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

new와 delete 연산자

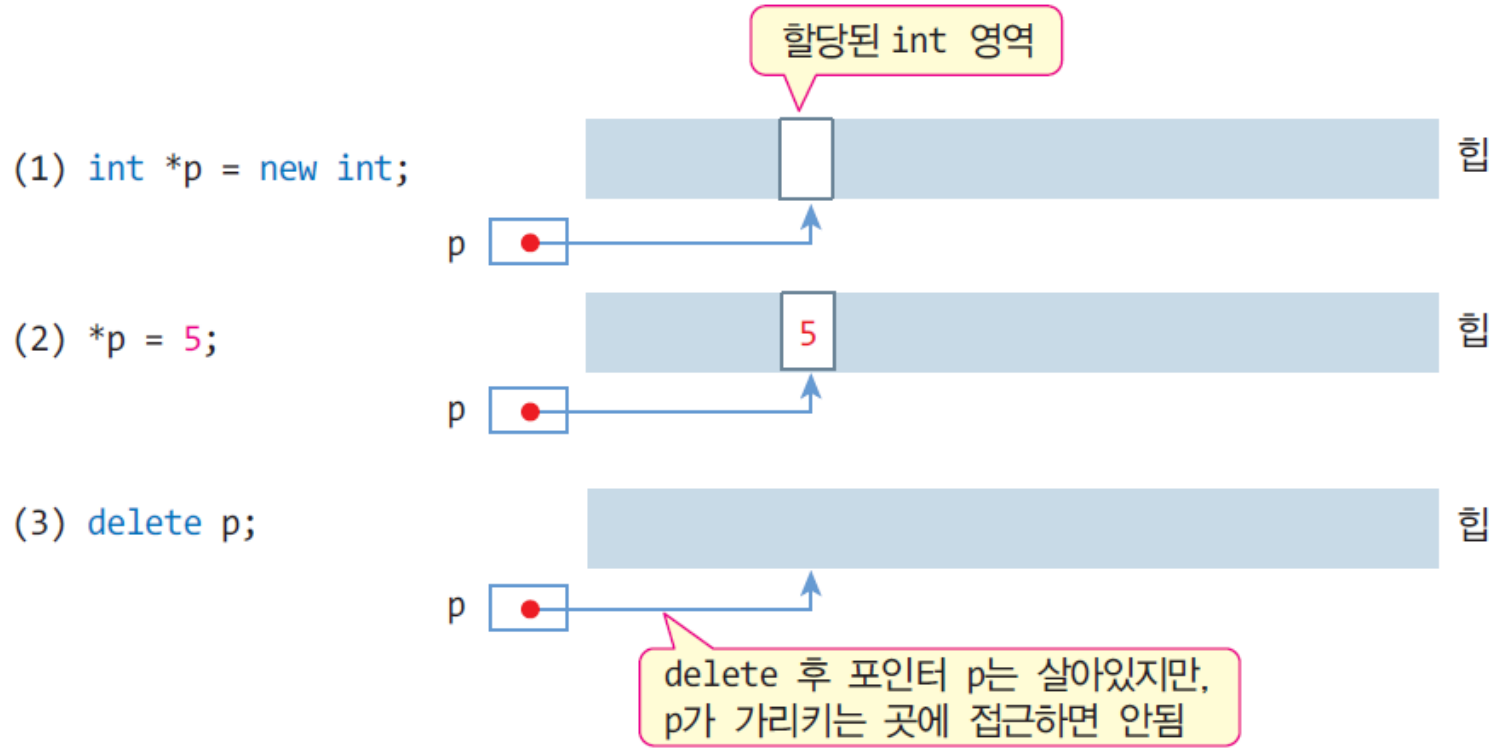
- C++의 기본 연산자
- new/delete 연산자의 사용 형식

```
데이터타입 *포인터변수 = new 데이터타입 ;
delete 포인터변수;
```

- new/delete의 사용

```
int *pInt = new int; // int 타입의 메모리 동적 할당
char *pChar = new char; // char 타입의 메모리 동적 할당
Circle *pCircle = new Circle(); // Circle 클래스 타입의 메모리 동적 할당

delete pInt; // 할당 받은 정수 공간 반환
delete pChar; // 할당 받은 문자 공간 반환
delete pCircle; // 할당 받은 객체 공간 반환
```

기본 타입의 메모리 동적 할당 및 반환



예제 4-5 정수형 공간의 동적 할당 및 반환 예


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

int main() {
   int *p;

   p = new int;
   if(!p) {
      cout << "메모리를 할당할 수 없습니다.";
      return 0;
   }

   *p = 5; // 할당 받은 정수 공간에 5 삽입
   int n = *p;
   cout << "*p = " << *p << '₩n';
   cout << "n = " << n << '₩n';

   delete p;
}
```

int 타입 1개 할당

p 가 NULL이면, 메모리 할당 실패

할당 받은 메모리 반환

```
*p = 5
n = 5
```

delete 사용 시 주의 사항

- 적절치 못한 포인터로 delete하면 실행 시간 오류 발생
  - 동적으로 할당 받지 않는 메모리 반환 – 오류

```
int n;
int *p = &n;
delete p; // 실행 시간 오류
// 포인터 p가 가리키는 메모리는 동적으로 할당 받은 것이 아님
```

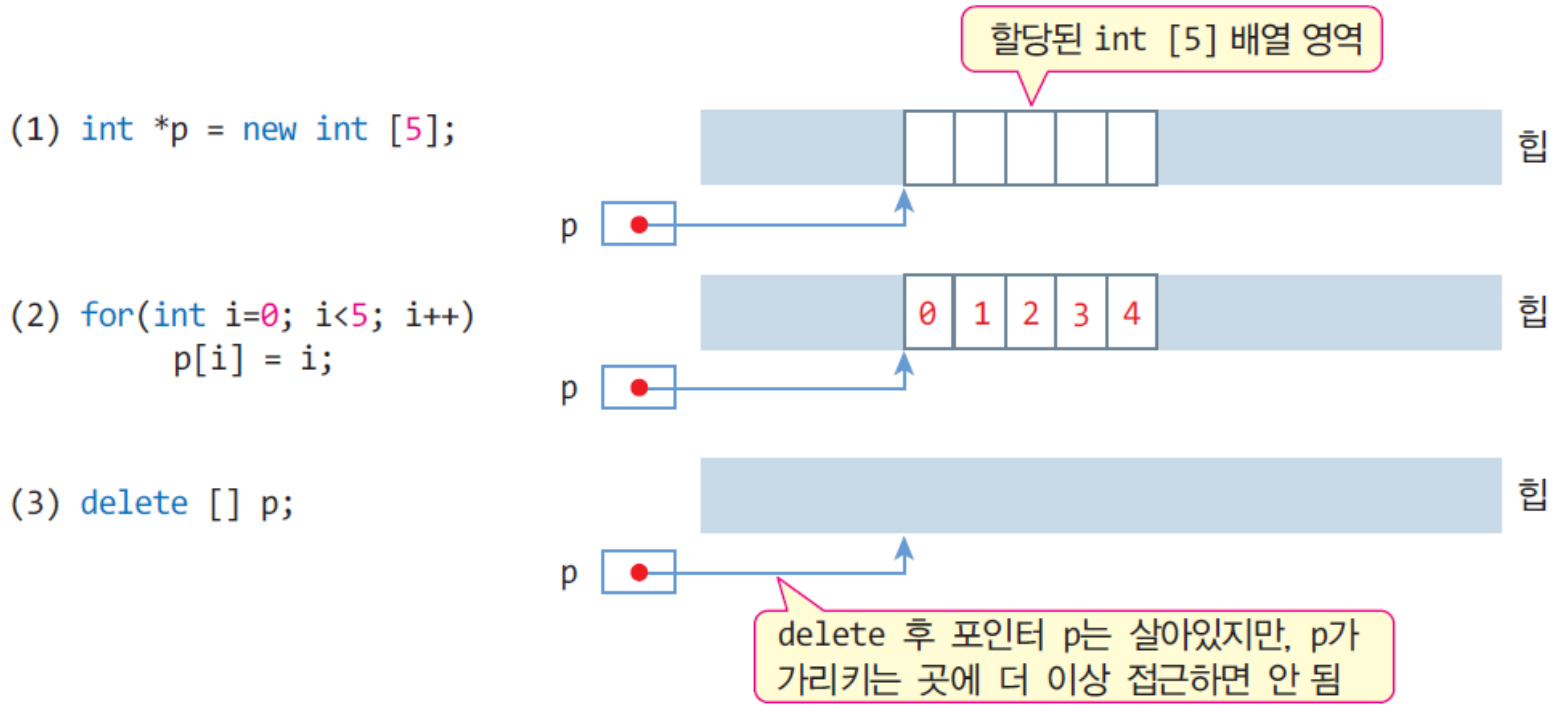
  - 동일한 메모리 두 번 반환 - 오류

```
int *p = new int;
delete p; // 정상적인 메모리 반환
delete p; // 실행 시간 오류. 이미 반환한 메모리를 중복 반환할 수 없음
```

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

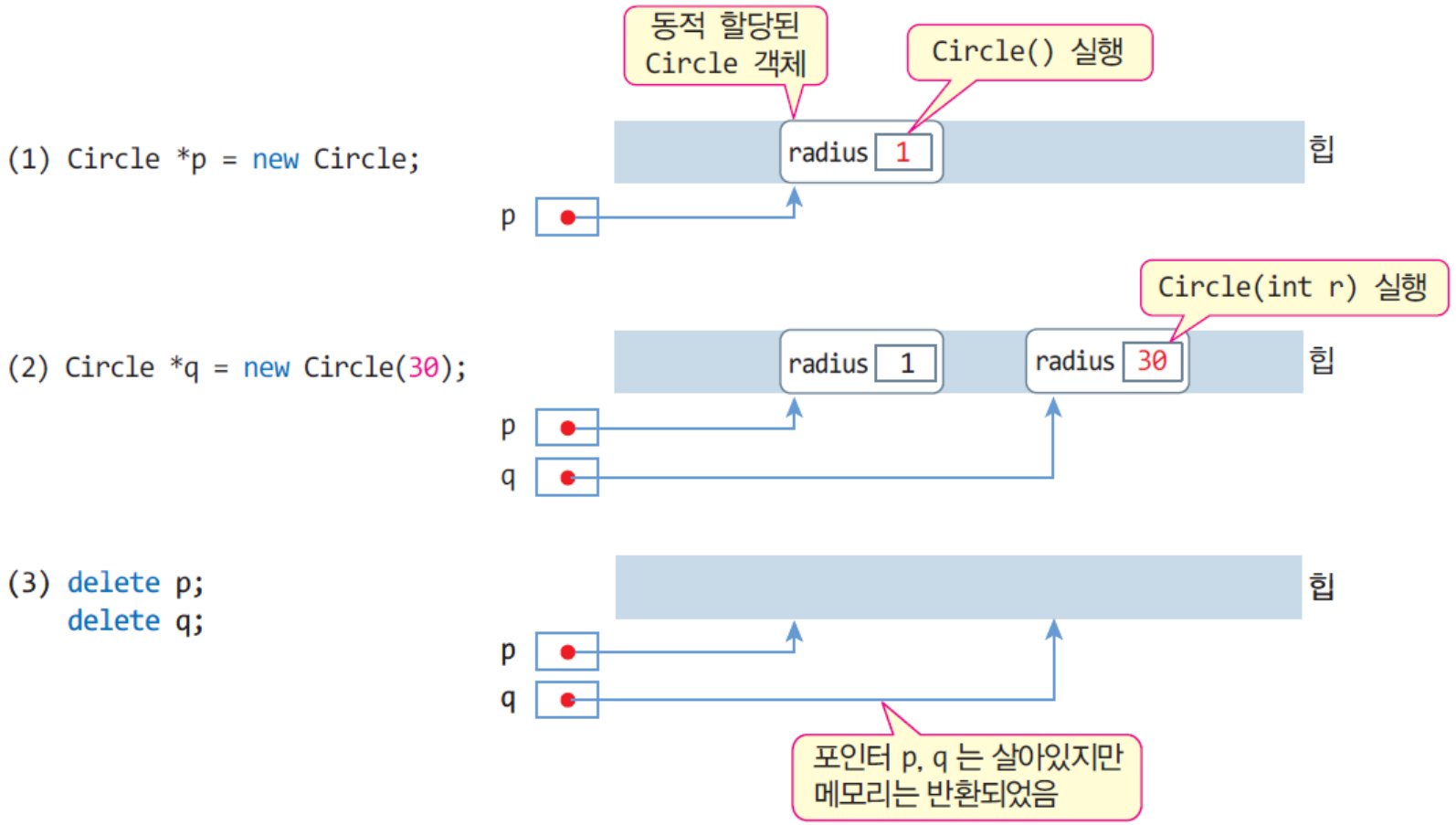
```
int *pArray = new int [10](20); // 구문 오류. 컴파일 오류 발생
int *pArray = new int(20)[10]; // 구문 오류. 컴파일 오류 발생
```

- 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() {
    int radius;
    while(true) {
        cout << "정수 반지름 입력(음수이면 종료)>> ";
        cin >> radius;
        if(radius < 0) break; // 음수가 입력되어 종료한다.
        Circle *p = new Circle(radius); // 동적 객체 생성
        cout << "원의 면적은 " << p->getArea() << endl;
        delete p; // 객체 반환
    }
}
```

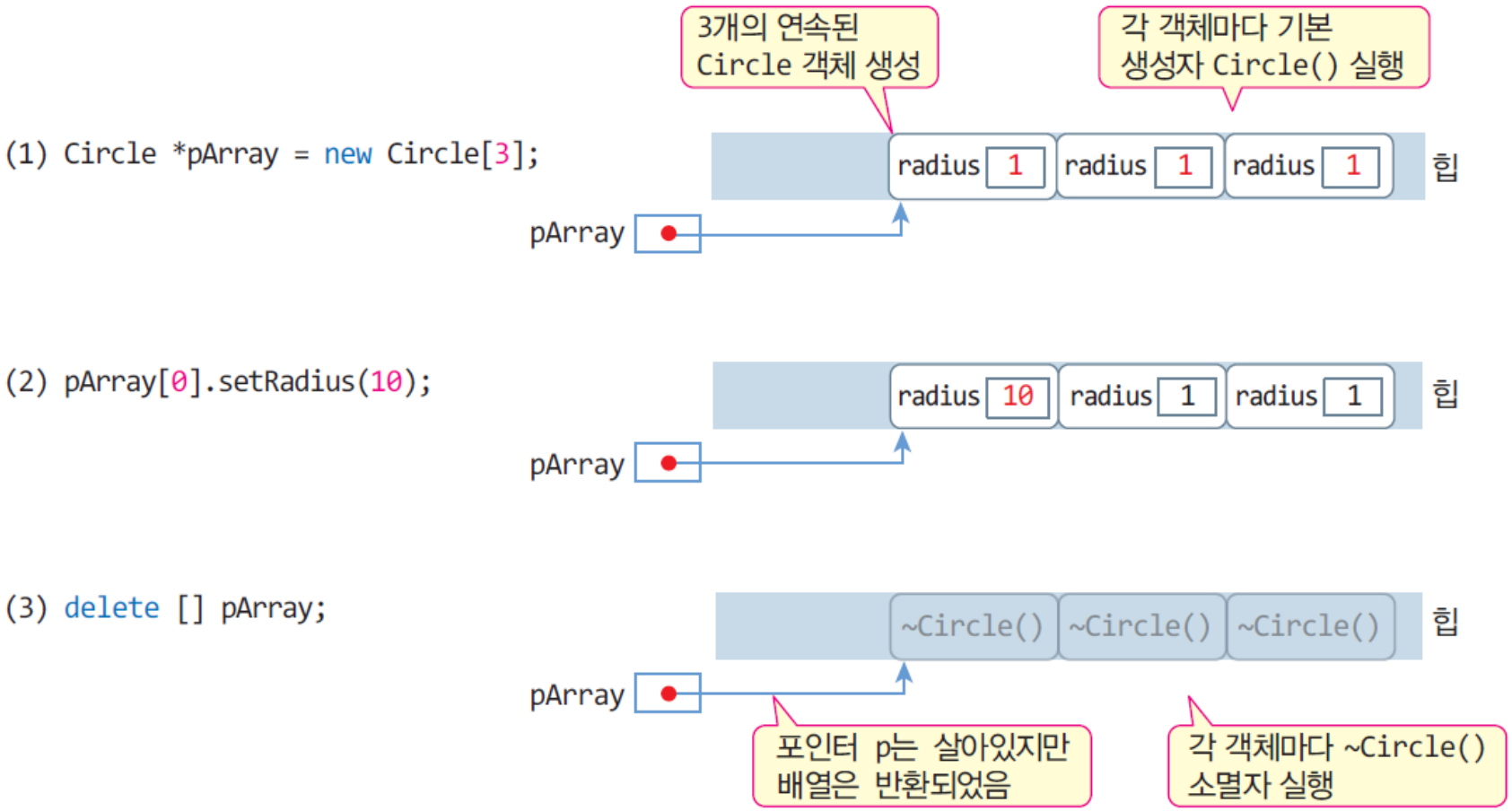
delete 문이 없다면 메모리 누수 발생

```
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

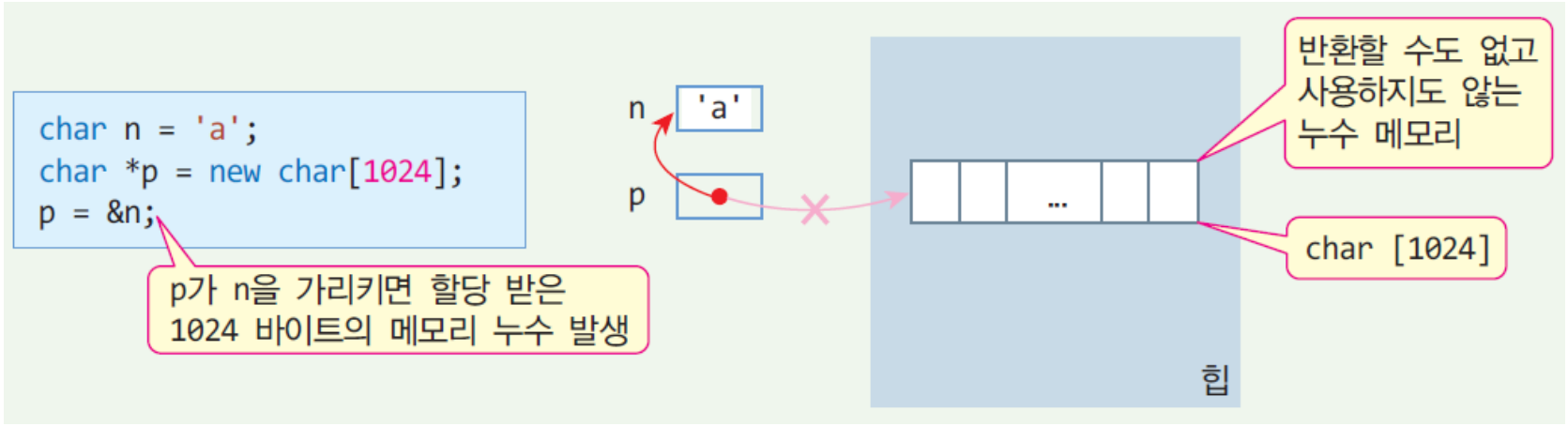
원을 개수를 입력 받고 Circle 배열을 동적 생성하라. 반지름 값을 입력 받아 Circle 배열에 저장하고, 면적이 100에서 200 사이인 원의 개수를 출력하라.

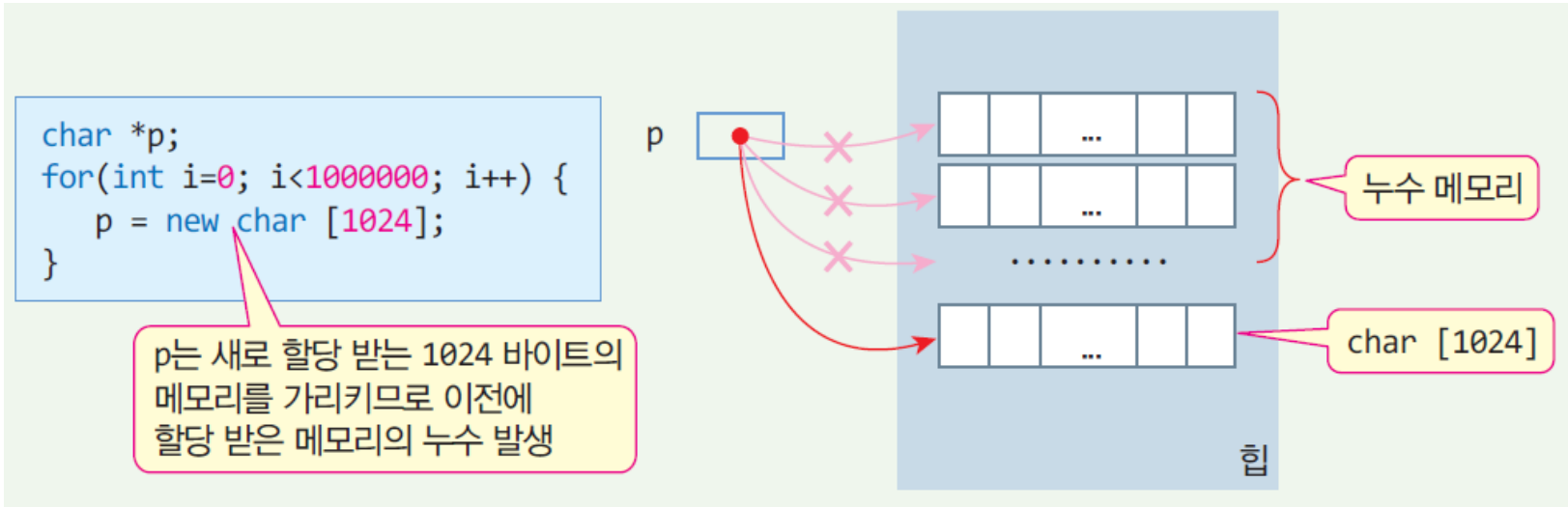
```
#include <iostream>
using namespace std;

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?";
    int n, radius;
    cin >> n; // 원의 개수 입력

    Circle *pArray = new Circle [n]; // n 개의 Circle 배열 생성
    for(int i=0; i<n; i++) {
        cout << "원" << i+1 << ": "; // 프롬프트 출력
        cin >> radius; // 반지름 입력
        pArray[i].setRadius(radius); // 각 Circle 객체를 반지름으로 초기화
    }

    int count =0; // 카운트 변수
    Circle* p = pArray;
    for(int i=0; i<n; i++) {
        cout << p->getArea() << ' '; // 원의 면적 출력
        if(p->getArea() >= 100 && p->getArea() <= 200)
            count++;
        p++;
    }
    cout << endl << "면적이 100에서 200 사이인 원의 개수는 "
            << count << endl;

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

```
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