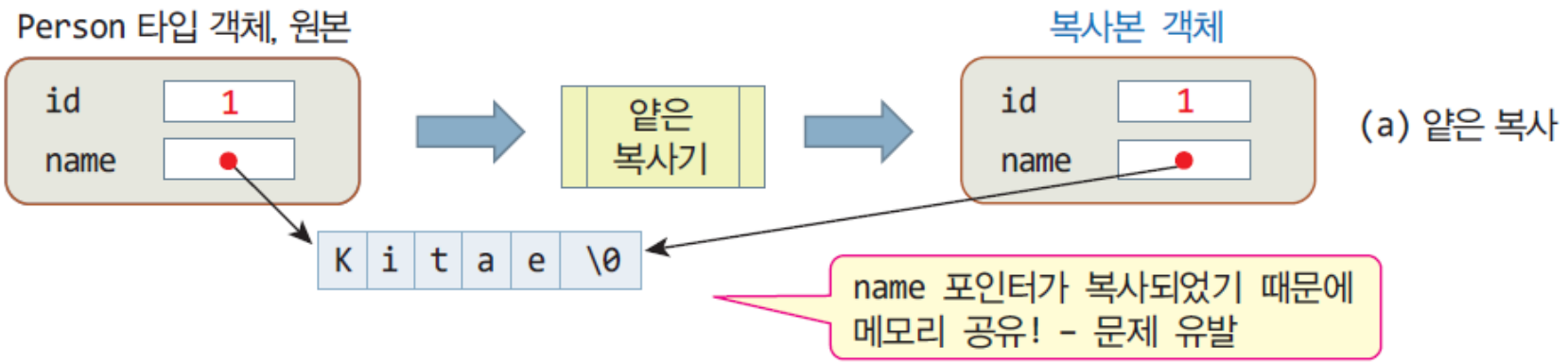

내 장난감 이야!
얕은 복사기
그 장난감 내거야!
원본
복사본
(a) 얕은 복사
(어린이만 복사)
깊은 복사기
원본
복사본
(b) 깊은 복사
(어린이가 소유한 장난감도 복사)

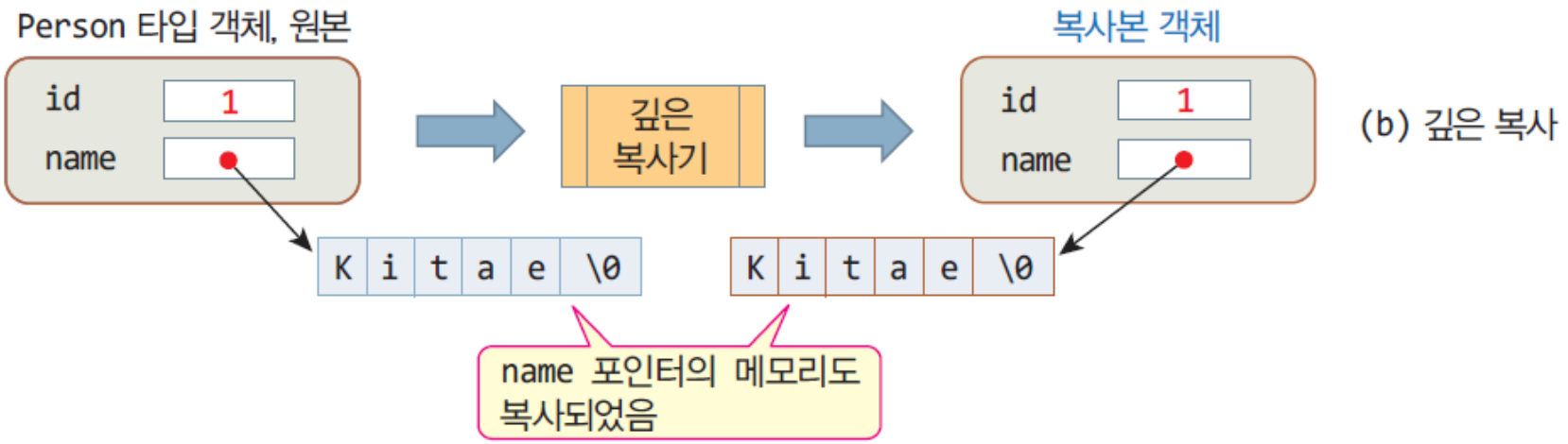
C++에서 얕은 복사와 깊은 복사

- 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;
   ..................
};
```





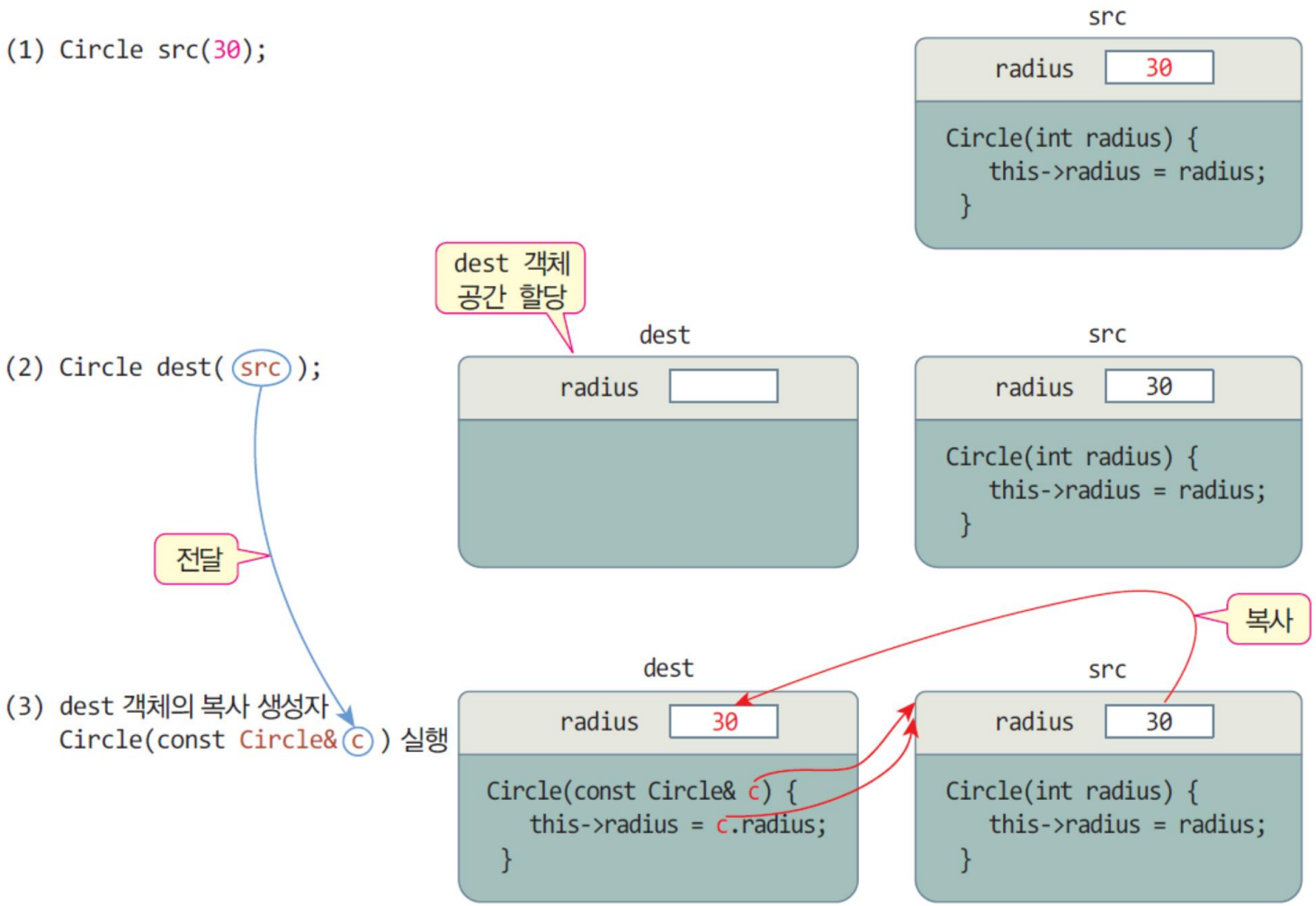
복사 생성자

- 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
    .....
}
```

자기 클래스에 대한 참조 매개 변수

복사 생성 과정


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

예제 5-9 Circle의 복사 생성자와 객체 복사


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

class Circle {
private:
    int radius;
public:
    Circle(const Circle& c);
    Circle();
    Circle(int r);
    double getArea();

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

Circle::Circle() {
    radius = 1;
}
Circle::Circle(int radius) {
    this->radius = radius;
}
double Circle::getArea() {
    return 3.14 * radius * radius;
}
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
src area = 2826
dest area = 2826
```

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;
}
```

예제 5–10 shallow copy


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

class ShallowCopy {
private:
    int* arr; // 배열을 동적 할당으로 관리
    int size;
public:
    // 생성자
    ShallowCopy(int s)  {
        arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = i + 1; // 배열 초기화
        }
    }
    // 복사 생성자 (shallow copy 복사)
    ShallowCopy(const ShallowCopy& obj) {
        arr = obj.arr; // 메모리 주소만 복사
        size = obj.size;
    }

    void setElement(int index, int value) {
        if (index >= 0 && index < size) {
            arr[index] = value;
        }
    }

    void print() {
        for (int i = 0; i < size; i++) cout << arr[i] << " ";
        cout << endl;
    }

    ~ShallowCopy() {
        delete[] arr; // 메모리 해제
    }
};
```

```
int main() {
    ShallowCopy obj1(5);
    ShallowCopy obj2 = obj1; // 얕은 복사

    cout << "Original array in obj1: ";
    obj1.print();

    obj2.setElement(0, 100); // obj2에서 값 변경

    cout << "Modified array in obj1 (affected by obj2): ";
    obj1.print(); // obj1도 영향을 받음

    return 0;
}
```

예제 5–10 shallow copy


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

class ShallowCopy {
private:
    int* arr; // 배열을 동적 할당으로 관리
    int size;
public:
    // 생성자
    ShallowCopy(int s)  {
        arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = i + 1; // 배열 초기화
        }
    }
    // 복사 생성자 (shallow copy 복사)
    ShallowCopy(const ShallowCopy& obj) {
        arr = obj.arr; // 메모리 주소만 복사
        size = obj.size;
    }

    void setElement(int index, int value) {
        if (index >= 0 && index < size) {
            arr[index] = value;
        }
    }

    void print() {
        for (int i = 0; i < size; i++) cout << arr[i] << " ";
        cout << endl;
    }

    ~ShallowCopy() {
        delete[] arr; // 메모리 해제
    }
};
```

```
int main() {
    ShallowCopy obj1(5);
    ShallowCopy obj2 = obj1; // 얕은 복사

    cout << "Original array in obj1: ";
    obj1.print();

    obj2.setElement(0, 100); // obj2에서 값 변경

    cout << "Modified array in obj1 (affected by obj2): ";
    obj1.print(); // obj1도 영향을 받음

    return 0;
}
```

예제 5–11 deep copy

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

class DeepCopy {
private:
    int* arr; // 배열을 동적 할당으로 관리
    int size;
public:
    // 생성자
    DeepCopy(int s) : size(s) {
        arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = i + 1; // 배열 초기화
        }
    }

    // 복사 생성자 (deep copy)
    DeepCopy(const DeepCopy& obj) {
        size = obj.size;
        arr = new int[size]; // 새로운 메모리 할당
        for (int i = 0; i < size; i++) {
            arr[i] = obj.arr[i]; // 데이터 복사
        }
    }

    void setElement(int index, int value) {
        if (index >= 0 && index < size) {
            arr[index] = value;
        }
    }

    void print() {
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
            cout << endl;
        }
    }

    ~DeepCopy() {
        delete[] arr; // 메모리 해제
    }
};
```

```
int main() {
    DeepCopy obj1(5);
    DeepCopy obj2 = obj1; // 깊은 복사

    cout << "Original array in obj1: ";
    obj1.print();

    obj2.setElement(0, 100); // obj2에서 값 변경

    cout << "Modified array in obj2: ";
    obj2.print();

    cout << "Array in obj1 remains unchanged: ";
    obj1.print(); // obj1은 영향을 받지 않음

    return 0;
}
```