객체 포인터

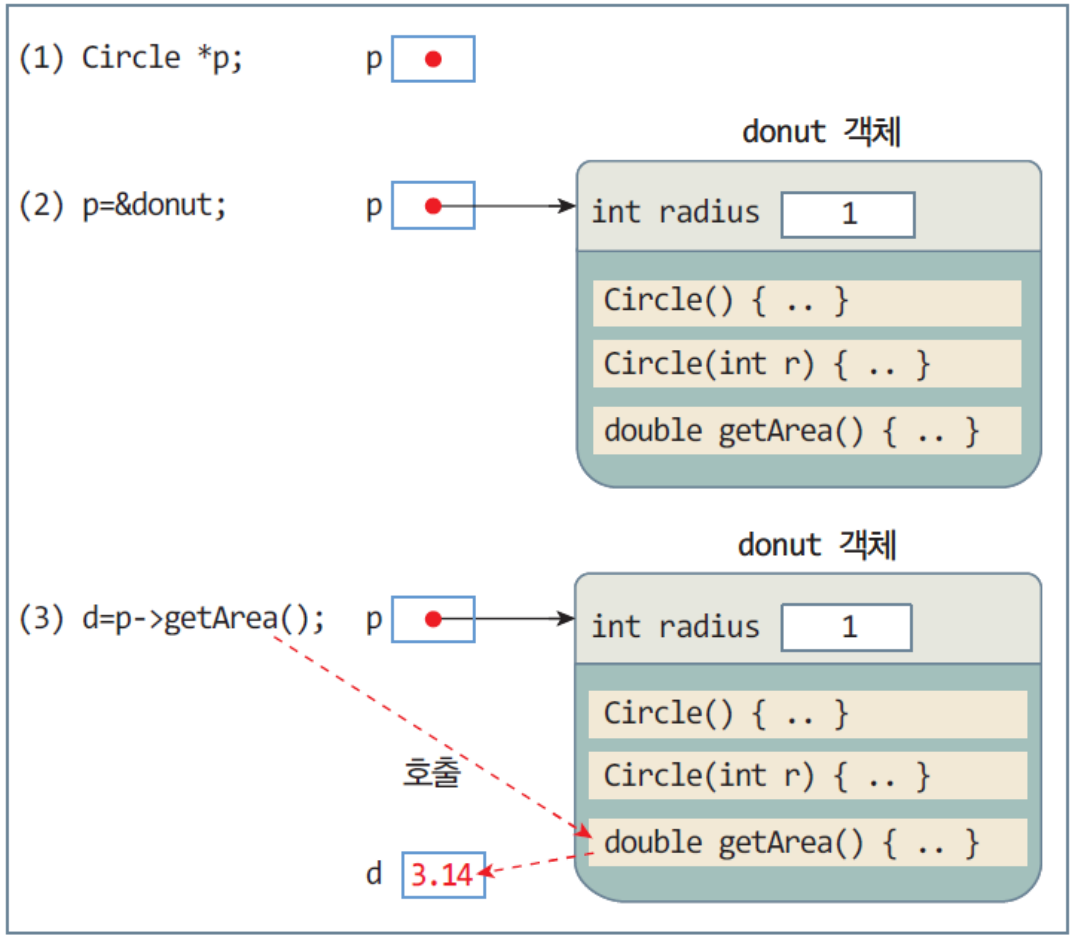
- 객체에 대한 포인터
  - C 언어의 포인터와 동일
  - 객체의 주소 값을 가지는 변수
- 포인터로 멤버를 접근할 때
  - 객체포인터->멤버

```
Circle donut;
double d = donut.getArea();

Circle *p; // (1)
p = &donut; // (2)
d = p->getArea(); // (3)
```

객체에 대한 포인터 선언

포인터에 객체 주소 저장

멤버 함수 호출



객체 배열, 생성 및 소멸

- 객체 배열 선언 가능
  - 기본 타입 배열 선언과 형식 동일
    - int n[3]; // 정수형 배열 선언
    - Circle c[3]; // Circle 타입의 배열 선언
- 객체 배열 선언
  1. 객체 배열을 위한 공간 할당
  2. 배열의 각 원소 객체마다 생성자 실행
     - c[0]의 생성자, c[1]의 생성자, c[2]의 생성자 실행
     - 매개 변수 없는 생성자 호출
  - 매개 변수 있는 생성자를 호출할 수 없음
    - Circle circleArray[3](5); // 오류
- 배열 소멸
  - 배열의 각 객체마다 소멸자 호출. 생성의 반대순으로 소멸
    - c[2]의 소멸자, c[1]의 소멸자, c[0]의 소멸자 실행

배열 생성과 활용(예제 4-2의 실행 과정)

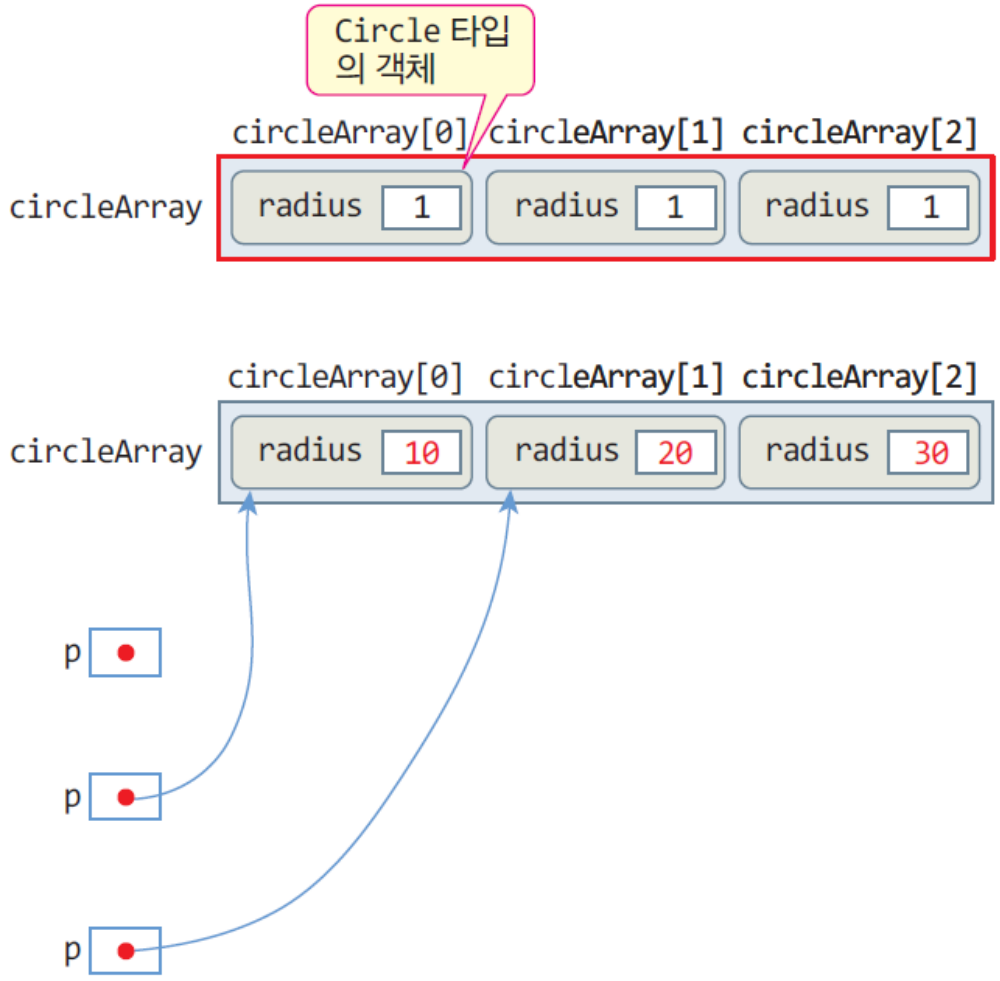
(1) Circle circleArray[3];

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

(3) Circle *p;

(4) p = circleArray;

(5) p++;



객체 배열 초기화

- 객체 배열 초기화 방법
  - 배열의 각 원소 객체당 생성자 지정하는 방법

```
Circle circleArray[3] = { Circle(10), Circle(20), Circle() };
```

  - circleArray[0] 객체가 생성될 때, 생성자 Circle(10) 호출
  - circleArray[1] 객체가 생성될 때, 생성자 Circle(20) 호출
  - circleArray[2] 객체가 생성될 때, 생성자 Circle() 호출

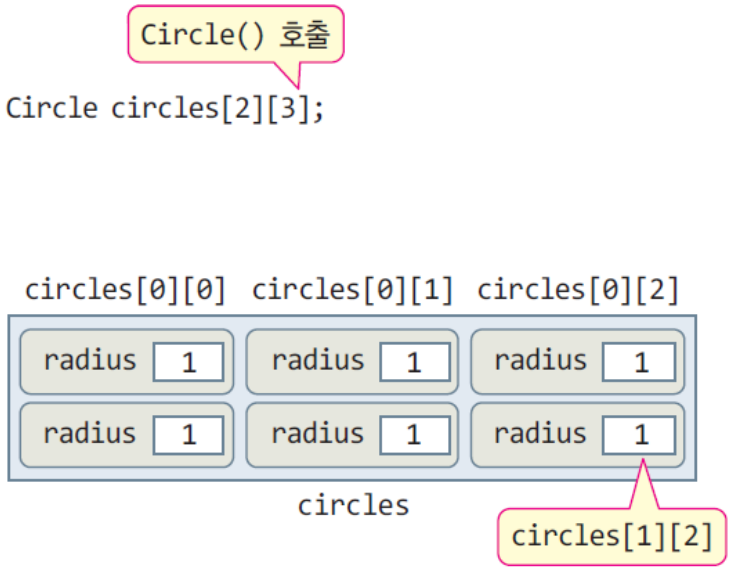
객체 배열 초기화


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

class Circle {
    int radius;
public:
    Circle() {  radius = 1; }
    Circle(int r)  { radius = r; }
    void setRadius(int r)  { radius = r; }
    double getArea();
};

double Circle::getArea() {
    return 3.14*radius*radius;
}

int main() {
    Circle circleArray[3] = { Circle(10), Circle(20), Circle() }; // Circle 배열 초기화

    for(int i=0; i<3; i++)
        cout << "Circle " << i << "의 면적은 " << circleArray[i].getArea() << endl;
}
```

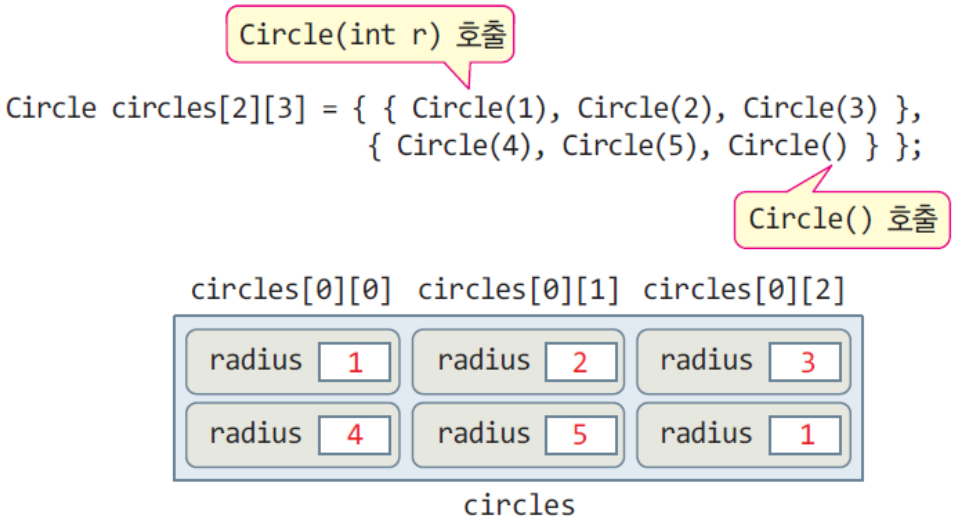
circleArray[0] 객체가 생성될 때, 생성자 Circle(10),
circleArray[1] 객체가 생성될 때, 생성자 Circle(20),
circleArray[2] 객체가 생성될 때, 기본 생성자 Circle()
이 호출된다.

```
Circle 0의 면적은 314
Circle 1의 면적은 1256
Circle 2의 면적은 3.14
```

2차원 배열



(a) 2차원 배열 선언 시



(b) 2차원 배열 선언과 초기화

```
circles[0][0].setRadius(1);
circles[0][1].setRadius(2);
circles[0][2].setRadius(3);
circles[1][0].setRadius(4);
circles[1][1].setRadius(5);
circles[1][2].setRadius(6);
```

2차원 배열을 초기화하는 다른 방식

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

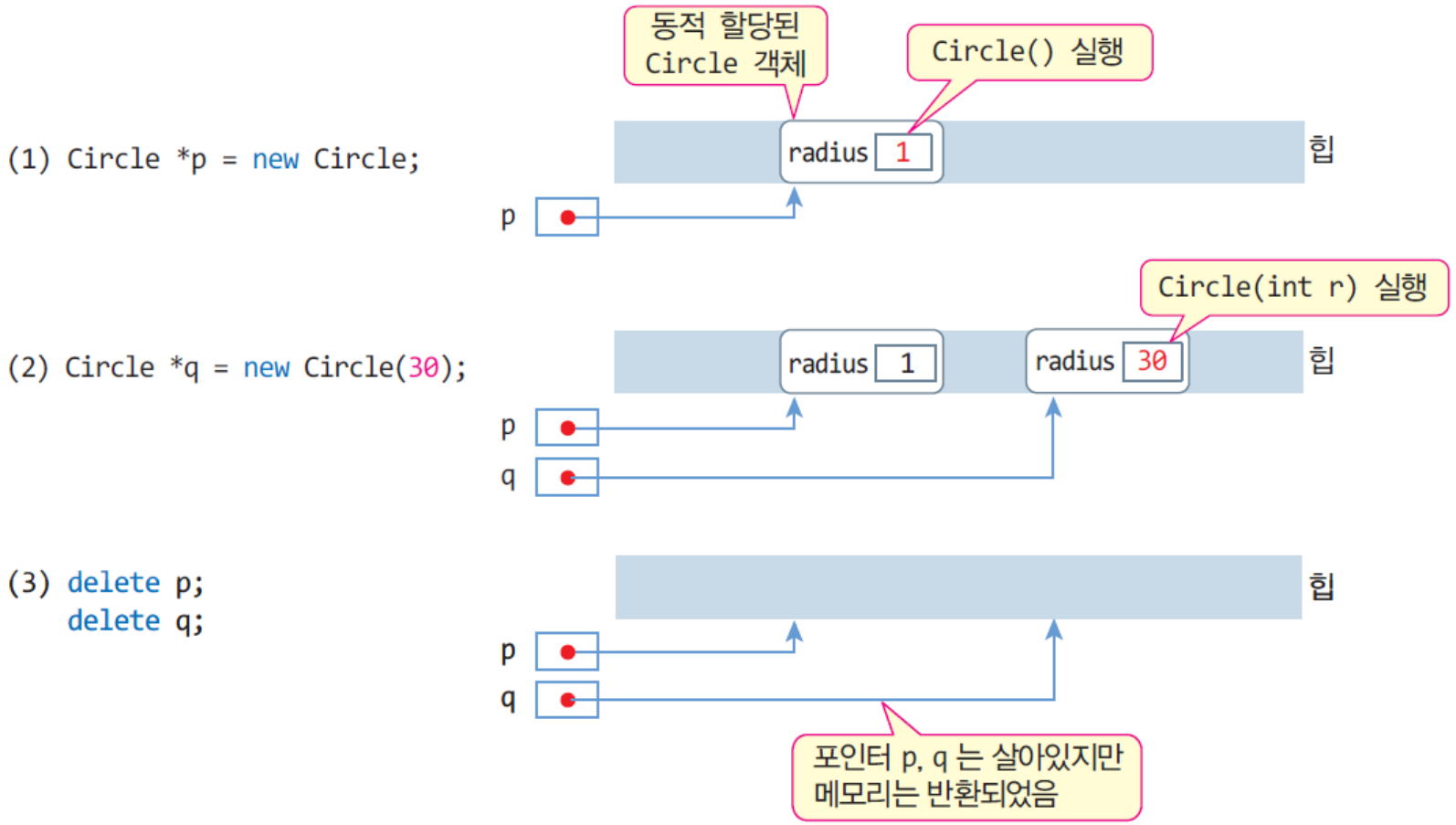
```
int *pArray = new int [10](20); // 구문 오류. 컴파일 오류 발생
int *pArray = new int(20)[10]; // 구문 오류. 컴파일 오류 발생
```

- 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.
```

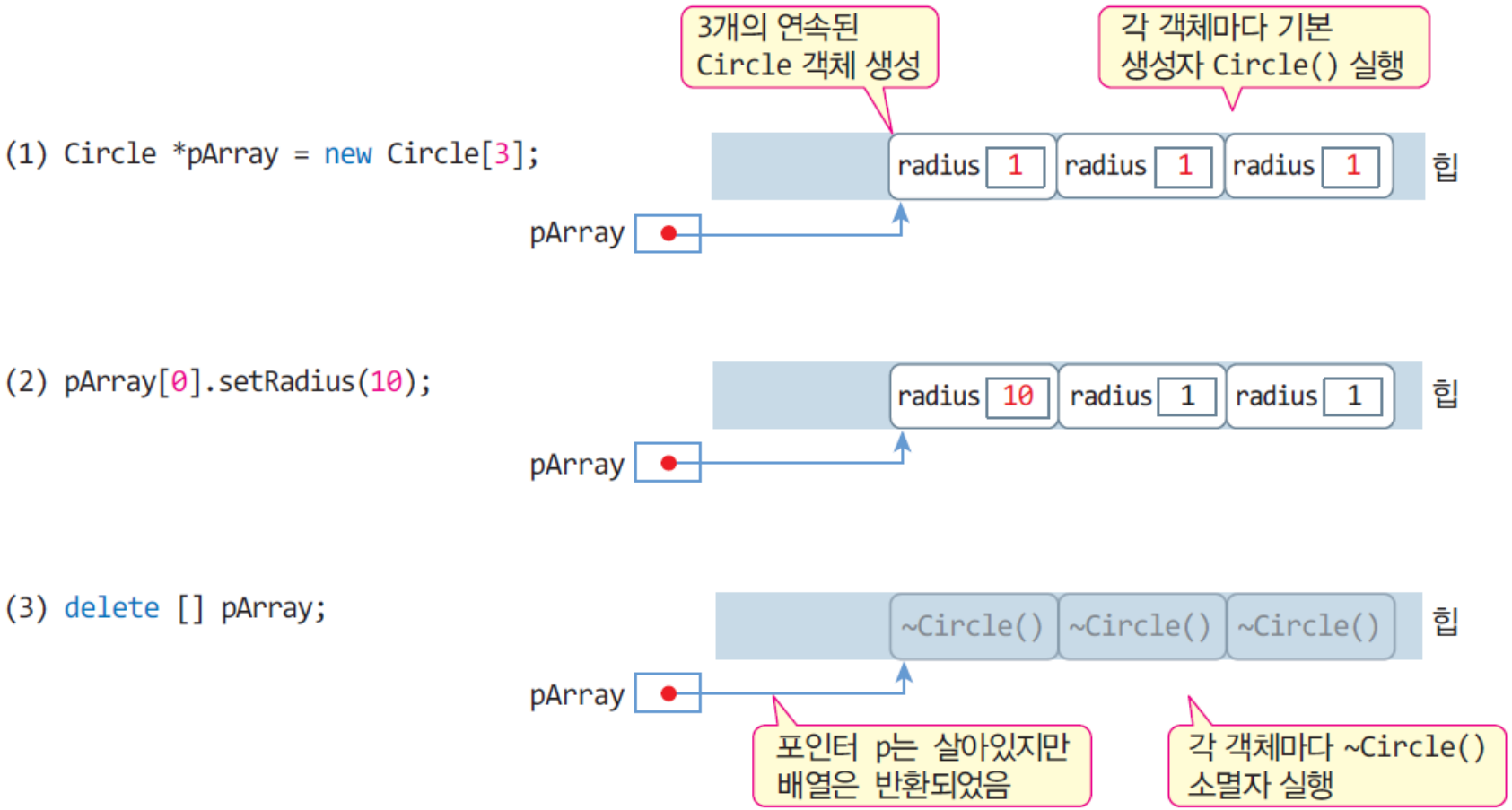
객체의 동적 생성 및 반환

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

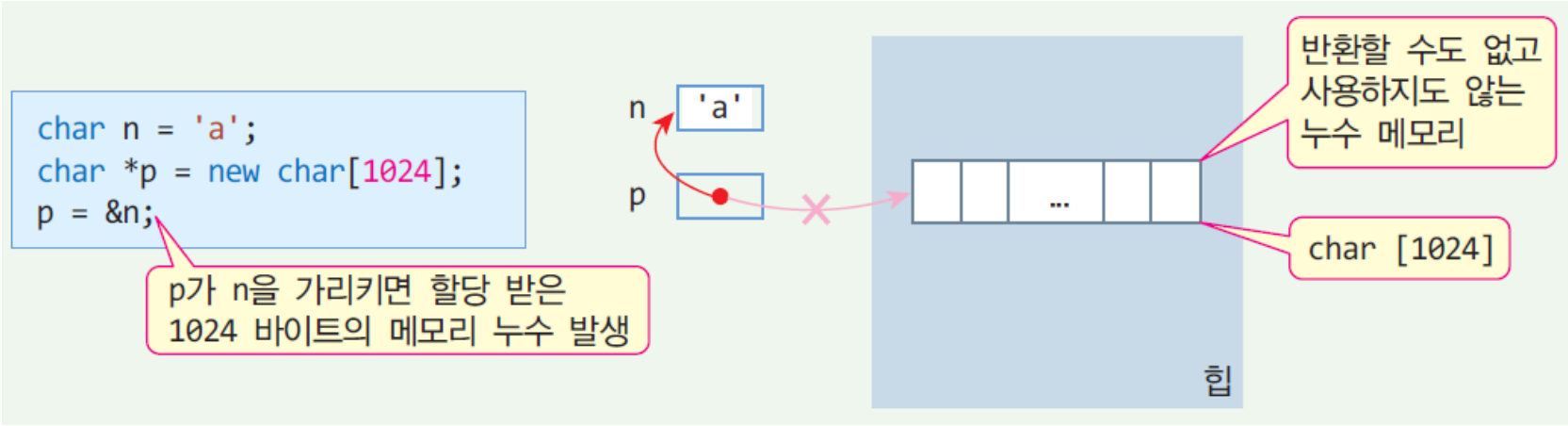


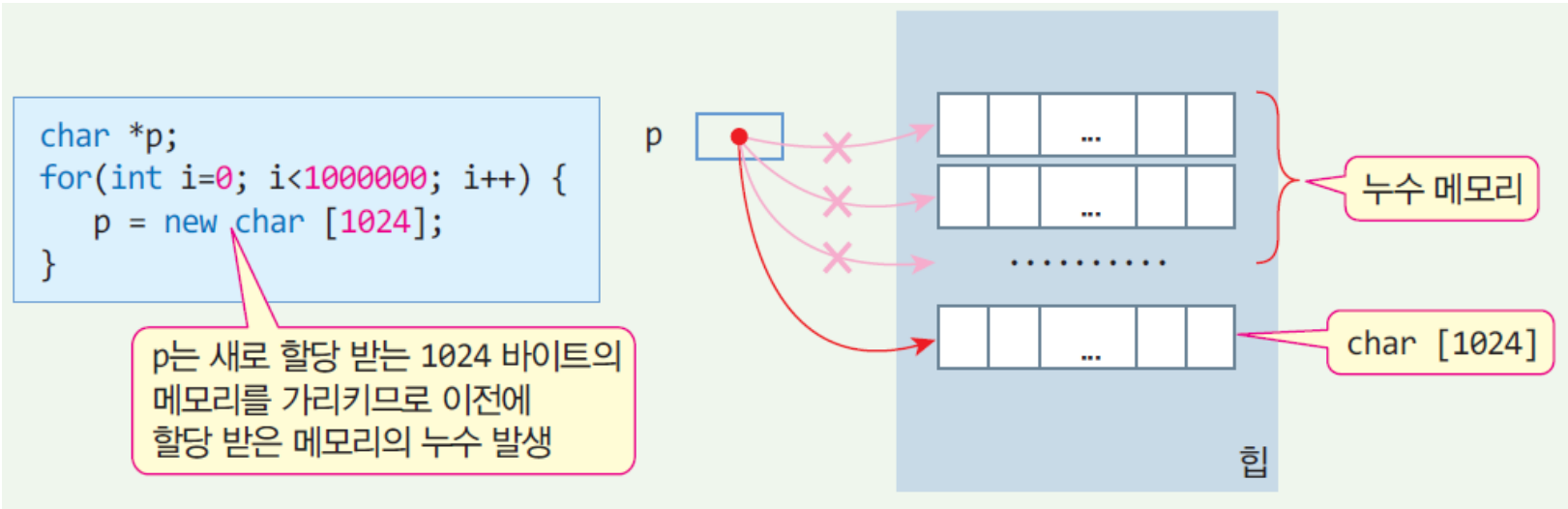
객체 배열의 동적 생성 및 반환

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



동적 메모리 할당과 메모리 누수





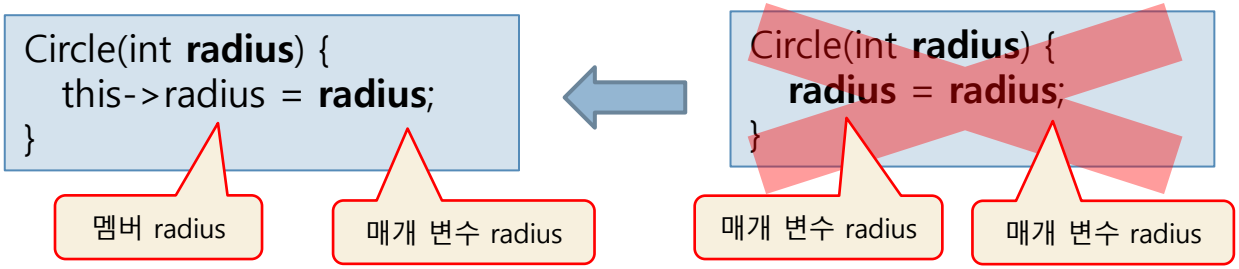
this 포인터

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

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

string 클래스를 이용한 문자열

- C++ 문자열
  - C-스트링
  - C++ string 클래스의 객체
- string 클래스
  - C++ 표준 라이브러리, <string> 헤더 파일에 선언

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

  - 가변 크기의 문자열

```
string str = "I love "; // str은 'I', ' ', 'l', 'o', 'v', 'e', ' '의 7개 문자로 구성
str.append("C++."); // str은 "I love C++."이 된다. 11개의 문자
```

  - 다양한 문자열 연산을 실행하는 연산자와 멤버 함수 포함
    - 문자열 복사, 문자열 비교, 문자열 길이 등
  - 문자열, 스트링, 문자열 객체, string 객체 등으로 혼용

string 객체 생성 및 입출력

- 문자열 생성

```
string str; // 빈 문자열을 가진 스트링 객체
string address("서울시 강남구 역삼동"); // 문자열 리터럴로 초기화
string copyAddress(address); // address를 복사한 copyAddress 생성

// C-스트링(char [] 배열)으로부터 스트링 객체 생성
char text[] = {'L', 'o', 'v', 'e', ' ', 'C', '+', '+', '₩0'};
string title(text); // "Love C++" 문자열을 가진 title 생성
```

- 문자열 출력
  - cout과 << 연산자

```
cout << address << endl; // "서울시 성북구 삼선동 389" 출력
cout << title << endl; // "Love C++" 출력
```

- 문자열 입력
  - cin과 >> 연산자

```
string name;
cin >> name; // 공백이 입력되면 하나의 문자열로 입력
```

- 문자열 숫자 변환
  - stoi() 함수 이용
    - 2011 C++ 표준부터

```
string s="123";
int n = stoi(s); // n은 정수 123. 비주얼 C++ 2010 이상 버전
```

```
string s="123";
int n = atoi(s.c_str()); // n은 정수 123. 비주얼 C++ 2008 이하
```

string 객체의 동적 생성

- new/delete를 이용하여 문자열을 동적 생성/반환 가능

```
string *p = new string("C++"); // 스트링 객체 동적 생성

cout << *p; // "C++" 출력
p->append(" Great!!"); // p가 가리키는 스트링이 "C++ Great!!"이 됨
cout << *p; // "C++ Great!!" 출력

delete p; // 스트링 객체 반환
```