05

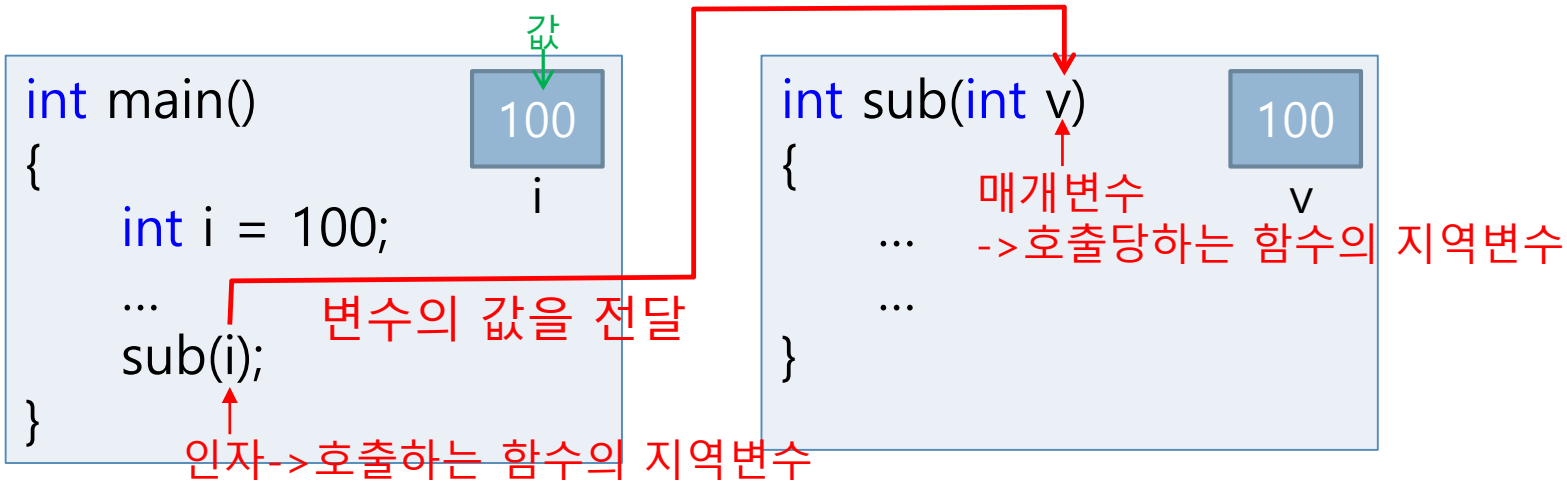
함수와 참조, 복사 생성자

학습 목표

1. 값에 의한 호출과 주소에 의해 호출을 복습한다.
2. 함수 호출 시 객체가 전달되는 과정을 이해한다.
3. 객체 치환과 객체 리턴을 이해한다.
4. 참조에 대한 개념을 이해하고, 참조 변수를 선언할 수 있다.
5. 참조에 의한 호출과 참조 리턴에 대해 이해하고 코드를 작성할 수 있다.
6. 복사생성자의 필요성과 활용을 이해하고, 작성할 수 있다.

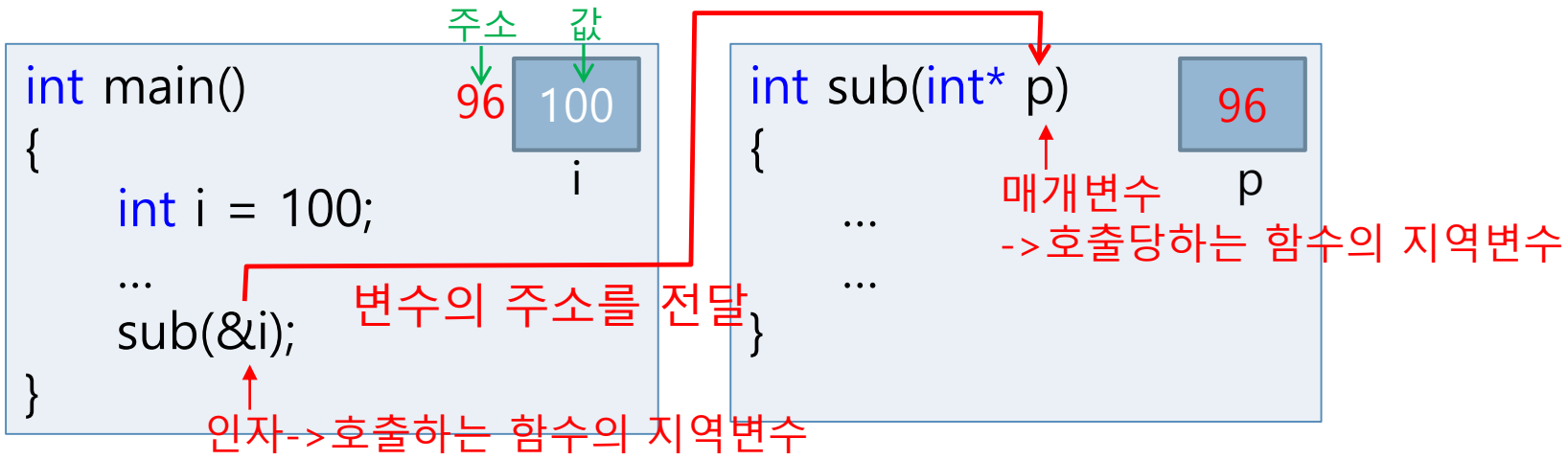
값에 의한 호출(call by value)

- 함수를 호출할 때 인자로 변수의 값을 넘겨줌
- 함수가 호출되면 매개 변수가 메모리에 생성됨
- 호출하는 코드에서 전달된 인자가 매개 변수에 복사됨



주소에 의한 호출(call by address)

- 호출하는 코드에서는 인자로 변수의 주소를 넘겨줌
- 함수의 매개 변수는 포인터 타입
- 함수가 호출되면 포인터 타입의 매개 변수가 메모리에 생성됨
- 호출하는 코드에서 넘어온 인자가 매개 변수에 저장됨



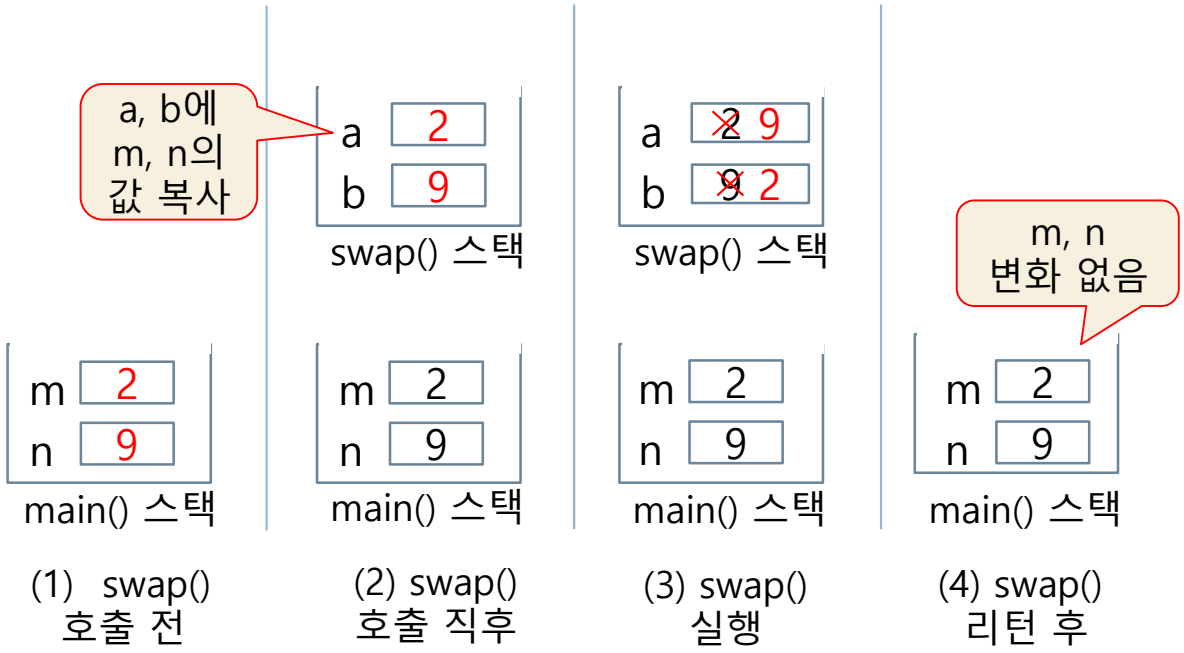
값에 의한 호출


```
#include<iostream>
using namespace std;
int getSum(int a, int b);
int main(void)
{
	int a, b;
	cout << "두 정수를 입력하세요:";
	cin >> a >> b;
	int sum = getSum(a, b);		// 값에 의한 호출
	cout << "합 :" << sum << endl;
	return 0;
}
int getSum(int a, int b)		// 호출시 매개변수 할당및 인수로 초기화
{
	int sum;
	sum = a + b;
	return sum;
}
```

```
두 정수를 입력하세요:10 20
합 :30
```

값에 의한 호출의 문제점


```
#include <iostream>          2 9
using namespace std;
void swap(int a, int b);
int main() {
        int m = 2, n = 9;
        swap(m, n);
        cout << m << ' ' << n;
}
void swap(int a, int b) {
        int tmp;
        tmp = a;
        a = b;
        b = tmp;
}
```

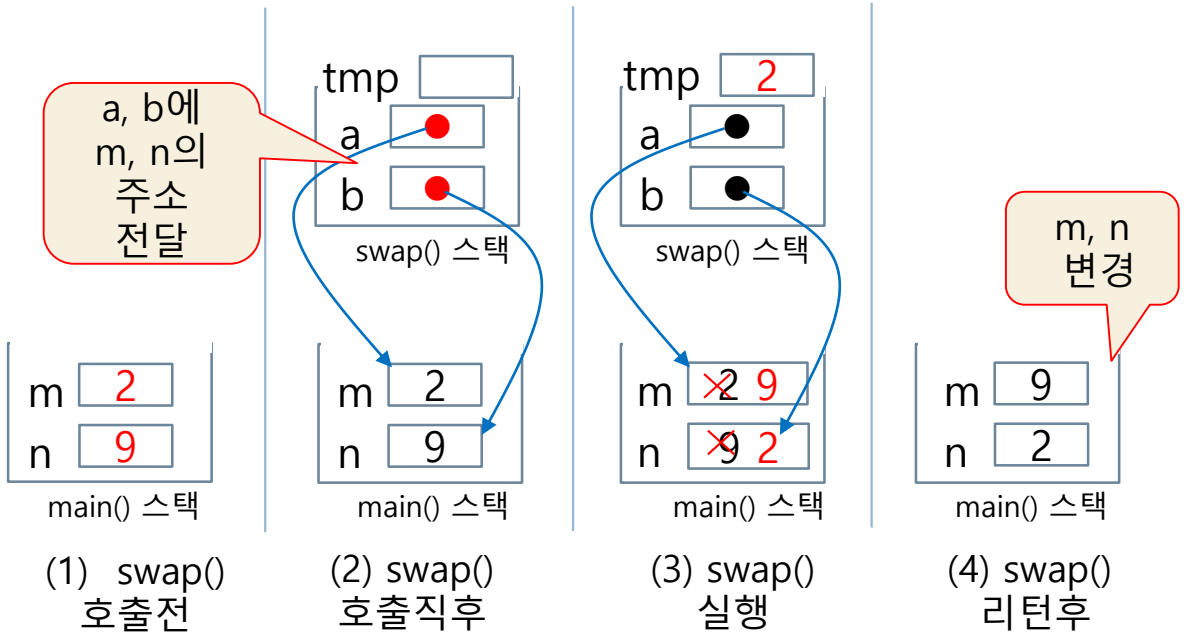


값에 의한 호출

-> 다른 함수에서 선언된 변수의 값을 변경할 수 없다
스택 : 지역변수가 할당되는 메모리 영역

주소에 의한 호출


```
#include <iostream>          9 2
using namespace std;
void swap(int* a, int* b );
int main() {
        int m = 2, n = 9;
        swap(&m, &n);
        cout << m << ' ' << n;
}
void swap(int* a, int* b) {
        int tmp;
        tmp = *a;
        *a = *b;
        *b = tmp;
}
```



주소에 의한 호출

->다른 함수에서 선언된 변수의 값을 변경할 수 있다

스택 : 지역변수가 할당되는 메모리 영역

함수와 객체

- 클래스도 자료형 -> 함수의 매개변수와 반환형에 클래스 선언 가능
- 객체를 값에 의한 호출, 주소에 의한 호출 방식으로 전달가능

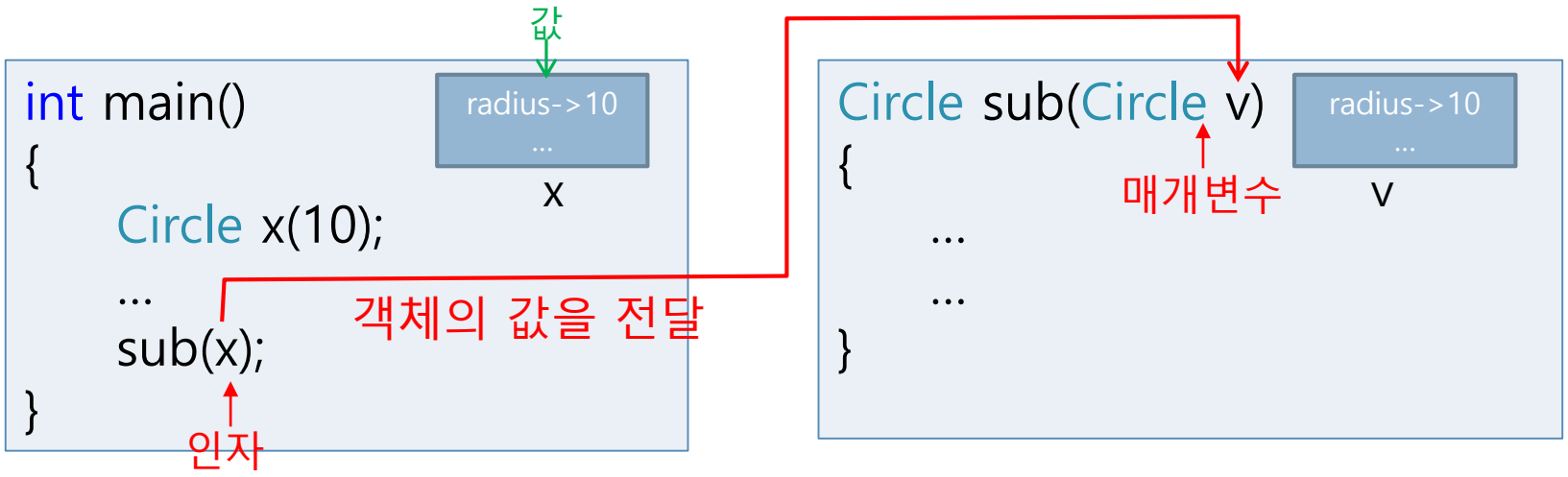
```
Circle funCircle(Circle x, Circle y) {          // 값에 의한 호출
    …
}

Circle funCircle(Circle* x, Circle* y) {        // 주소에 의한 호출
    …
}
```

값에 의한 호출로 함수에 객체 전달

- 함수 호출 시 매개 변수 객체 생성 및 인자 객체로 초기화
  - 매개 변수 객체가 메모리에 생성됨
  - **매개 변수 객체의 생성자는 호출되지 않음**
  - 호출하는 쪽의 인자객체의 값이 매개 변수 객체에 복사됨
  - **생성자 대신에 복사생성자가 호출됨**
- 함수 종료 시 매개변수 객체 소멸
  - 매개 변수 객체의 소멸자 호출 후 제거
- 매개 변수 객체의 생성자가 실행되지 않는 이유
  - 생성자실행 -> 인자객체를 매개변수 객체에 대입(연산자함수) -> 2번의 함수 호출 -> 실행시간 길어 짐
  - 복사생성자 1개의 함수 호출로 생성자 호출, 대입연산자 함수 호출을 대체함 -> 성능 향상

값에 의한 호출로 함수에 객체 전달

- 매개변수가 객체인 경우 값에 의한 호출

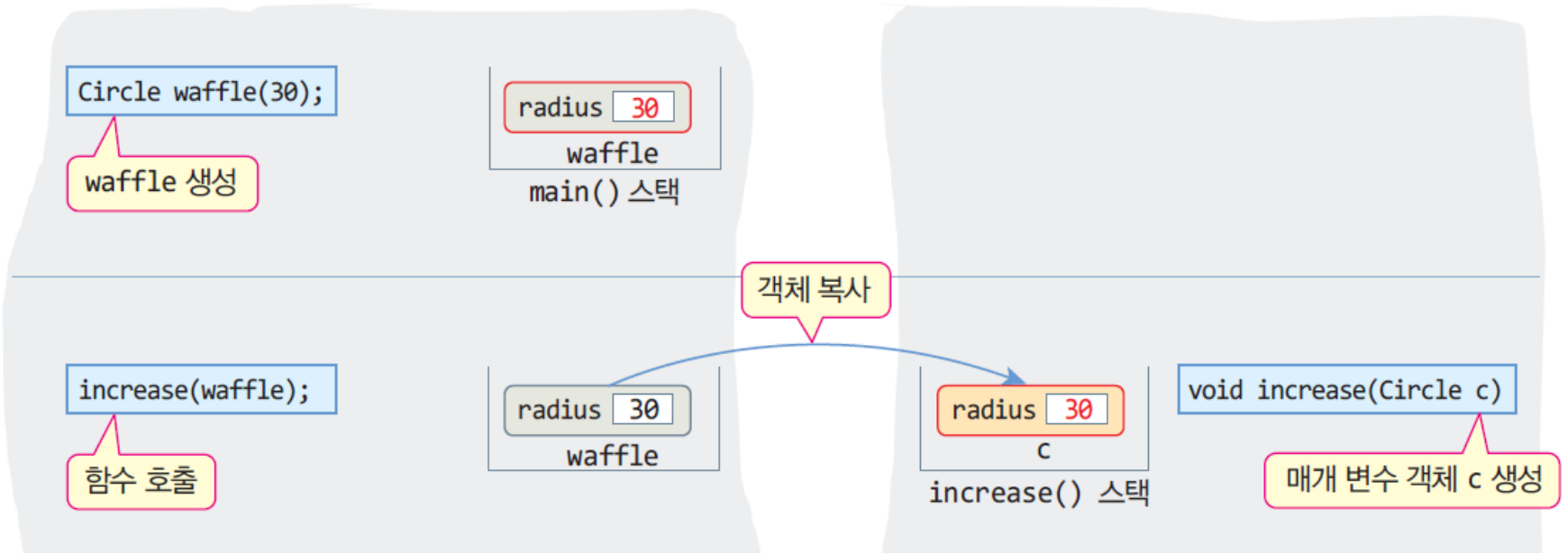


값에 의한 호출로 함수가 호출되는 과정

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

30

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

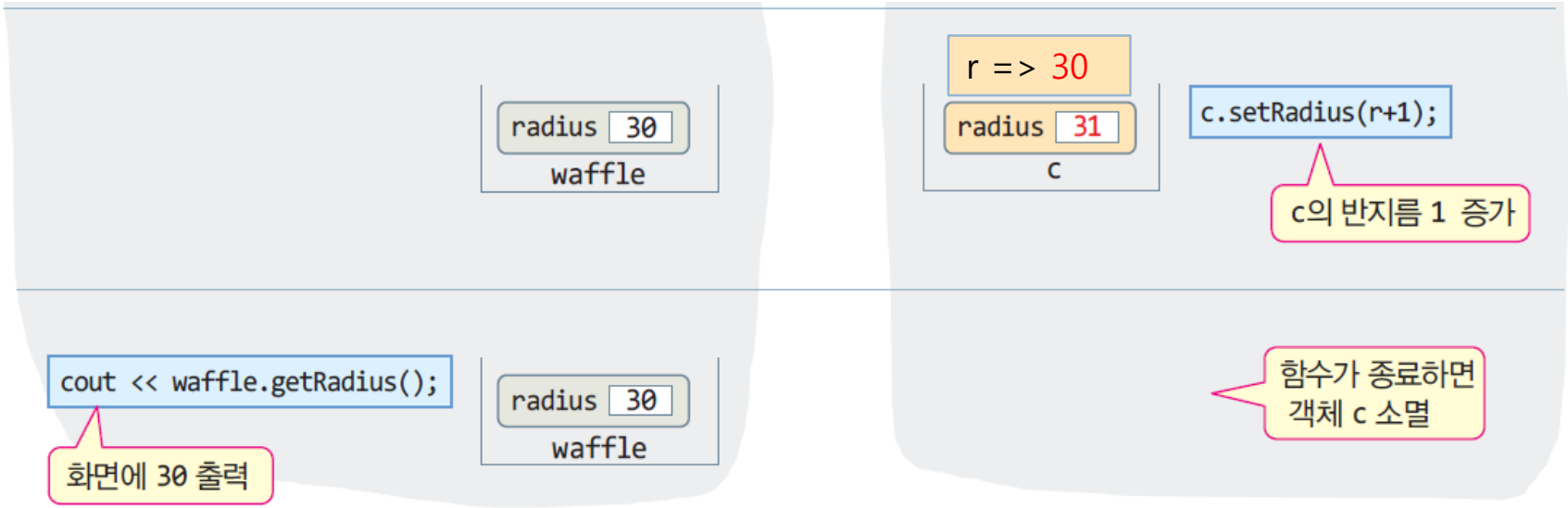


값에 의한 호출로 함수가 호출되는 과정

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

30

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



예제 5-1 값에 의한 호출시 생성자 미실행


```
#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;
}
```

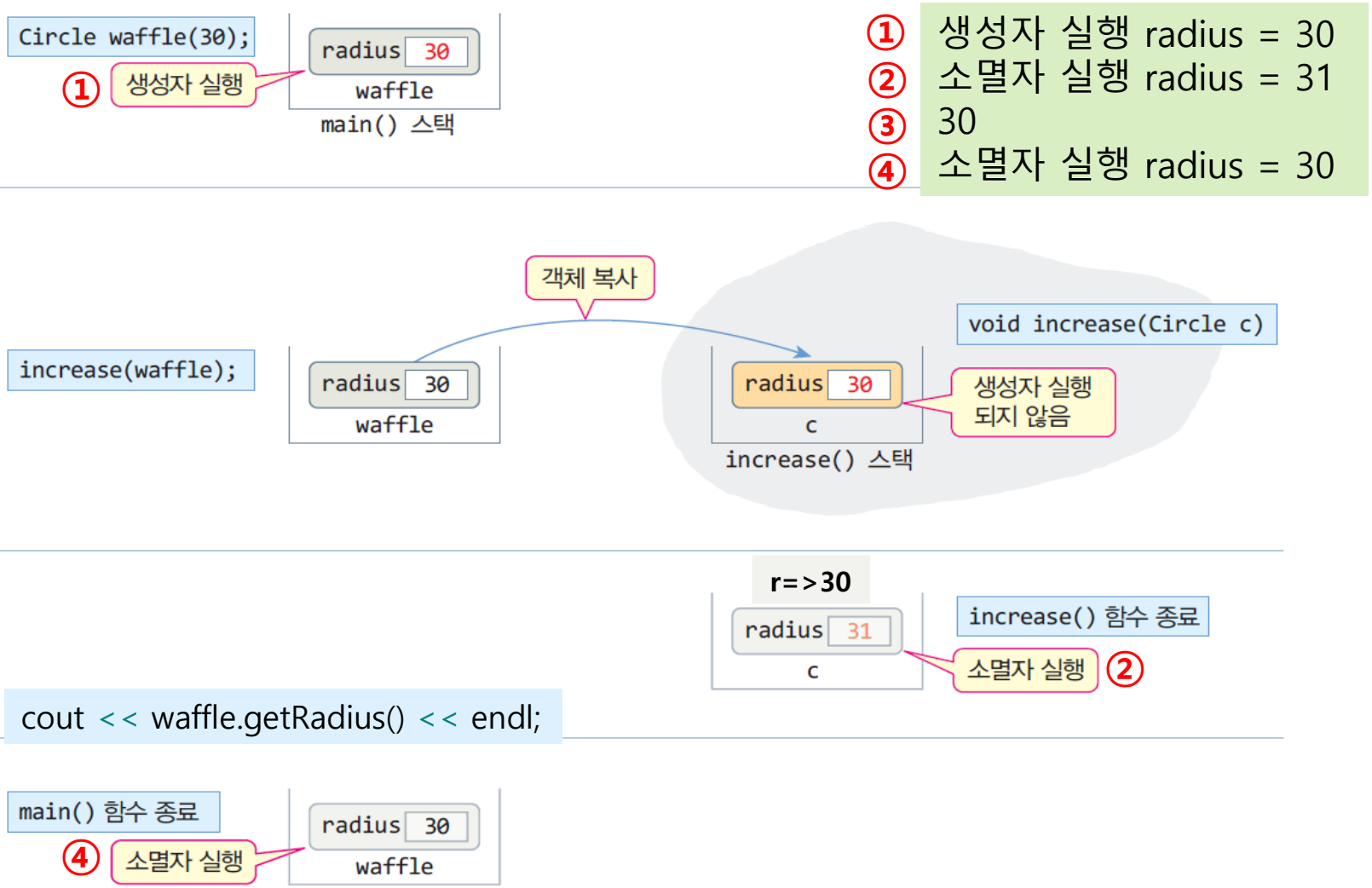
예제 5-1 값에 의한 호출시 생성자 미실행


```
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;
}
```

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

매개변수 c의 생성자 실행되지 않았음

예제 5-1 값에 의한 호출시 생성자 미실행

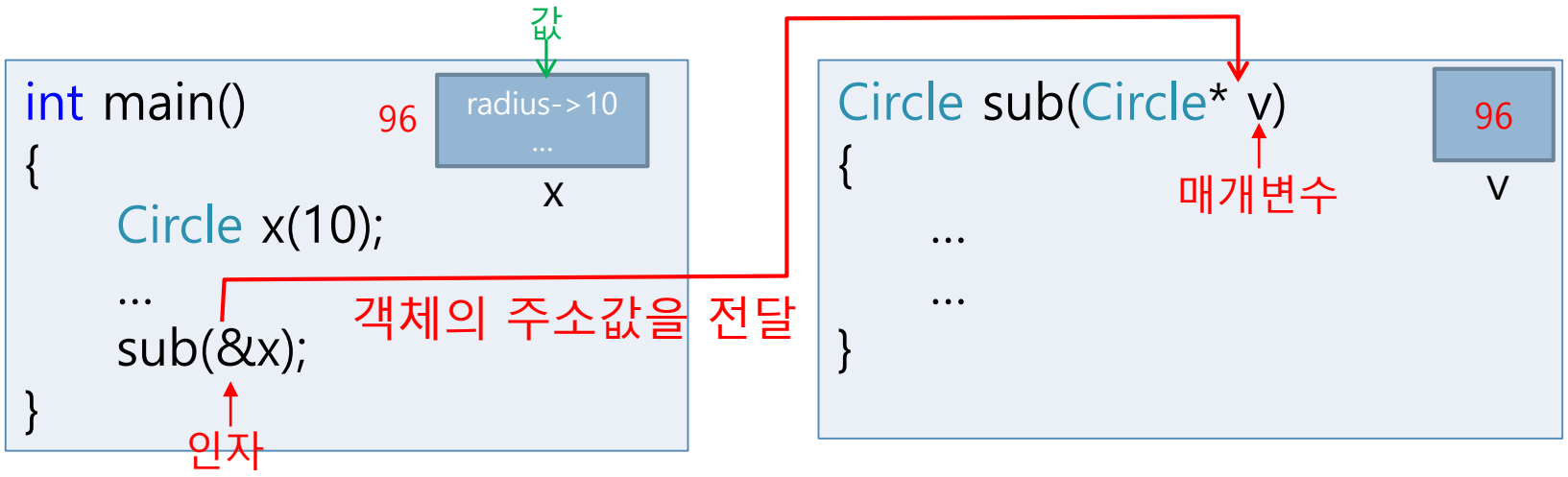


주소에 의한 호출로 함수에 객체 전달

- 함수 호출시 인자로 객체의 주소를 전달
- 함수의 매개 변수는 객체 포인터로 선언
- 객체 포인터는 객체가 아니므로 함수 호출 시 생성자, 소멸자가 실행되지 않음
- 주소만 복사하면 되므로 객체를 복사하는데 걸리는 시간을 절약
- **메모리를 절약하고 실행시간이 빠른 장점이 있음**

주소에 의한 호출로 함수에 객체 전달

- 매개변수가 객체인 경우 주소에 의한 호출

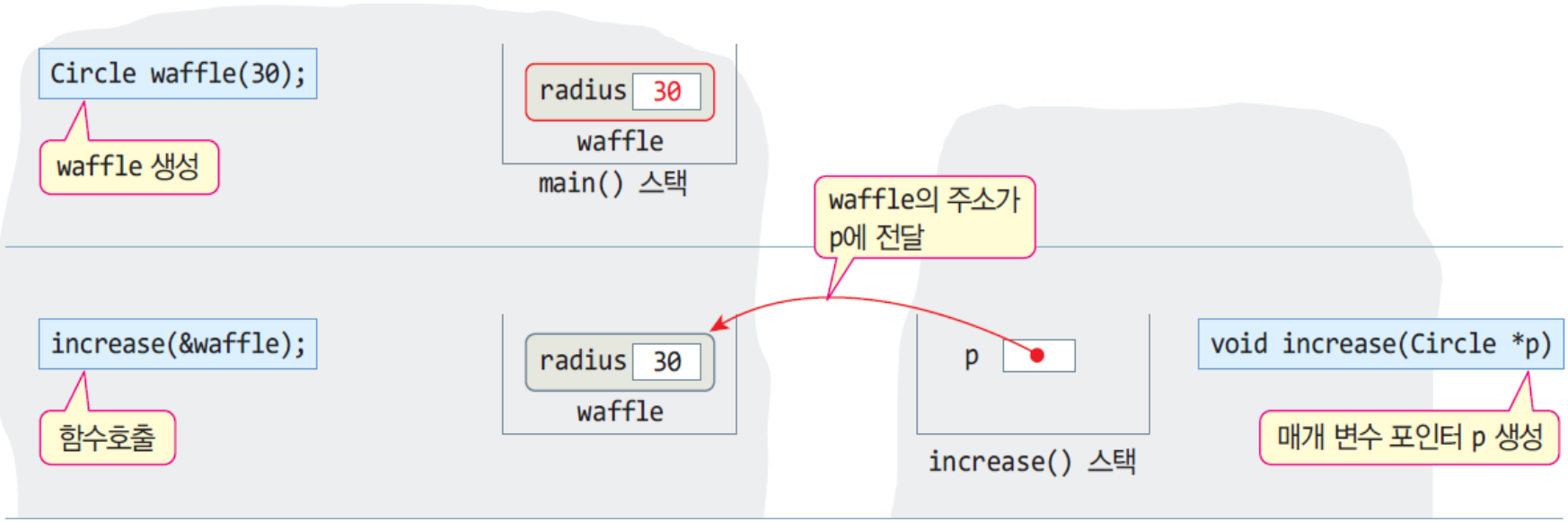


주소에 의한 호출로 함수가 호출되는 과정


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

31

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

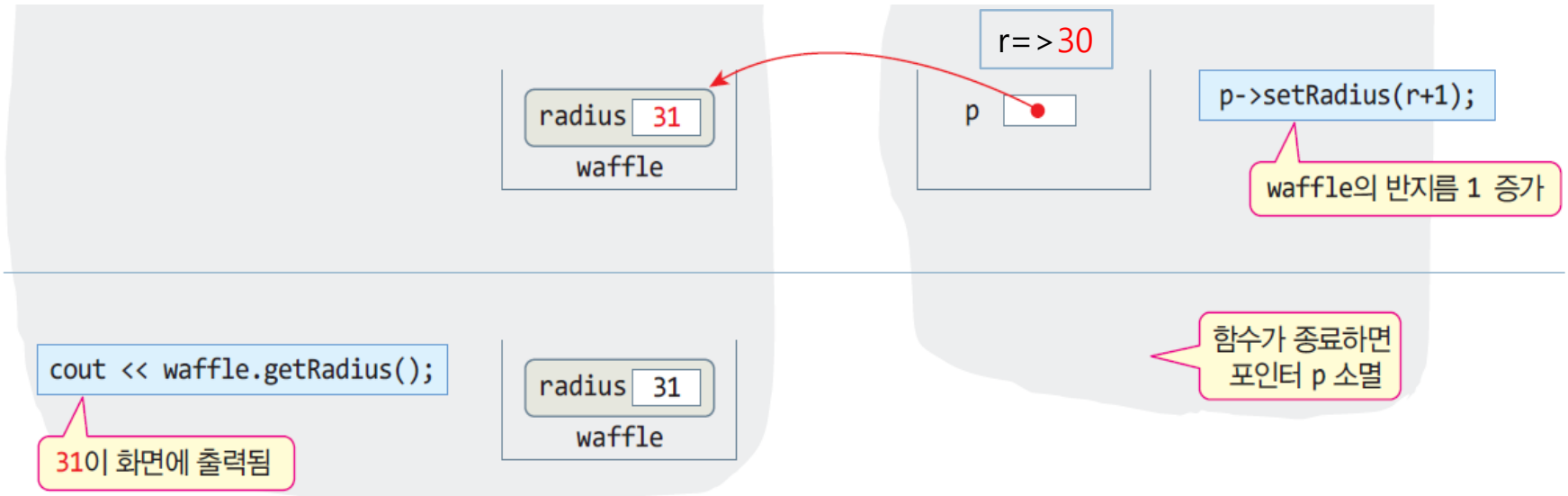


주소에 의한 호출로 함수가 호출되는 과정


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

31

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



객체의 연산

- 기본자료형(char, int, double)의 변수에 대해서는 모든 연산(산술, 논리, 관계 등)이 가능
- 연산자는 컴퓨터의 CPU에 의해 바로 실행됨 -> 함수가 아님

```
int c1=1, c2=2, c3;
c3 = c1 + c2;          // 정상
c3 = c1 * c2;          // 정상
c3 = c1 > c2;          // 정상
c3 = c1 || c2;         // 정상
c3 = c1 % c2;          // 정상
```

객체의 연산

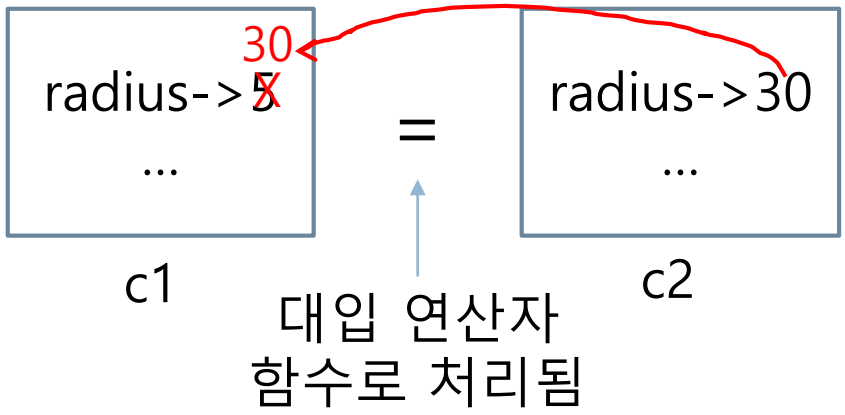
- 피연산자가 객체인 경우는 기존의 모든 연산이 불가능
- 객체 연산이 가능 하려면 연산자함수로 정의해야 함(7장)
- 대입연산은 컴파일러에 의해 연산자 함수가 자동으로 추가됨 -> 연산자 함수 정의하지 않아도 대입연산만 가능

```
Circle c1(5), c2(30), c3;
c1 = c2;                    // 대입연산만 가능
c3 = c1 + c2;               // 오류 -> +연산자함수를 정의해줘야 가능
c3 = c1 * c2;               // 오류 -> * 연산자함수를 정의해줘야 가능
c3 = c1 > c2;               // 오류 -> >연산자함수를 정의해줘야 가능
```

객체 대입(=)

- 대입 연산은 모든 객체에 적용 가능 -> 컴파일러가 객체를 생성할 때 대입연산자 함수를 자동으로 추가해줌
- 동일한 클래스 타입의 객체에 적용가능한 연산
- 객체의 모든 데이터가 비트 단위로 복사(멤버변수끼리 복사)

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



객체 리턴

- 함수의 반환형으로 클래스 선언가능

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

```
Circle c;                //c의 반지름 1
c = getCircle();         //tmp객체의 복사본이 c에 대입
```

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 tmp을 리턴한다.
}
```

예제 5–3 객체 리턴


```
int main() {
	Circle c;	// 객체 생성시 radius=1로 초기화
	cout << c.getArea() << endl;

	c = getCircle();		// radius=30으로 변경됨
	cout << c.getArea() << endl;
}
```

```
3.14
2826
```

예제 : 주소에 의한 호출


```
#include <iostream>                                          <hello>
#include <string>
using namespace std;
void AddString(string* s1);          //주소에 의한 호출
int main() {
      string s1("hello");
      AddString(&s1);
      cout << s1 << endl;
}
void AddString(string* s1)          //string객체에 <>를 연결후 저장해줌
{
      string left("<"), right(">");
      *s1 = left + *s1 + right;      //string객체는 덧셈, 대입 모두 가능
}
```

예제 : 객체를 리턴하는 함수

helloworld

```
#include <iostream>
#include <string>
using namespace std;
string GetStringCat(string s1, string s2);  //값에 의한 호출
int main() {
	string s1("hello");
	string s2("world");
	cout << GetStringCat(s1,s2) << endl;
}
string GetStringCat(string s1, string s2) //string객체2개를 연결후 리턴
{
	string res;
	res = s1 + s2;		//string객체는 덧셈, 대입연산 모두 가능
	return res;
}
```

실습과제1

- 인자와 매개변수를 구분하여 설명하시오.
- 함수 호출시 자동으로 처리되는 2가지는 무엇인가?
- 값에 의한 호출로 다른 함수에서 선언된 변수의 값을 변경할 수 없다. 이유를 설명하라.
- 주소에 의한 호출을 사용해야하는 경우를 2가지 이상 설명하라.

실습과제2

- 함수호출문을 참고하여 GetLatterString 함수의 정의, 선언을 추가 하시오. 힌트) 값에 의한 호출을 사용하고 매개변수, 리턴값 모두 객체임

사전에서 뒤에 나오는 문자열은 world 입니다.

```
#include <iostream>
#include <string>
using namespace std;
// 함수 선언 추가
int main() {
	string s1("hello");
	string s2("world");
	string res;
	res = GetLatterString(s1, s2);		// 함수 호출
	cout << "사전에서 뒤에 나오는 문자열은 " << res << "입니다." << endl;
}
// 함수 정의 추가
```

실습과제3

- 함수호출문을 참고하여 GetLatterString 함수의 정의, 선언을 추가 하시오. 힌트) 주소에 의한 호출을 사용하고 매개변수는 객체포인터로 리턴값은 객체로 선언할 것

사전에서 뒤에 나오는 문자열은 world 입니다.

```
#include <iostream>
#include <string>
using namespace std;
// 함수 선언 추가
int main() {
	string s1("hello");
	string s2("world");
	string res;
	res = GetLatterString(&s1, &s2);		// 함수 호출
	cout << "사전에서 뒤에 나오는 문자열은 " << res << "입니다." << endl;
}
// 함수 정의 추가
```

실습과제4

- 다음 결과가 나오도록 두개의 string 객체의 문자열을 교환하는 SwapString 함수를 만드시오.

호출전 s1:hello s2:world
호출후 s1:world s2:hello

```
#include <iostream>
#include <string>
using namespace std;
// 함수 선언 추가
int main() {
        string s1("hello");
        string s2("world");
        cout << "호출전 s1:" << s1 << "s2:" << s2 << 두이;
        SwapString(....);          // 함수 호출
        cout << "호출후 s1:" << s1 << "s2:" << s2 << endl;
}
// 함수 정의 추가
```

실습과제5

- 사전에서 제일 뒤에 나오는 문자열을 구하는 함수를 작성 하시오. 힌트 : string객체배열을 인수로 받고 string객체를 리턴하는 함수임

```
#include <iostream>
#include <string>
using namespace std;
// 함수 선언 추가
int main() {
	string names[5];
	for (int i = 0; i < 5; i++) {
		cout << "이름 >> ";
		getline(cin, names[i], '₩n');
	}
	// 함수 호출
	cout << "사전에서 가장 뒤에 나오는 문자열은 " << res << endl;
}
// 함수 정의 추가
```

```
이름 >> Kim Nam Yun<엔터>
이름 >> Chang Jae Young<엔터>
이름 >> Lee Jae Moon<엔터>
이름 >> Han Won Sun<엔터>
이름 >> Hwang Su hee<엔터>
사전에서 가장 뒤에 나오는 문자열은 Lee Jae Moon
```

과제제출방법

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

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

// 소스코드 작성
```