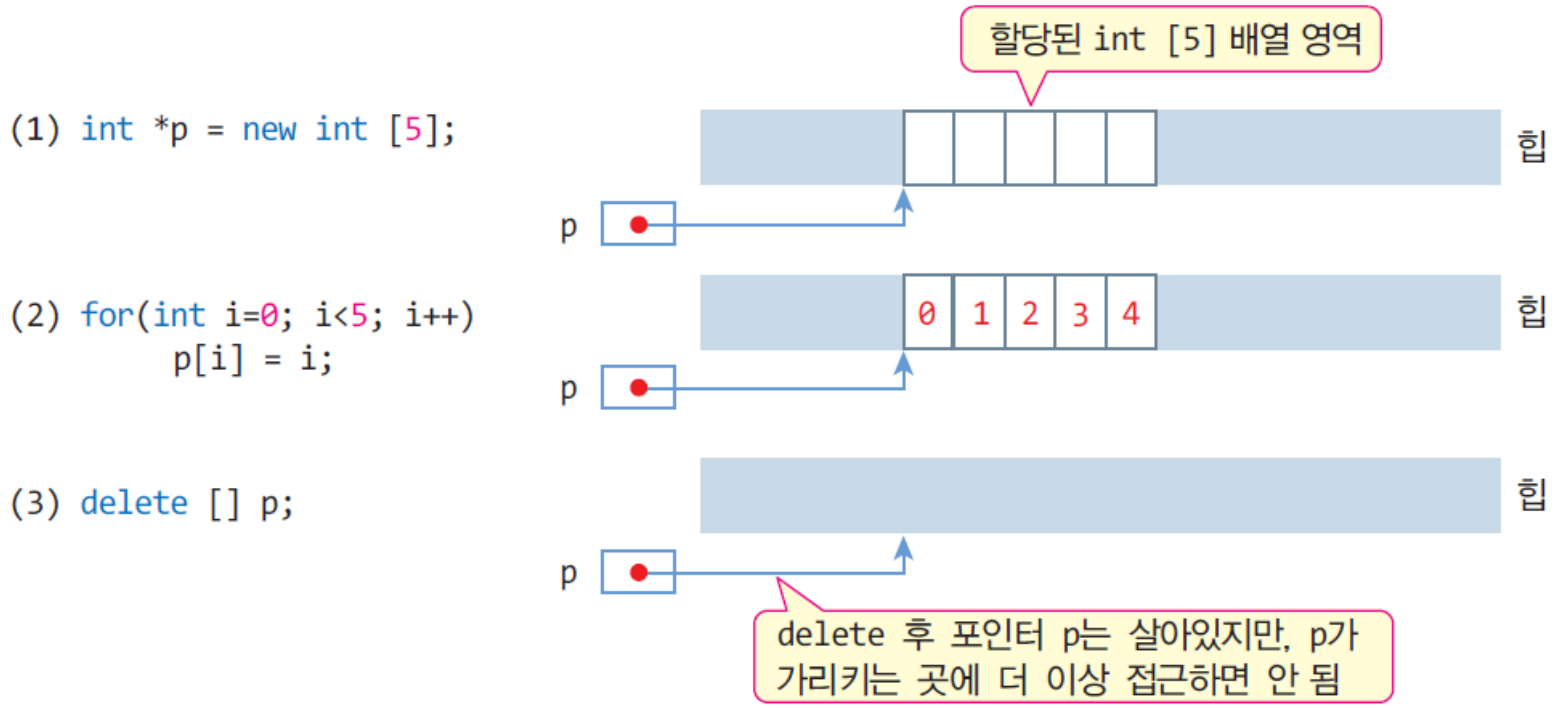
04

객체 포인터와 객체 배열, 객체의 동적 생성

배열의 동적 할당 및 반환

- new/delete 연산자의 사용 형식

데이터타입 *포인터변수 = **new** 데이터타입 **[배열의 크기]**; // 동적 배열 할당
delete [] 포인터변수; // 배열 반환



예제 4-6 정수형 배열의 동적 할당 및 반환

사용자로부터 입력할 정수의 개수를
입력 받아 배열을 동적 할당 받고,
하나씩 정수를 입력 받은 후
합을 출력하는 프로그램을 작성하라.

```
입력할 정수의 개수는?4
1번째 정수: 4
2번째 정수: 20
3번째 정수: -5
4번째 정수: 9
평균 = 7
```

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

int main() {
    cout << "입력할 정수의 개수는?";
    int n;
    cin >> n; // 정수의 개수 입력
    if(n <= 0) return 0;
    int *p = new int[n]; // n 개의 정수 배열 동적 할당
    if(!p) {
        cout << "메모리를 할당할 수 없습니다.";
        return 0;
    }

    for(int i=0; i<n; i++) {
        cout << i+1 << "번째 정수: "; // 프롬프트 출력
        cin >> p[i]; // 키보드로부터 정수 입력
    }

    int sum = 0;
    for(int i=0; i<n; i++)
        sum += p[i];
    cout << "평균 = " << sum/n << endl;

    delete [] p; // 배열 메모리 반환
}
```

동적 할당 메모리 초기화 및 delete 시 유의 사항

- 동적 할당 메모리 초기화
  - 동적 할당 시 초기화

```
데이터타입 *포인터변수 = new 데이터타입(초깃값);
```

```
int *pInt = new int(20); // 20으로 초기화된 int 타입 할당
char *pChar = new char('a'); // 'a'로 초기화된 char 타입 할당
```

  - 배열은 동적 할당 시 초기화 불가능

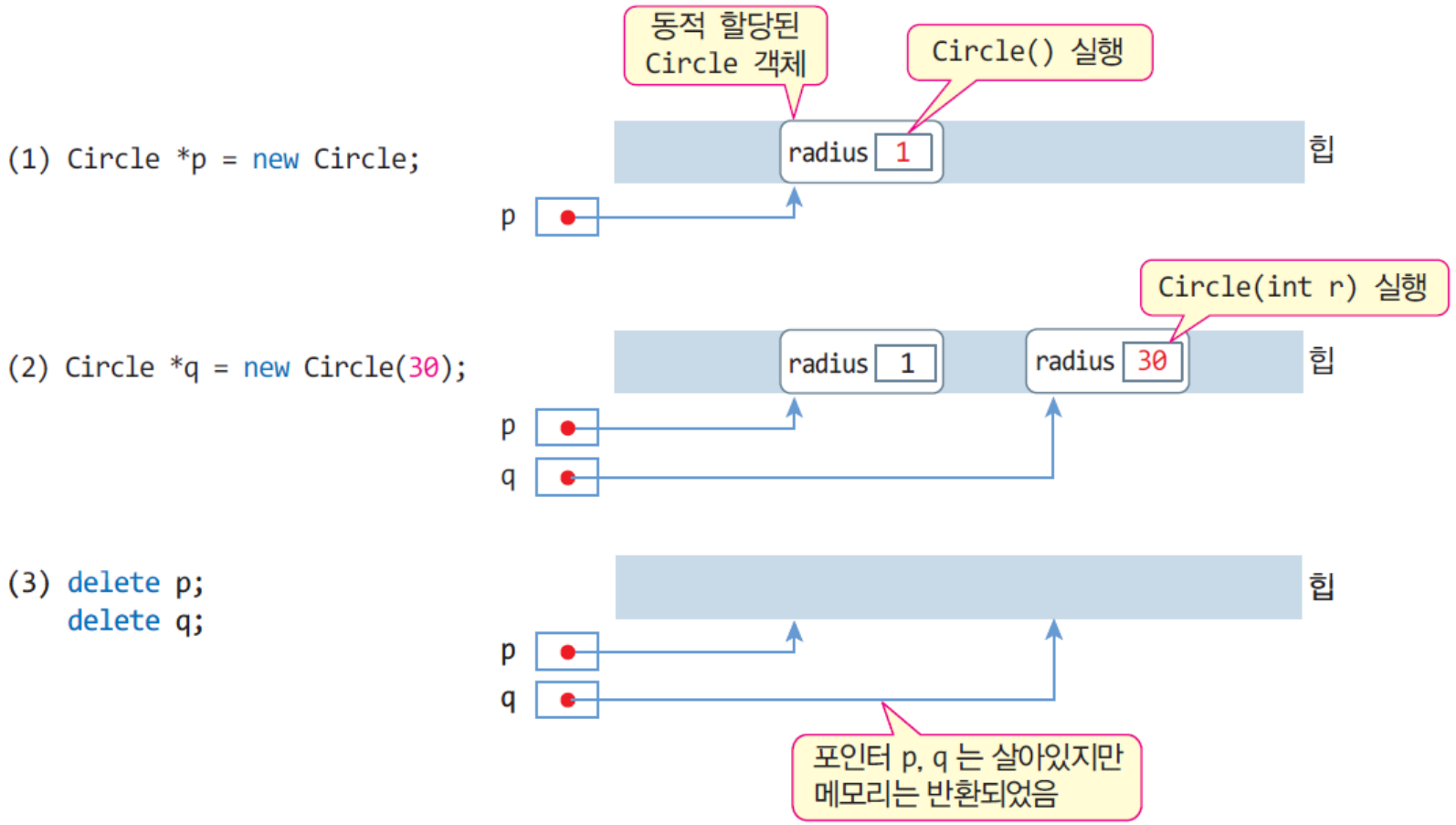
```
int *pArray = new int [10](20); // 구문 오류. 컴파일 오류 발생
int *pArray = new int(20)[10]; // 구문 오류. 컴파일 오류 발생
```

- delete시 [] 생략
  - 컴파일 오류는 아니지만 비정상적인 반환

```
int *p = new int [10];
delete p; // 비정상 반환. delete [] p;로 하여야 함.

int *q = new int;
delete [] q; // 비정상 반환. delete q;로 하여야 함.
```

객체의 동적 생성 및 반환

클래스이름 *포인터변수 = **new** 클래스이름;
클래스이름 *포인터변수 = **new** 클래스이름(생성자매개변수리스트);
delete 포인터변수;



예제 4-7 Circle 객체의 동적 생성 및 반환


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

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

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

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

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

```
int main() {
    Circle *p, *q;
    p = new Circle;
    q = new Circle(30);
    cout << p->getArea() << endl << q->getArea() << endl;
    delete p;
    delete q;
}
```

생성한 순서에 관계 없이 원하는 순서대로 delete 할 수 있음

```
생성자 실행 radius = 1
생성자 실행 radius = 30
3.14
2826
소멸자 실행 radius = 1
소멸자 실행 radius = 30
```

예제 4–8 Circle 객체의 동적 생성과 반환 응용

정수 반지름을 입력 받고 Circle 객체를 동적 생성하여 면적을 출력하라. 음수가 입력되면 프로그램은 종료한다.

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

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

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

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

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

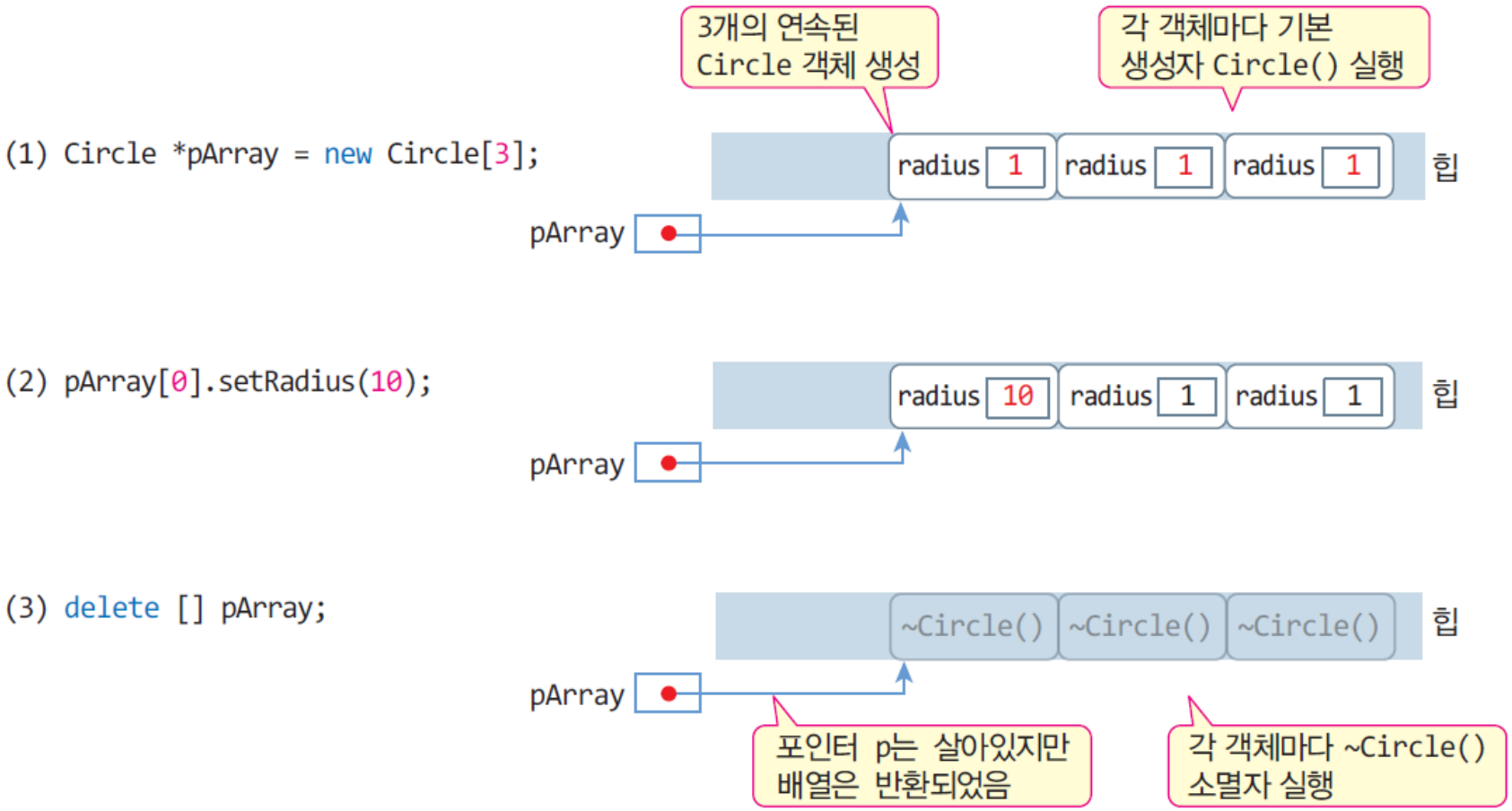
```
int main() {




}
```

```
정수 반지름 입력(음수이면 종료)>> 5
생성자 실행 radius = 5
원의 면적은 78.5
소멸자 실행 radius = 5
정수 반지름 입력(음수이면 종료)>> 9
생성자 실행 radius = 9
원의 면적은 254.34
소멸자 실행 radius = 9
정수 반지름 입력(음수이면 종료)>> -1
```

음수가 입력되면 종료

객체 배열의 동적 생성 및 반환

클래스이름 *포인터변수 = **new** 클래스이름 **[배열 크기]**;
delete [] 포인터변수; // 포인터변수가 가리키는 객체 배열을 반환



객체 배열의 사용, 배열의 반환과 소멸자

- 동적으로 생성된 배열도 보통 배열처럼 사용

```
Circle *pArray = new Circle[3]; // 3개의 Circle 객체 배열의 동적 생성

pArray[0].setRadius(10); // 배열의 첫 번째 객체의 setRadius() 멤버 함수 호출
pArray[1].setRadius(20); // 배열의 두 번째 객체의 setRadius() 멤버 함수 호출
pArray[2].setRadius(30); // 배열의 세 번째 객체의 setRadius() 멤버 함수 호출

for(int i=0; i<3; i++) {
    cout << pArray[i].getArea(); // 배열의 i 번째 객체의 getArea() 멤버 함수 호출
}
```

- 포인터로 배열 접근

```
pArray->setRadius(10);
(pArray+1)->setRadius(20);
(pArray+2)->setRadius(30);

for(int i=0; i<3; i++) {
    (pArray+i)->getArea();
}
```

- 배열 소멸

```
delete [] pArray;
```

pArray[2] 객체의 소멸자 실행(1)
pArray[1] 객체의 소멸자 실행(2)
pArray[0] 객체의 소멸자 실행(3)

각 원소 객체의 소멸자 별도 실행. 생성의 반대순

예제 4-9 Circle 배열의 동적 생성 및 반환

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

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

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

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

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

```
int main() {
    Circle *pArray = new Circle [3]; // 객체 배열 생성

    pArray[0].setRadius(10);
    pArray[1].setRadius(20);
    pArray[2].setRadius(30);

    for(int i=0; i<3; i++) {
        cout << pArray[i].getArea() << '₩n';
    }
    Circle *p = pArray; // 포인터 p에 배열의 주소값으로 설정
    for(int i=0; i<3; i++) {
        cout << p->getArea() << '₩n';
        p++; // 다음 원소의 주소로 증가
    }

    delete [] pArray; // 객체 배열 소멸
}
```

각 원소 객체의 기본 생성자 Circle() 실행

각 배열 원소 객체의 소멸자 ~Circle() 실행

```
생성자 실행 radius = 1
생성자 실행 radius = 1
생성자 실행 radius = 1
314
1256
2826
314
1256
2826
소멸자 실행 radius = 30
소멸자 실행 radius = 20
소멸자 실행 radius = 10
```

소멸자는 생성의 반대 순으로 실행

예제 4-10 객체 배열의 동적 생성과 반환 응용

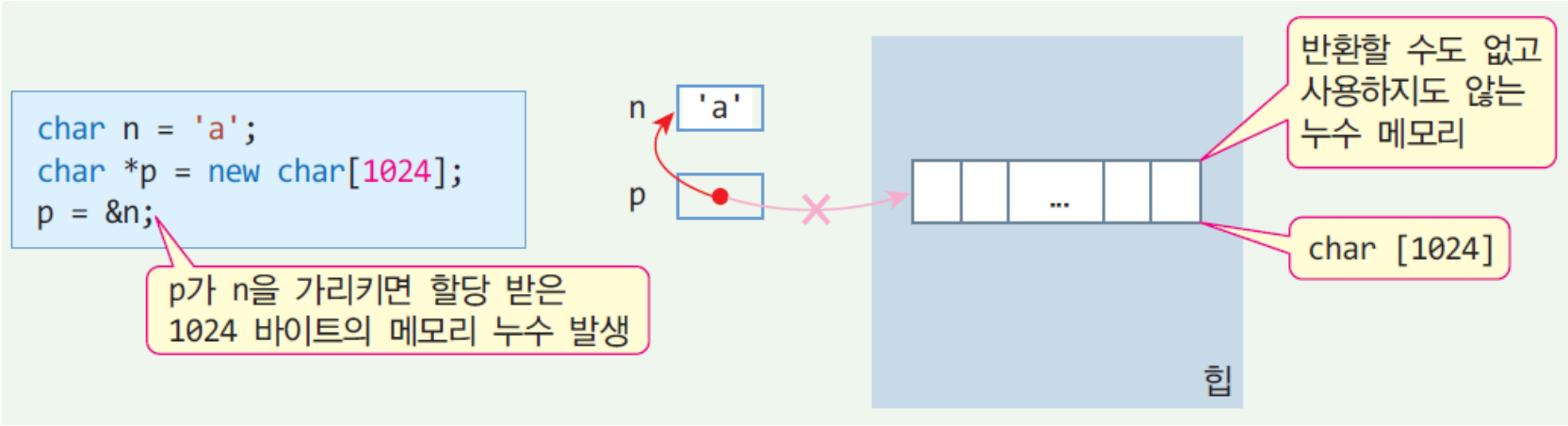
원을 개수를 입력 받고 Circle 배열을 동적 생성하라. 반지름 값을 입력 받아 Circle 배열에 저장하고, 면적이 100에서 200 사이인 원의 개수를 출력하라.

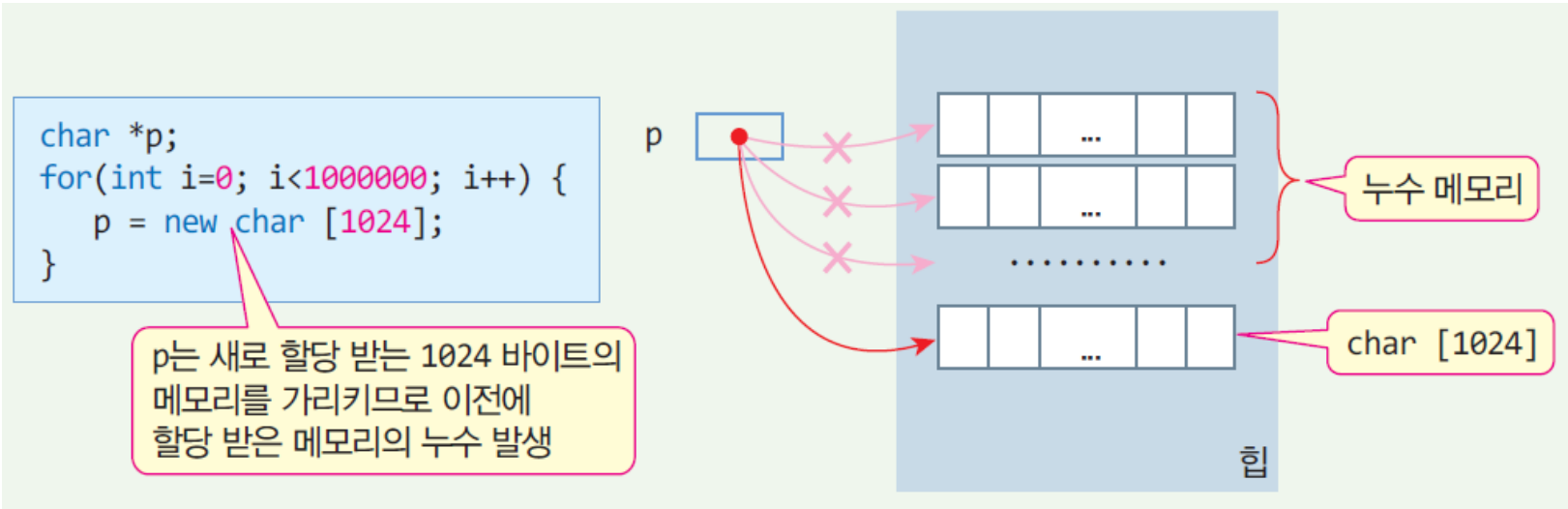
```
#include <iostream>
using namespace std;

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

Circle::Circle() {
    radius = 1;
}
```

```
int main() {
    cout << "생성하고자 하는 원의 개수?";




}
```

```
생성하고자 하는 원의 개수?4
원1: 5
원2: 6
원3: 7
원4: 8
78.5 113.04 153.86 200.96
면적이 100에서 200 사이인 원의 개수는 2
```

동적 메모리 할당과 메모리 누수





* 프로그램이 종료되면, 운영체제는 누수 메모리를 모두 힙에 반환

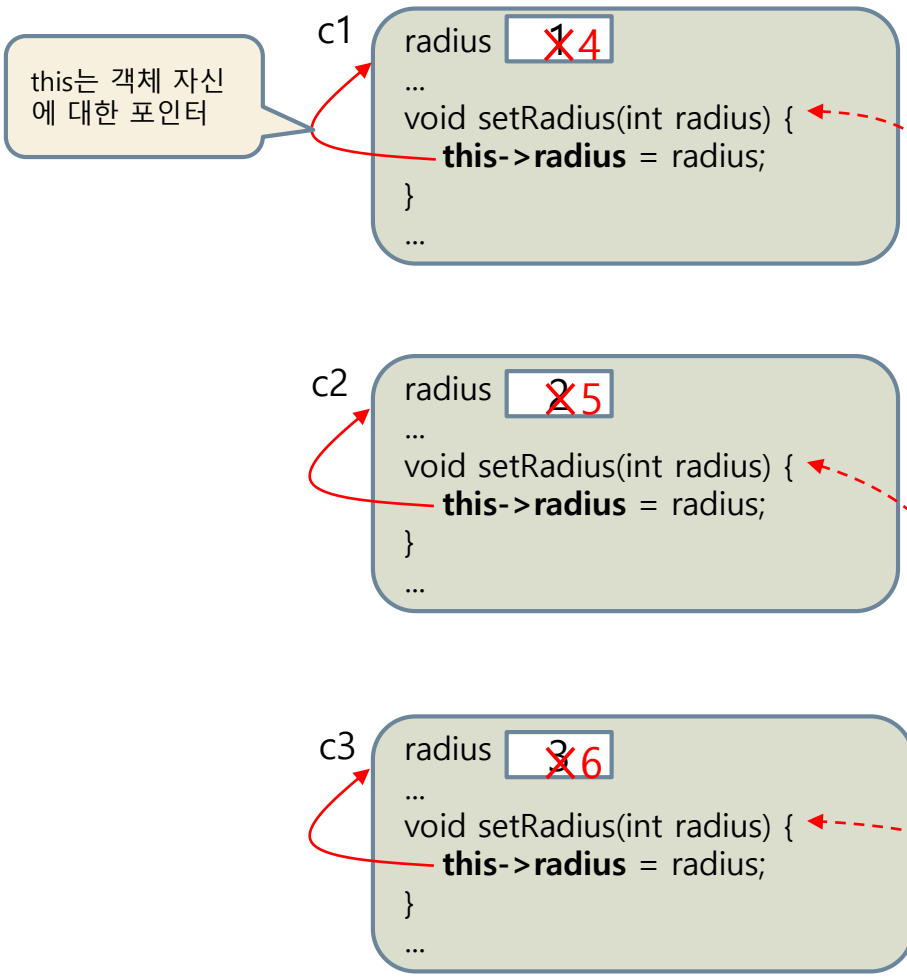
this 포인터

- this
  - 포인터, 객체 자신 포인터
  - 클래스의 멤버 함수 내에서만 사용
  - 개발자가 선언하는 변수가 아니고, 컴파일러가 선언한 변수
    - 멤버 함수에 컴파일러에 의해 묵시적으로 삽입 선언되는 매개 변수

```
class Circle {
   int radius;
public:
   Circle() { this->radius=1; }
   Circle(int radius) { this->radius = radius; }
   void setRadius(int radius) { this->radius = radius; }
   ....
};
```

this와 객체

* 각 객체 속의 this는 다른 객체의 this와 다름

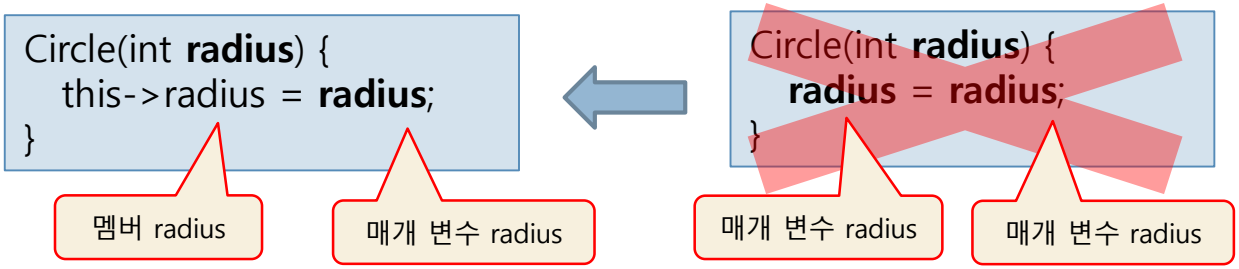



```
class Circle {
   int radius;
public:
   Circle() {
      this->radius=1;
   }
   Circle(int radius) {
      this->radius = radius;
   }
   void setRadius(int radius) {
      this->radius = radius;
   }
};
```

```
int main() {
   Circle c1;
   Circle c2(2);
   Circle c3(3);

   c1.setRadius(4);
   c2.setRadius(5);
   c3.setRadius(6);
}
```

this가 필요한 경우

- 매개변수의 이름과 멤버 변수의 이름이 같은 경우



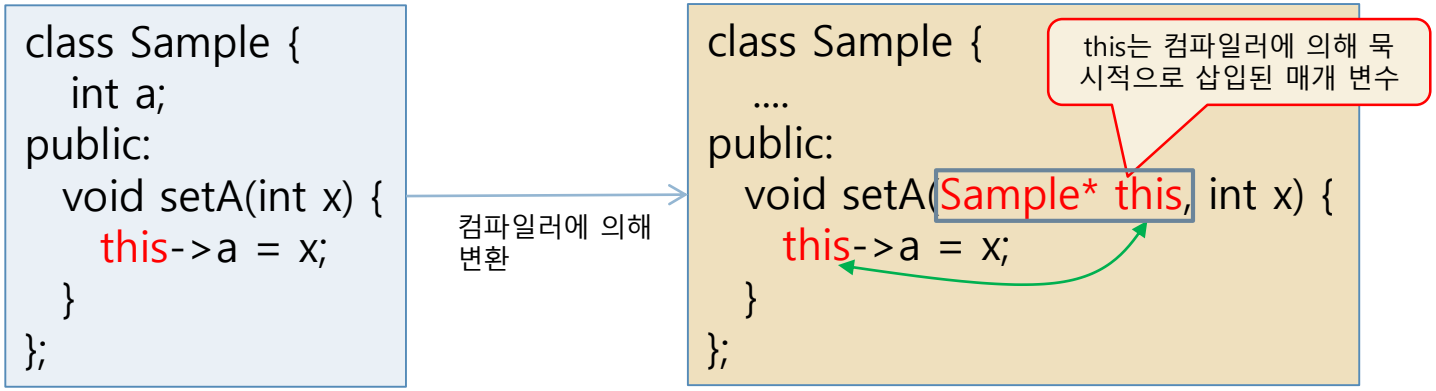
- 멤버 함수가 객체 자신의 주소를 리턴할 때
  - 연산자 중복 시에 매우 필요

```
class Sample {
public:
   Sample* f() {
      ....
      return this;
   }
};
```

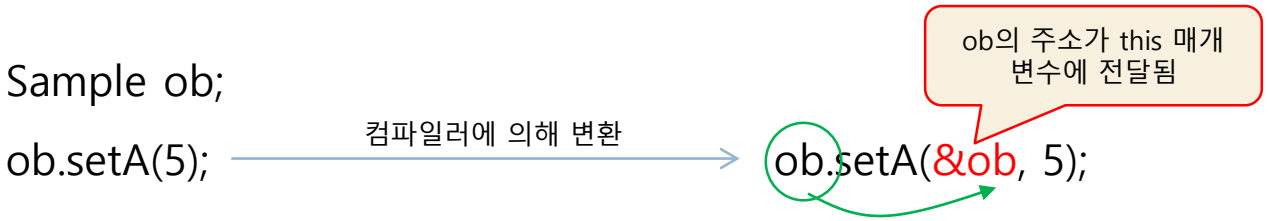
this의 제약 사항

- 멤버 함수가 아닌 함수에서 this 사용 불가
  - 객체와의 관련성이 없기 때문
- static 멤버 함수에서 this 사용 불가
  - 객체가 생기기 전에 static 함수 호출이 있을 수 있기 때문에

this 포인터의 실체 – 컴파일러에서 처리



(a) 개발자가 작성한 클래스 (b) 컴파일러에 의해 변환된 클래스



(c) 객체의 멤버 함수를 호출하는 코드의 변환