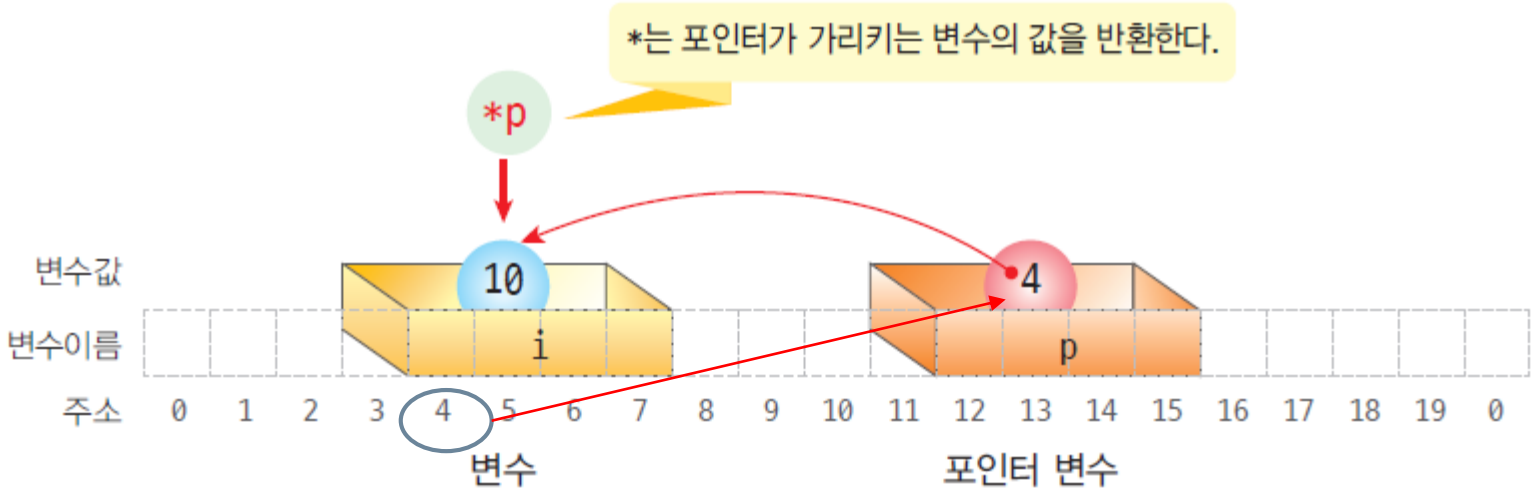
Ch10 - 객체 포인터

간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

```
int i = 10;

int* p;
p = &i;

printf("%d \n", *p);
```



객체 포인터

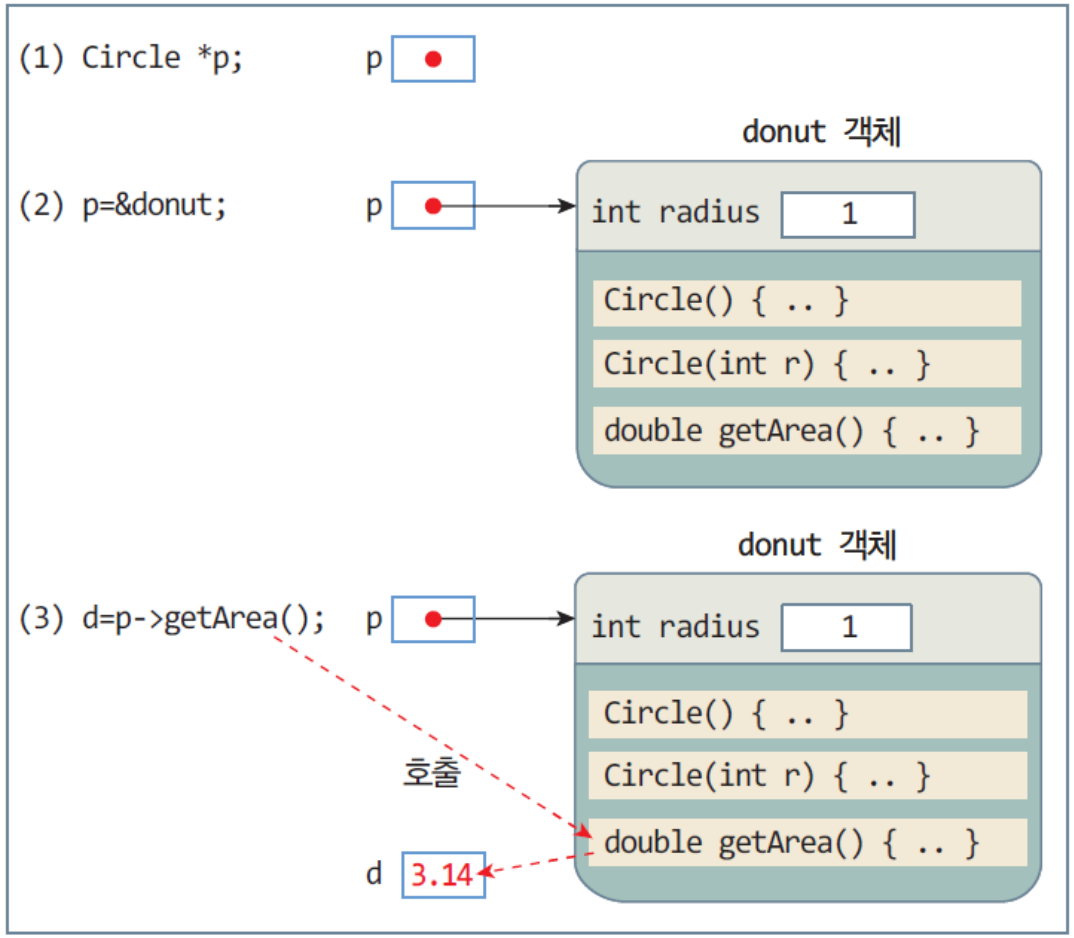
- 객체에 대한 포인터
  - C 언어의 포인터와 동일
  - 객체의 주소 값을 가지는 변수
- 포인터로 멤버를 접근할 때
  - 객체포인터->멤버

```
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



예제 4–1 객체 포인터 선언 및 활용

ex4-1-CirclePointer.cpp

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

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

Circle::Circle() {
   radius = 1;
}
Circle::Circle(int r) {
   radius = r;
}
double Circle::getArea() {
   return 3.14 * radius * radius;
}
```

```
int main() {
   Circle donut;
   Circle pizza(30);

   // 객체 이름으로 멤버 접근
   cout << donut.getArea() << endl;

   // 객체 포인터로 멤버 접근
   Circle *p;
   p = &donut;
   // donut의 getArea() 호출
   cout << p->getArea() << endl;
   cout << (*p).getArea() <<endl;
   // pizza의 getArea() 호출
   p = &pizza;
   cout << p->getArea() << endl;
   cout << (*p).getArea() << endl;
}
```

```
3.14
3.14
3.14
2826
2826
```

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

예제 4– 2 Circle 클래스의 배열 선언 및 활용

ex4-2-CircleArray.cpp

```
class Circle {
    int radius;
public:
    Circle();
    Circle(int r);
    void setRadius(int r);
    double getArea();
};
Circle::Circle() {
    radius = 1;
}
Circle::Circle(int r) {
    radius = r;
}
double Circle::getArea() {
    return 3.14 * radius * radius;
}
void Circle::setRadius(int r) {
    radius = r;
}
```

```
int main() {
    Circle circleArray[3];                          // (1) Circle 객체 배열 생성

    // 배열의 각 원소 객체의 멤버 접근
    circleArray[0].setRadius(10);                   // (2)
    circleArray[1].setRadius(20);
    circleArray[2].setRadius(30);

    for(int i=0; i<3; i++) // 배열의 각 원소 객체의 멤버 접근
        cout << "Circle " << i << "의 면적은 " << circleArray[i].getArea() << endl;

    Circle *p;                                      // (3)
    p = circleArray;                                // (4)
    for(int i=0; i<3; i++) {  // 객체 포인터로 배열 접근
        cout << "Circle " << i << "의 면적은 " << p->getArea() << endl;
        p++;                                        // (5)
    }
}
```

```
Circle 0의 면적은 314
Circle 1의 면적은 1256
Circle 2의 면적은 2826
Circle 0의 면적은 314
Circle 1의 면적은 1256
Circle 2의 면적은 2826
```

배열 생성과 활용(예제 4-2의 실행 과정)

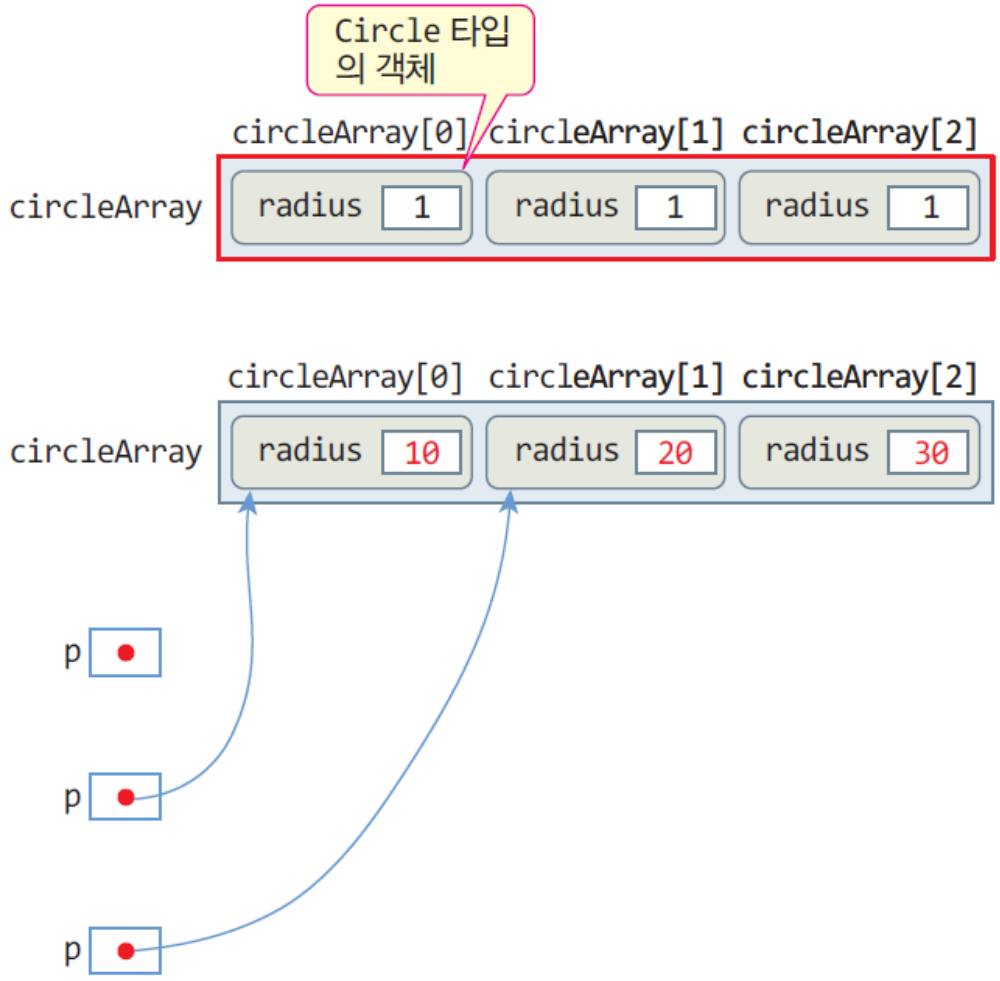
(1) Circle circleArray[3];

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

(3) Circle *p;

(4) p = circleArray;

(5) p++;



객체 배열 생성시 기본 생성자 호출


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

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

int main() {
   Circle circleArray[3];
}
```

컴파일러가 자동으로 기본 생성자 Circle() { } 삽입.
컴파일 오류가 발생하지 않음

기본 생성자 Circle() 호출

(a) 생성자가 선언되어 있지 않은 Circle 클래스

Circle(int r) 호출

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

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

int main() {
   Circle waffle(15);
   Circle circleArray[3];
}
```

기본 생성자 Circle() 호출. 기본 생성자가 없으므로 컴파일 오류

error.cpp(15): error C2512: 'Circle' : 사용할 수 있는 적절한 기본 생성자가 없습니다

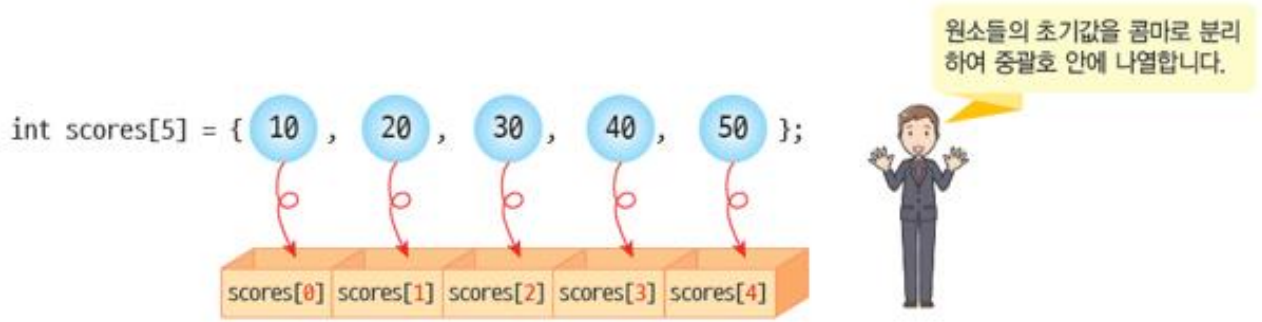
(b) 기본 생성자가 없으므로 컴파일 오류

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



예제 4–3 객체 배열 초기화

ex4-3-CircleArrayInit.cpp

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

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

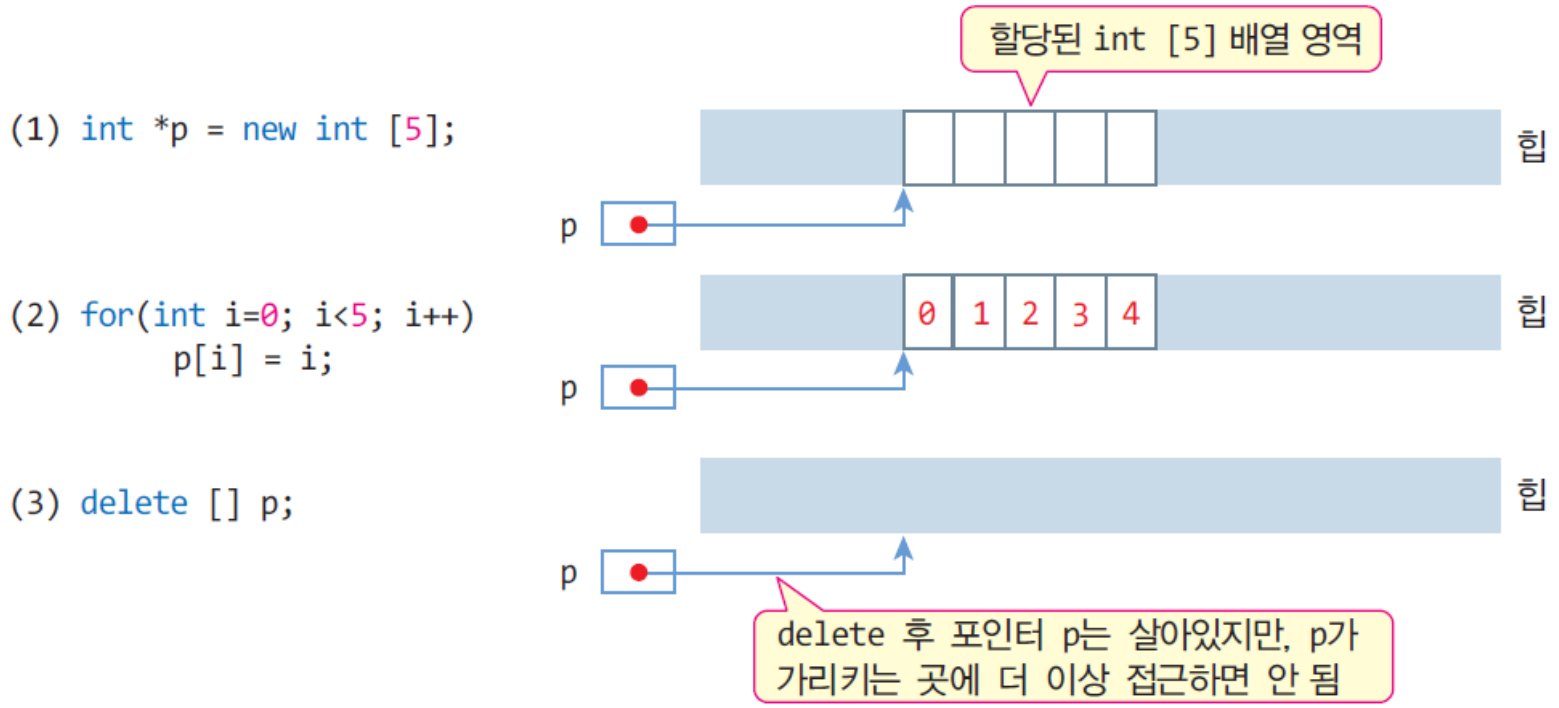
동적 메모리 할당 및 반환

- 정적 할당
  - 변수 선언을 통해 필요한 메모리 할당
    - 많은 양의 메모리는 배열 선언을 통해 할당
- 동적 할당
  - 필요한 양이 예측되지 않는 경우. 프로그램 작성시 할당 받을 수 없음
  - 실행 중에 힙 메모리에서 할당
    - 힙(heap)으로부터 할당
      - 힙은 운영체제가 프로세스(프로그램)의 실행을 시작 시킬 때 동적 할당 공간으로 준 메모리 공간
- C 언어의 동적 메모리 할당 : malloc()/free() 라이브러리 함수 사용
- C++의 동적 메모리 할당/반환
  - new 연산자
    - 기본 타입 메모리 할당, 배열 할당, 객체 할당, 객체 배열 할당
    - 객체의 동적 생성 – 힙 메모리로부터 객체를 위한 메모리 할당 요청
    - 객체 할당 시 생성자 호출
  - delete 연산자
    - new로 할당 받은 메모리 반환
    - 객체의 동적 소멸 – 소멸자 호출 뒤 객체를 힙에 반환

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

ex4-6-newIntArray.cpp

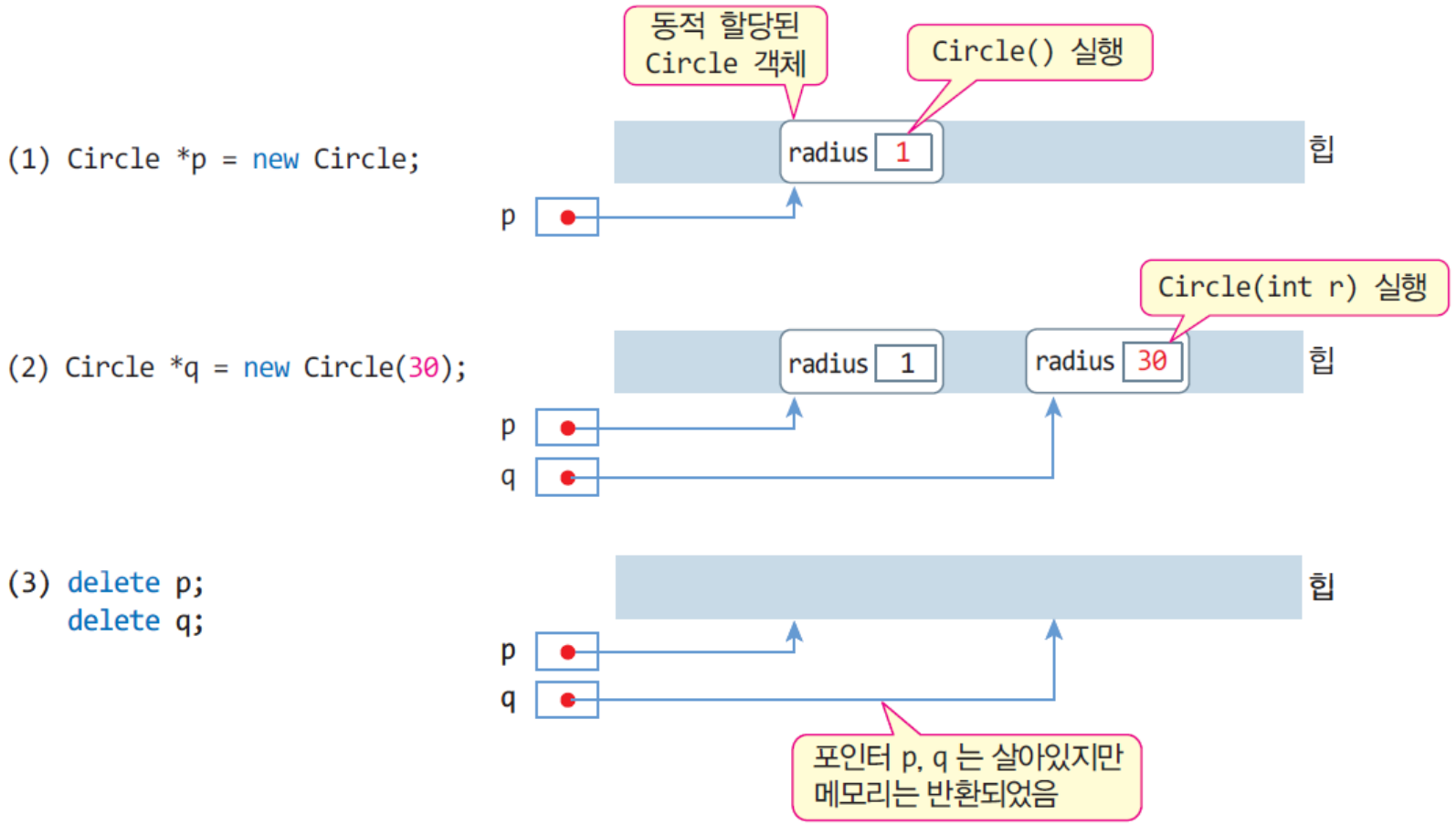
사용자로부터 입력할 정수의 개수를
입력 받아 배열을 동적 할당 받고,
하나씩 정수를 입력 받은 후
합을 출력하는 프로그램을 작성하라.

```
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
}
```

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);
    double getArea();
};

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

Circle::Circle(int r) {
    radius = r;
    cout << "생성자 실행 radius = " << radius << endl;
}
void Circle::setRadius(int r) {
    radius = r;
}
double Circle::getArea() {
    return 3.14 * radius * radius;
}
```

ex4-7-newCircle.cpp

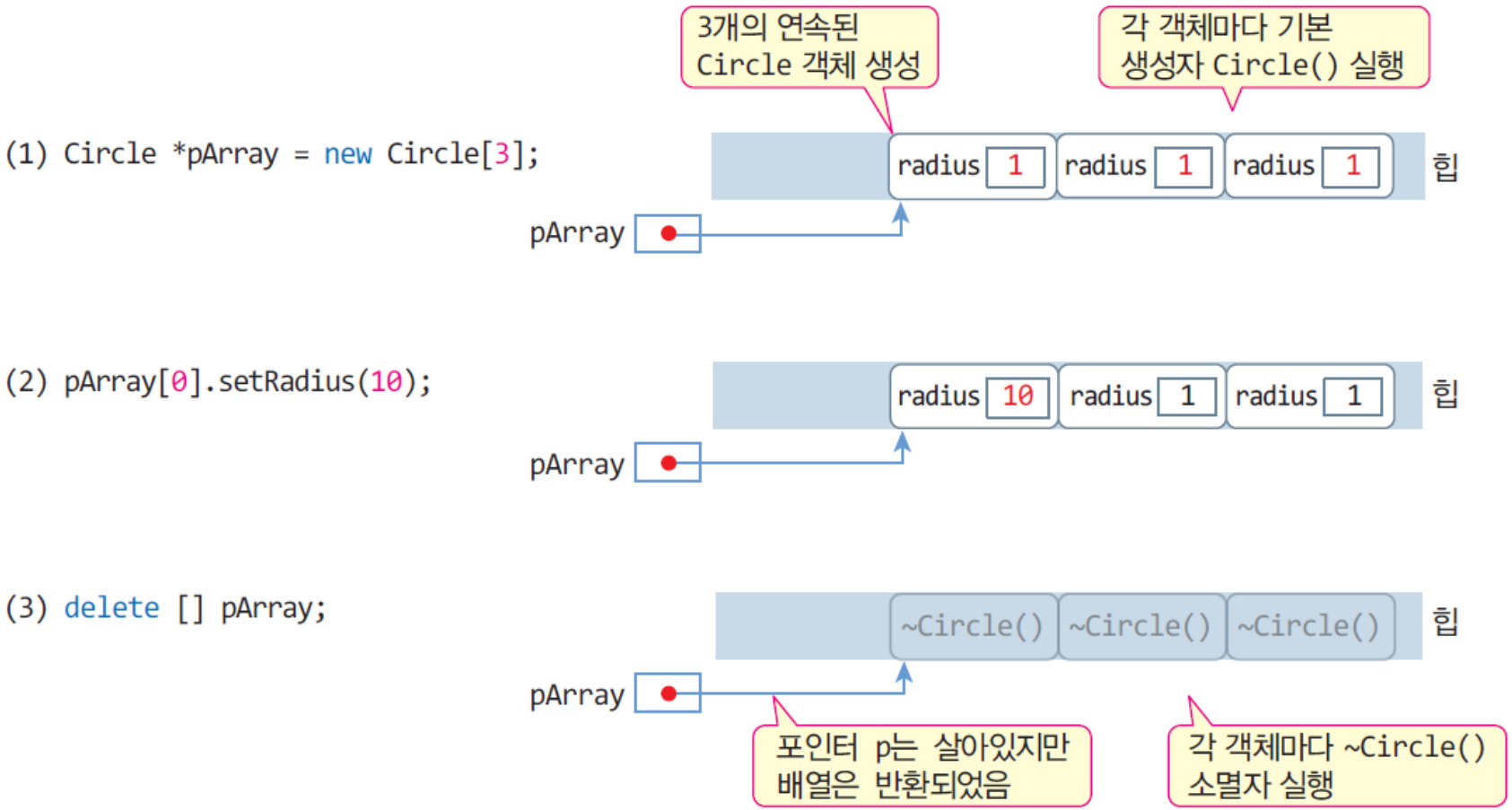
```
int main() {
    Circle *p, *q;
    p = new Circle;
    q = new Circle(30);
    cout << p->getArea() << endl << q->getArea() << endl;
    delete p;
    delete q;
}
```

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
```

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

ex4-9-NewCircleArray.cpp

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


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

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

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

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

Circle::~Circle() {
```

```
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
}
```

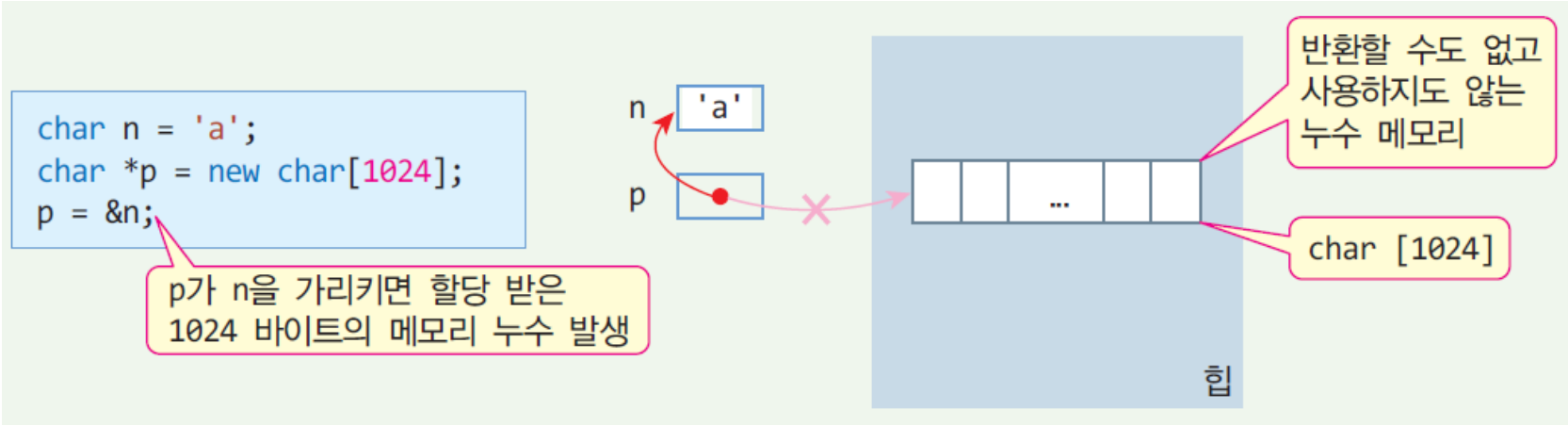
각 원소 객체의 기본 생성자 Circle() 실행

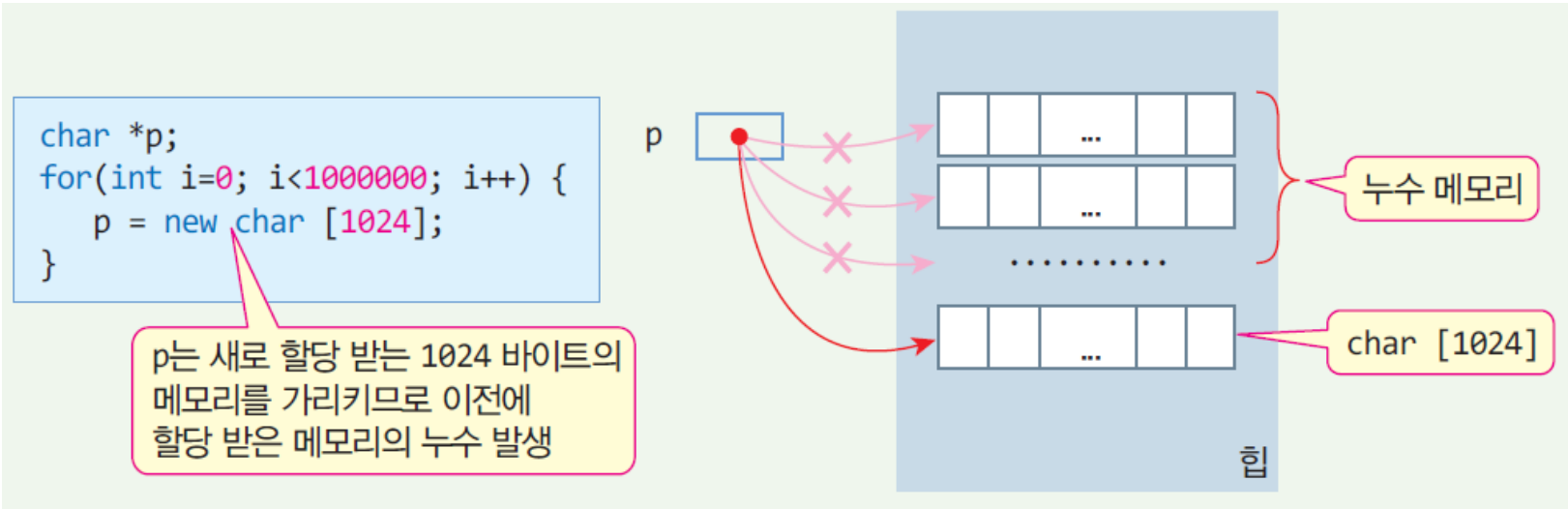
각 배열 원소 객체의 소멸자 ~Circle() 실행

```
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

동적 메모리 할당과 메모리 누수





* 프로그램이 종료되면, 운영체제는 누수 메모리를 모두 힙에 반환

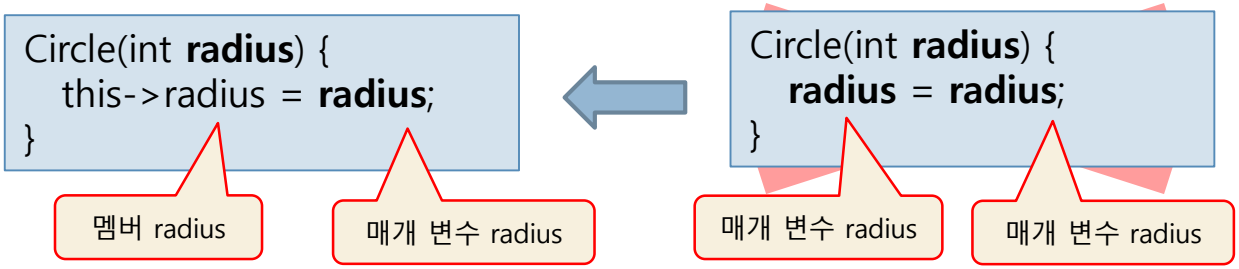
this 포인터

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

this가 필요한 경우

- 매개변수의 이름과 멤버 변수의 이름이 같은 경우



ex4-10_this-1.cpp

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

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

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

Circle::Circle(int r) {
   radius = r;
}

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

Circle::~Circle() {
}
```

ex4-10_this-2.cpp

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

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

Circle::Circle() {
   this->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::~Circle() {
}
```

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

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

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

Circle::Circle(int r) {
    radius = r;
}

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

Circle::~Circle() {
}
```

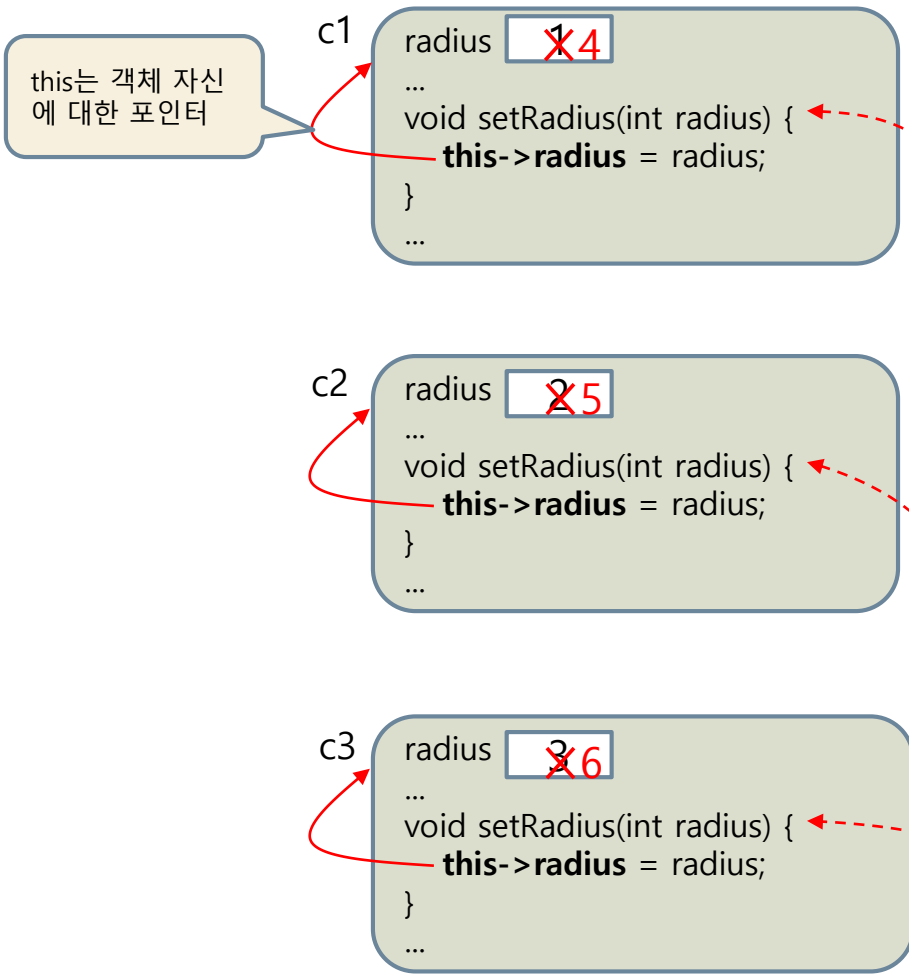
```
int main() {
    Circle c1;
    Circle c2(2);
    Circle c3(3);

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

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

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

Circle::Circle() {
    this->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::~Circle() {
}
```

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

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

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);
}
```

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 성북구 삼선동 389"); // 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 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
```

예제 4–11 string 클래스를 이용한 문자열 생성 및 출력

ex4-11-string.cpp

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

int main() {
   // 스트링 생성
   string str; // 빈 문자열을 가진 스트링 객체 생성
   string address("서울시 성북구 삼선동 389");
   string copyAddress(address); // address의 문자열을 복사한 스트링 객체 생성

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

   // 스트링 출력
   cout << str << endl; // 빈 스트링. 아무 값도 출력되지 않음
   cout << address << endl;
   cout << copyAddress << endl;
   cout << title << endl;
}
```

string 클래스를 사용하기 위해 반드시 필요

빈 문자열을 가진 스트링 출력

```
서울시 성북구 삼선동 389
서울시 성북구 삼선동 389
Love C++
```

예제 4-12 string 배열 선언과 문자열 키 입력 응용

ex4-12-stringInput.cpp

5 개의 string 배열을 선언하고 getline()을 이용하여 문자열을 입력 받아 사전 순으로 가장 뒤에 나오는 문자열을 출력하라. 문자열 비교는 <, > 연산자를 간단히 이용하면 된다.

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

int main() {
    string names[5]; // 문자열 배열 선언

    for(int i=0; i<5; i++) {
        cout << "이름 >> ";
        getline(cin, names[i], '₩n');
    }

    string latter = names[0];
    for(int i=1; i<5; i++) {
        if(latter < names[i]) { // 사전 순으로 latter 문자열이 앞에 온다면
            latter = names[i]; // latter 문자열 변경
        }
    }
    cout << "사전에서 가장 뒤에 나오는 문자열은 " << latter << endl;
}
```

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

예제 4-14 문자열 처리 응용 - 덧셈 문자열을 입력 받아 덧셈 실행

4+125+4+77+102 등으로 표현된 덧셈식을 문자열로 입력받아 계산하는 프로그램 작성하라.

ex4-14-stringAddition.cpp

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

int main() {
    string s;
    cout << "7+23+5+100+25와 같이 덧셈 문자열을 입력하세요." << endl;
    getline(cin, s, '₩n'); // 문자열 입력
    int sum = 0;
    int startIndex = 0; // 문자열 내에 검색할 시작 인덱스
    while(true) {
        int fIndex = s.find('+', startIndex);
        if(fIndex == -1) { // '+' 문자 발견할 수 없음
            string part = s.substr(startIndex);
            if(part == "") break; // "2+3+", 즉 +로 끝나는 경우
            cout << part << endl;
            sum += stoi(part); // 문자열을 수로 변환하여 더하기
            break;
        }
        int count = fIndex - startIndex; // 서브스트링으로 자를 문자 개수
        string part = s.substr(startIndex, count); // startIndex부터 count 개의 문자로 서브스트링 만들기
        cout << part << endl;
        sum += stoi(part); // 문자열을 수로 변환하여 더하기
        startIndex = fIndex+1; // 검색을 시작할 인덱스 전진시킴
    }
    cout << "숫자들의 합은 " << sum;
}
```

```
7+23+5+100+25와 같이 덧셈 문자열을 입력하세요.
66+2+8+55+100
66
2
8
55
100
숫자들의 합은 231
```

예제 4-15 문자열 find 및 replace

&가 입력될 때까지 여러 줄의 영문 문자열을 입력 받고, 찾는 문자열과 대치할 문자열을 각각 입력 받아 문자열을 변경하라.

ex4-15-stringFindAndReplace.cpp

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

int main() {
    string s;
    cout << "여러 줄의 문자열을 입력하세요. 입력의 끝은 &문자입니다." << endl;
    getline(cin, s, '&'); // 문자열 입력
    cin.ignore();
    string f, r;
    cout << endl << "find: ";
    getline(cin, f, '₩n'); // 검색할 문자열 입력
    cout << "replace: ";
    getline(cin, r, '₩n'); // 대치할 문자열 입력

    int startIndex = 0;
    while(true) {
        int fIndex = s.find(f, startIndex); // startIndex부터 문자열 f 검색
        if(fIndex == -1)
            break; // 문자열 s의 끝까지 변경하였음
        s.replace(fIndex, f.length(), r); // fIndex부터 문자열 f의 길이만큼 문자열 r로 변경
        startIndex = fIndex + r.length();
    }
    cout << s << endl;
}
```

& 뒤에 따라 오는 <Enter> 키를 제거하기 위한 코드!!!

예제 4-15 실행 결과

여러 줄의 문자열을 입력하세요. 입력의 끝은 &문자입니다.
It's now or never, come hold me tight. Kiss me my darling, be mine tonight
Tomorrow will be too late. It's now or never, my love won't wait&

& 뒤에<Enter> 키 입력

find: now

검색할 단어

replace: Right Now

대치할 단어

It's Right Now or never, come hold me tight. Kiss me my darling, be mine tonight
Tomorrow will be too late. It's Right Now or never, my love won't wait