




# 고급프로그래밍

## 객체의 생성과 사용

Professor Jeong, Mun-Ho

Robot Vision & Intelligence Laboratory  
Kwangwoon University  
(02-940-5625, mhjeong@kw.ac.kr)

# Schedule

week	Topics		Homework	Quiz
1	과목소개	교과목 소개 (1), C++ 시작 (2)		
2	C++ 	C++ 프로그래밍의 기본(3, 3/12), 클래스와 객체(4, 3/14)	1	1
3		휴강(3/17), 객체생성과 사용(5, 3/19)	2	
4		함수와 참조, 복사 생성자와 함수중복, static, friend, 연산자 중복	3	2, 3
5		상속, 가상함수와 추상클래스	4	4
6		템플릿과 STL, 표준 입출력	5	5
7		파일 입출력		
8				
9	C++	예외처리 및 C 사용, 람다식	6	6
10		멀티스레딩	7	7
11		멀티스레딩, 고급문법	8	8
12		고급문법	9	9
13	병렬 프로그래밍	병렬프로그래밍		
14		병렬프로그래밍		
15	기말고사			

실스 3

- 아래와 같이 한 라인에 ' ; '으로 3 개의 이름을 구분하여 입력받아 각 이름을 출력하는 프로그램이다. 밑줄 친 곳에 코드를 삽입하여 완성하시오

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string name;
    cout << ">> ";
    for(int i=0; i<3; i++)
    {
        
        cout << i+1 << " : " << name << endl;
    }
    return 0;
}
```

```
>> Tom;Jane;Brown;
1 : Tom
2 : Jane
3 : Brown
```

# 오늘의 학습내용

- 객체의 생성과 사용

# 객체 포인터

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

The diagram illustrates three key operations in C++ related to object pointers, each with a callout box:

- 객체에 대한 포인터 선언** (Declaration of pointer for object): Points to the line `Circle *p; // (1)`.
- 포인터에 객체 주소 저장** (Storing object address in pointer): Points to the line `p = &donut; // (2)`.
- 멤버 함수 호출** (Member function call): Points to the line `d = p->getArea(); // (3)`.

The code snippets are as follows:

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

# 예제 - 객체 포인터 선언 및 활용

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

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

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;
    cout << p->getArea() << endl; // donut의 getArea() 호출
    cout << (*p).getArea() << endl; // donut의 getArea() 호출

    p = &pizza;
    cout << p->getArea() << endl; // pizza의 getArea() 호출
    cout << (*p).getArea() << endl; // pizza의 getArea() 호출
}
```

```
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

- 객체 배열을 위한 공간 할당
- 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 - 클래스의 배열 선언 및 활용

```
#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 객체 배열 사용

    // 배열의 각 원소 객체의 멤버 접근
    circleArray[0].setRadius(10);
    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;                       // 객체 포인터 변수 사용
    p = circleArray;
    for(int i=0; i<3; i++) { // 객체 포인터로 배열 접근
        cout << "Circle " << i << "의 면적은 " << p->getArea() << endl;
        p++;
    }
}
```

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



# 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) 기본 생성자 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];
}
```

Circle(int r)  
호출

기본 생성자 Circle() 호출.  
기본 생성자가 없으므로 컴  
파일 오류

(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 - 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;
}
```

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.

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

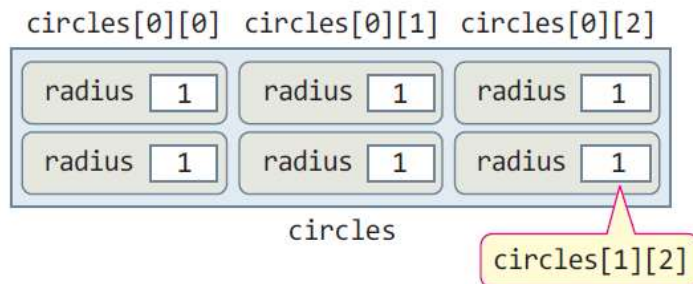
    for(int i=0; i<3; i++)
        cout << "Circle " << i << "의 면적은 " << circleArray[i].getArea() << endl;
}
```

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

# 2차원 배열

Circle() 호출

```
Circle circles[2][3];
```

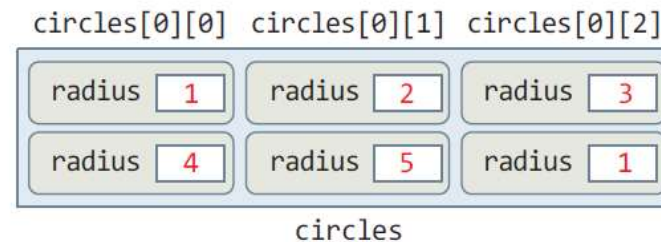


(a) 2차원 배열 선언 시

Circle(int r) 호출

```
Circle circles[2][3] = { { Circle(1), Circle(2), Circle(3) },  
                          { Circle(4), Circle(5), Circle() } };
```

Circle() 호출



(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

# 예제 - 클래스의 2차원 배열 활용

```
#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 circles[2][3];
```

```
    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);
```

Circle circles[2][3] =  
{ { Circle(1), Circle(2), Circle(3) },  
 { Circle(4), Circle(5), Circle() } };

```
    for(int i=0; i<2; i++) // 배열의 각 원소 객체의 멤버 접근
        for(int j=0; j<3; j++) {
            cout << "Circle [" << i << ", " << j << "]의 면적은 ";
            cout << circles[i][j].getArea() << endl;
        }
}
```

```
Circle [0,0]의 면적은 3.14
Circle [0,1]의 면적은 12.56
Circle [0,2]의 면적은 28.26
Circle [1,0]의 면적은 50.24
Circle [1,1]의 면적은 78.5
Circle [1,2]의 면적은 113.04
```

# C++ 메모리 구조

- 프로그램 실행을 위한 메모리(RAM) 영역

CODE	프로그램 실행코드, 기계어
DATA	전역변수, 정적변수(static) : 프로그램 시작 시 할당, 종료 시 소멸
HEAP	메모리 동적할당
STACK	지역변수, 매개변수 : 함수 호출 시 할당, 함수 종료 시 소멸

# 객체의 동적 할당

## ■ 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 운영체제로부터 할당 받음
  - 힙(heap)으로부터 할당
    - 힙은 운영체제가 소유하고 관리하는 메모리. 모든 프로세스가 공유할 수 있는 메모리

## ■ 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
  - `new`로 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

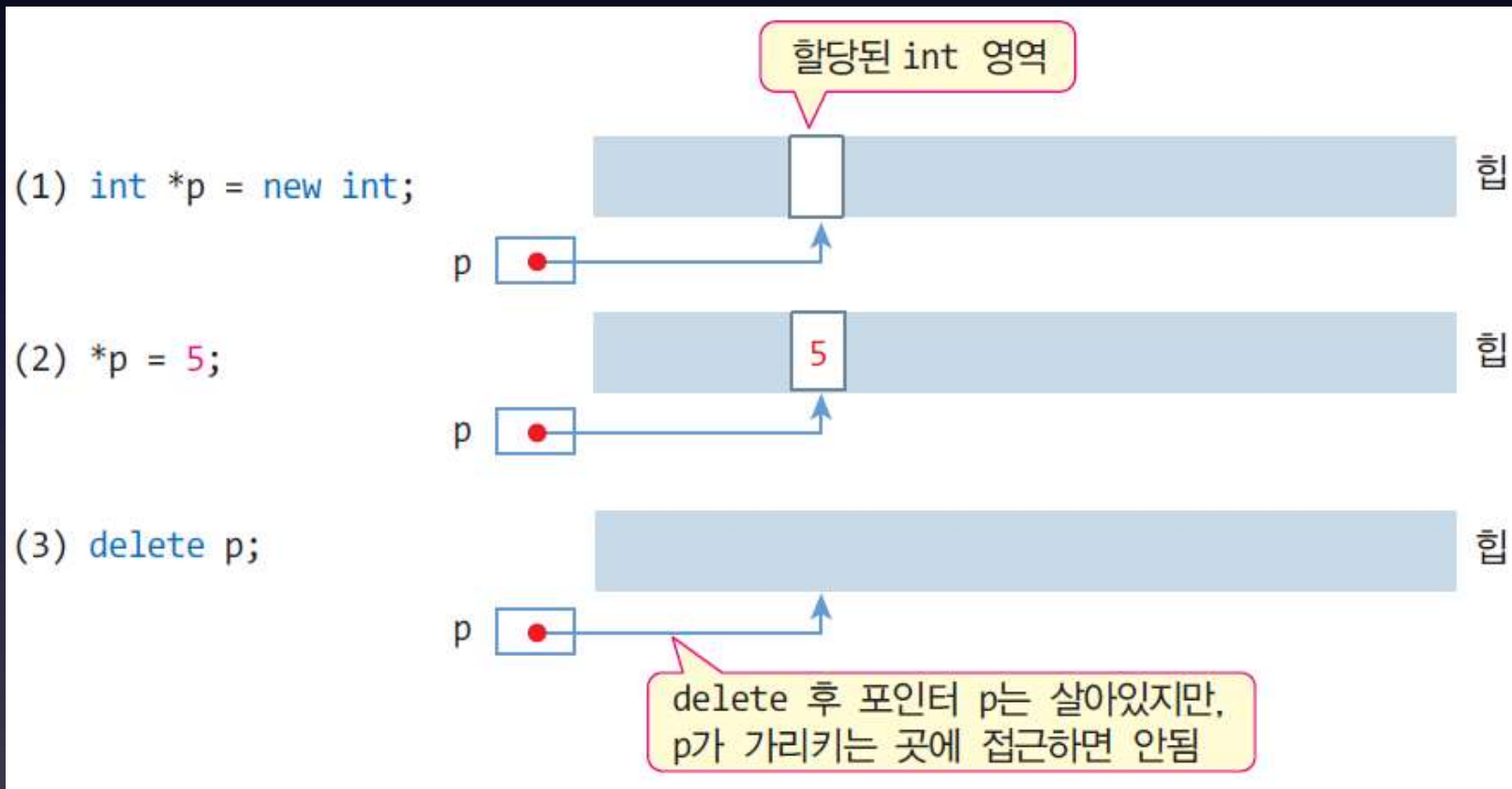
```
데이터타입 *포인터변수 = new 데이터타입 ;  
delete 포인터변수;
```

- 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
```



# 메모리 동적 할당 및 반환



# 예제 - 정수형 공간의 동적 할당

```
#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
```

# 예제 - 정수형 배열의 동적 할당

- 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 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

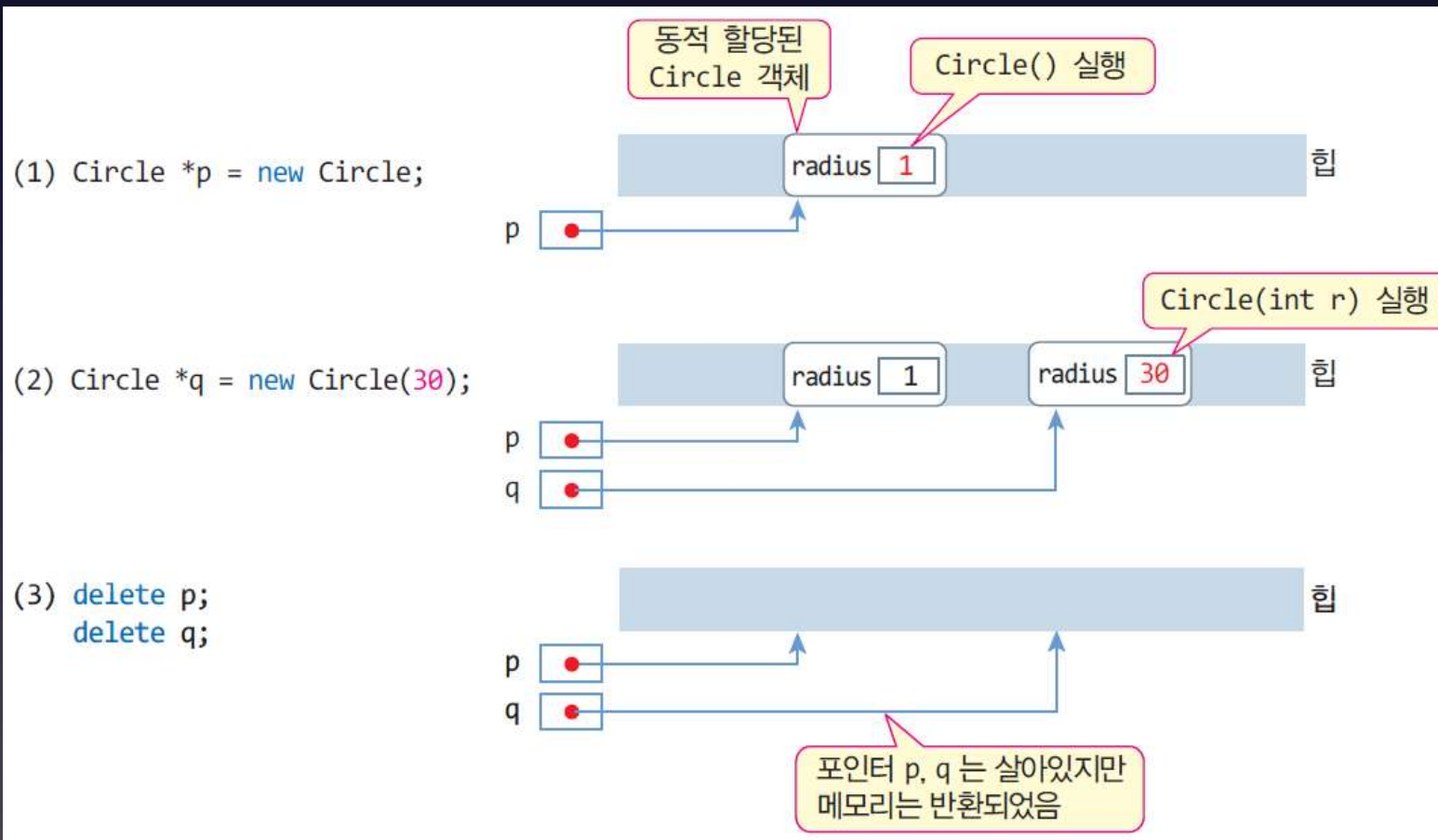
- 컴파일 오류는 아니지만 비정상적인 반환

```
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 - 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;
}
```

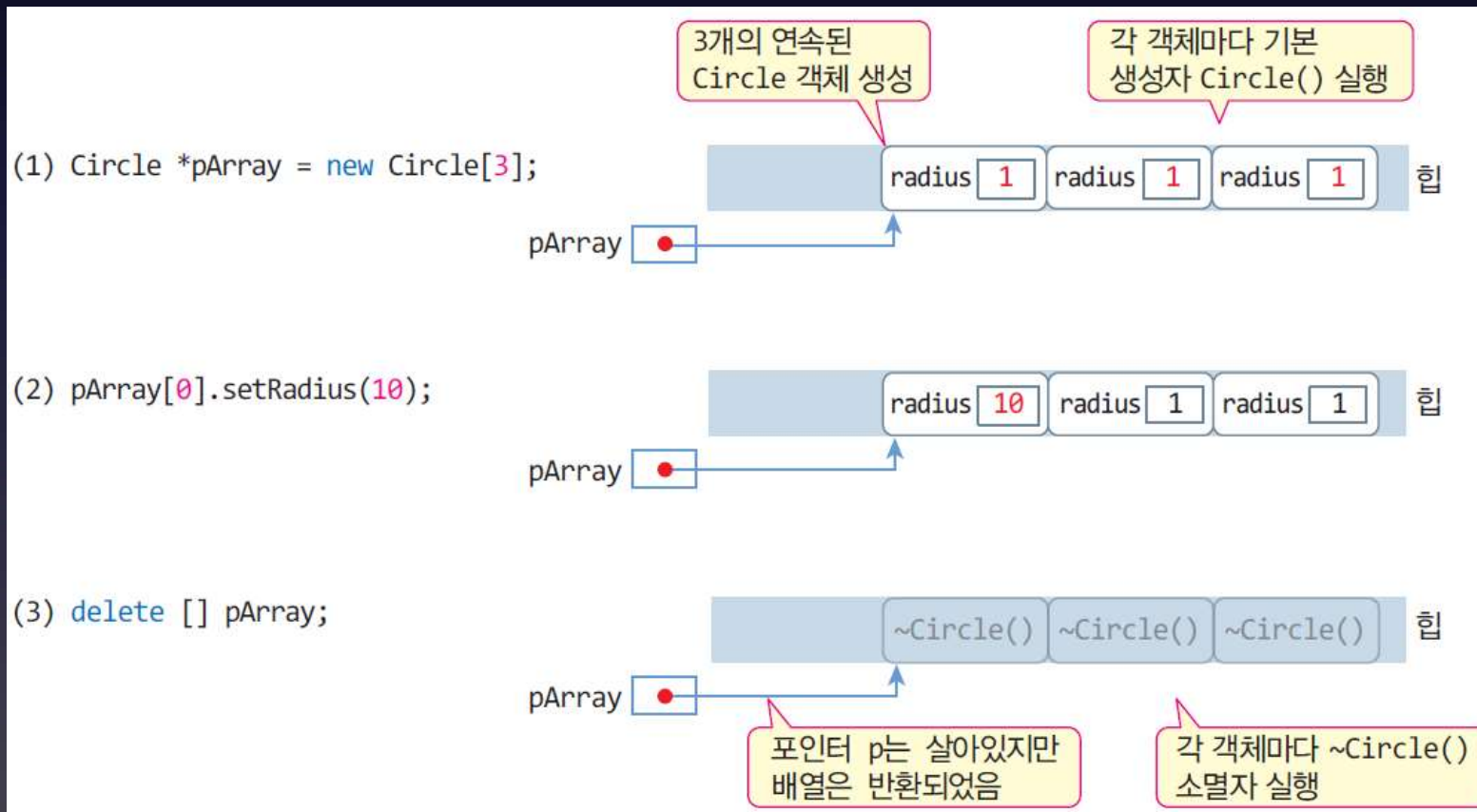
```
int main() {
    Circle *p, *q;
    p = new Circle;
    q = new Circle(30);
    cout << p->getArea() << endl << q->getArea() << endl;
    delete p;
    delete q;
}
```

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
```

# 객체 배열의 동적 생성 및 반환

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





# 객체 배열의 동적 사용

- 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();
}
```

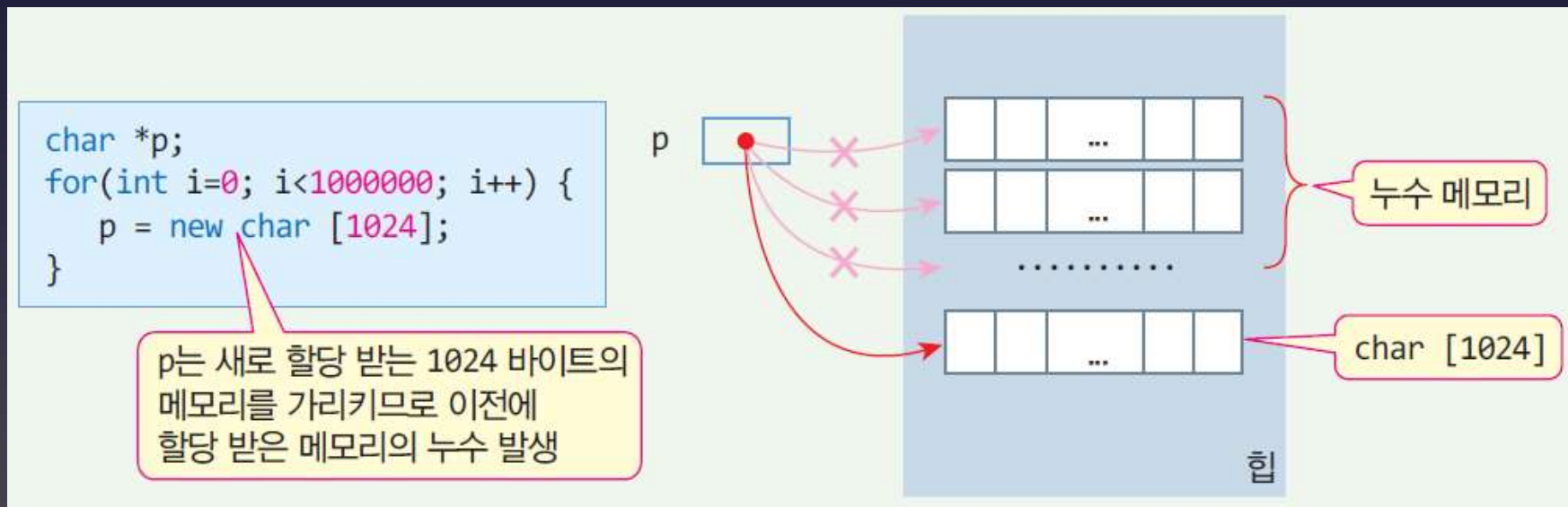
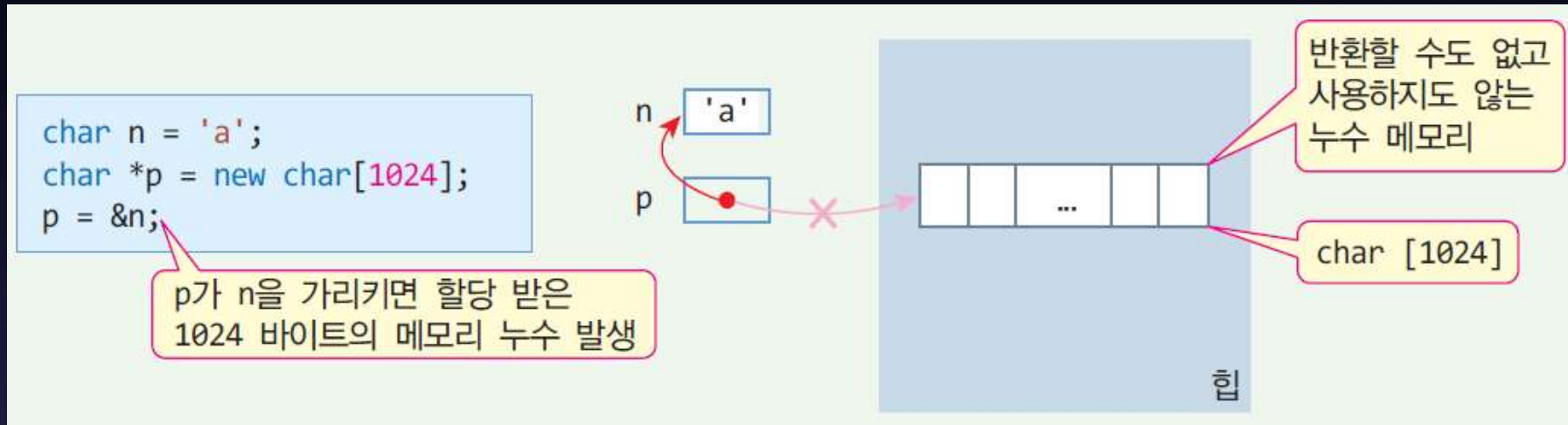
pArray[2] 객체의 소멸자 실행(1)  
pArray[1] 객체의 소멸자 실행(2)  
pArray[0] 객체의 소멸자 실행(3)

- 배열 소멸

```
delete [] pArray;
```

\* 각 원소 객체의 소멸자 별도 실행. 생성의 반대순

# 동적 메모리 할당과 메모리 누수



\* 프로그램이 종료되면, 운영체제는 누수 메모리를 모두 힙에 반환

# 실습 1

- 다음 main( ) 함수가 실행되도록 class Sample을 완성하시오.

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

class Sample
{
    int* _npData;
    int _nSize;

public:
    Sample() { _npData = 0; }
    void Read(int nSize);
    void Write( );
    int Big( );
    ~Sample( );
};
```

```
int main()
{
    Sample s;
    s.Read(5); //키보드에서 5개의 정수 읽어들이기
    s.Write();// 정수 배열 출력
    cout << "The big No. is " << s.Big( ) << endl;
}
```

# 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

c1

```
radius
...
void setRadius(int radius) {
    this->radius = radius;
}
...
```

c2

```
radius
...
void setRadius(int radius) {
    this->radius = radius;
}
...
```

c3

```
radius
...
void setRadius(int radius) {
    this->radius = radius;
}
...
```

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

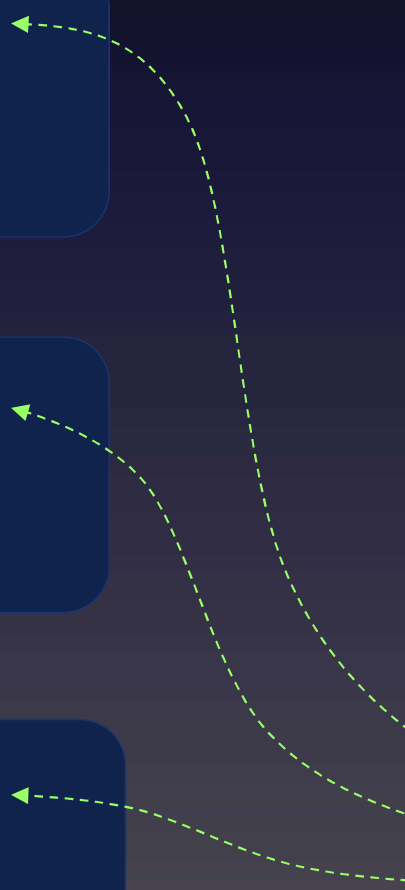
```
int main() {
    Circle c1;
    Circle c2(2);
    Circle c3(3);

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

~~4~~

~~5~~

~~6~~



# this가 필요한 경우

- 매개변수의 이름과 멤버 변수의 이름이 같은 경우

```
Circle(int radius)
{
    this->radius = radius;
}
```



```
Circle(int radius)
{
    radius = radius;
}
```

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

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

```
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

```
class Sample {  
    int a;  
public:  
    void setA(int x) {  
        this->a = x;  
    }  
};
```

개발자가 작성한 클래스

컴파일러에  
의해 변환

```
class Sample {  
    ...  
public:  
    void setA(Sample* this, int x) {  
        this->a = x;  
    }  
};
```

컴파일러에 의해 변환된 클래스

Sample ob;

ob.setA(5);

컴파일러에 의해 변환

ob.setA(&ob, 5);

객체의 멤버 함수를 호출하는 코드의 변환



# 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
  - C++11 부터

```
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/delete`를 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 – string 클래스 사용

```
#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;
}
```

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

# 예제 - string 배열

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

```
:  
  
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 – string 사용

- 빈칸을 포함하는 문자열을 입력 받고, 한 문자씩 왼쪽으로 회전하도록 문자열을 변경하고 출력하라.

```
:  
  
int main() {  
    string s;  
  
    cout << "문자열을 입력하세요 " << endl;  
    getline(cin, s, '\n'); // 문자열 입력  
    int len = s.length(); // 문자열의 길이  
  
    for(int i=0; i<len; i++)  
    {  
        string first = s.substr(0,1); // 맨 앞의 문자 1개를 문자열로 분리  
        string sub = s.substr(1, len-1); // 나머지 문자들을 문자열로 분리  
        s = sub + first; // 두 문자열을 연결하여 새로운 문자열로 만듦  
        cout << s << endl;  
    }  
}
```

```
문자열을 입력하세요  
I love you  
 love youI  
love youI  
ove youI I  
ve youI lo  
e youI lov  
 youI love  
youI love  
ouI love y  
uI love yo  
I love you
```

# 예제 – string 사용

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

```
#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();
```

& 뒤에 따라 오는  
<Enter> 키를 제거하기  
위한 코드!!!

```
    string fd;
    cout << endl << "find: ";
    getline(cin, fd, 'Wn'); // 검색할 문자열 입력
```

```
    string rp;
    cout << "replace: ";
    getline(cin, rp, 'Wn'); // 대치할 문자열 입력
```

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

# 예제 – string 사용

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&

find: **now**

replace: **Right Now**

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

& 뒤에 <Enter>키  
입력

## 실습 2

- Family 클래스를 완성하시오( family.h, family.cpp )

```
class Person {
    string name;
public:
    Person() { name=""; }
    Person(string name) { this->name = name; }
    string getName() { return name; }
    void setName(string name) { this->name = name; }
};
```

```
class Family {
    string name;
    Person* p; // Person 배열 포인터
    int size; // Person 배열의 크기. 가족 구성원 수
public:
    Family(string name, int size); // size 개수만큼 Person 배열 동적 생성
    void setName(int index, string name);
    void show(); // 모든 가족 구성원 출력
    ~Family();
};
```

```
int main() {
    Family *simpson = new Family("Simpson", 3);
    simpson->setName(0, "Mr. Simpson");
    simpson->setName(1, "Mrs. Simpson");
    simpson->setName(2, "Bart Simpson");
    simpson->show();
    delete simpson;
}
```

C:\Qt\Tools\QtCreator\bin\qtcreator\_process\_stub.exe

```
Simpson 가족은 다음과 같이 3명 입니다.
Mr. Simpson    Mrs. Simpson    Bart Simpson
Press <RETURN> to close this window...
```



