

객체 포인터
객체 배열
객체 동적 생성
문자열

객체 포인터 & 배열 & 동적 생성

- 한 주간 잘 지내셨나요?
- 오늘은
 - 객체 포인터
 - 객체 배열
 - 동적 메모리 할당
 - 문자열에 대하여 알아보려고 합니다
- 시작하기 전에 지난 시간에 대하여 간단하게 복습한 후 시작하도록 하겠습니다.
- 준비 되셨나요?

학습 목차

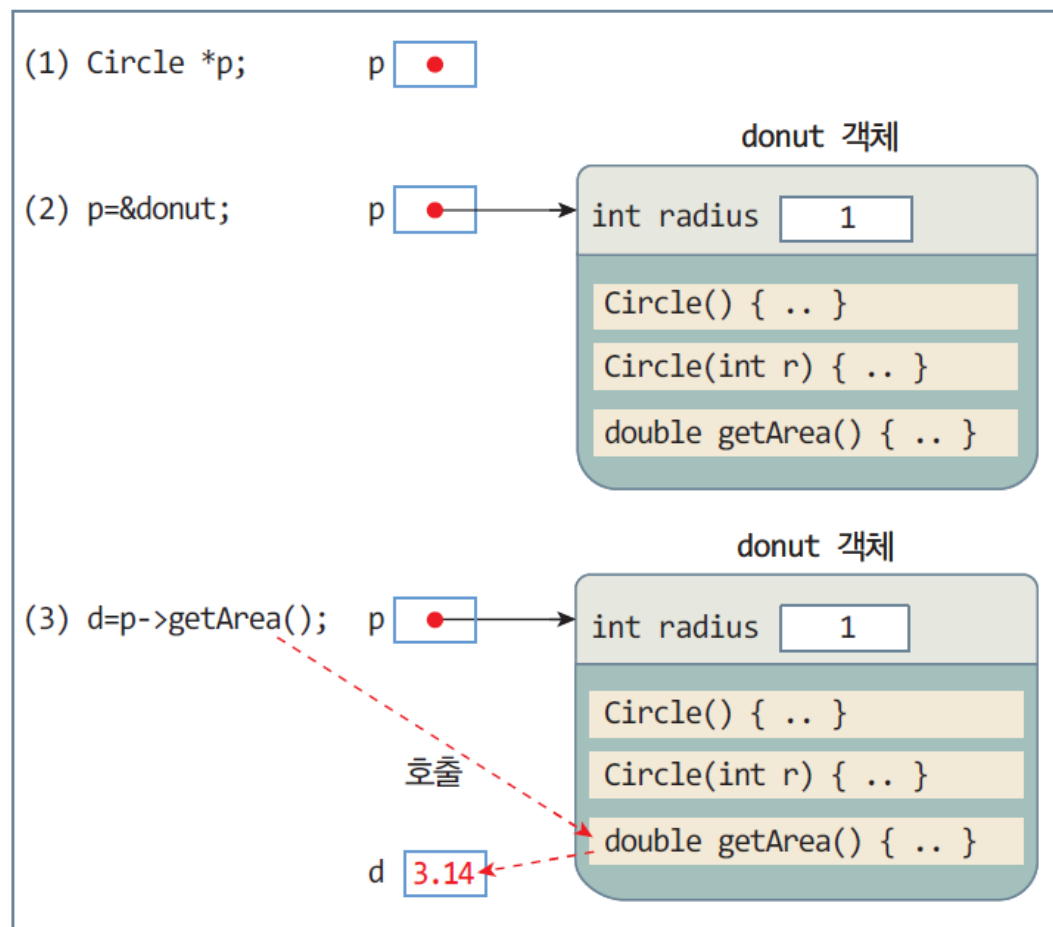
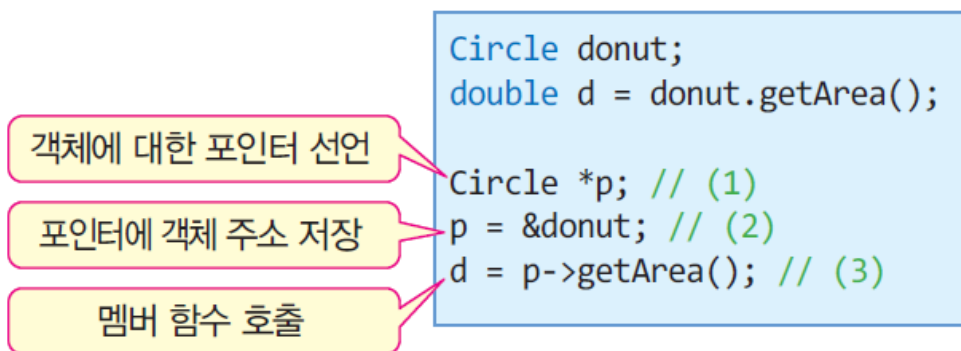
- 객체 포인터
- 객체 배열
- 객체 배열 초기화
- 동적 메모리 할당 및 반환
- this 포인터
- 스마트 포인터
- 객체 포인터 배열
- 문자열

학습 목표

- 객체에 대하여 포인터를 사용할 수 있다
- 객체 배열에 대하여 이해하고 활용할 수 있다
- new, delete 연산자를 사용하여 동적으로 메모리를 할당하고 반환하는 방법을 알 수 있다
- 객체 멤버에 대한 메모리 할당을 활용할 수 있다
- 스마트 포인터 개념을 이해하고 활용할 수 있다
- C언어와 C++ 언어의 동적 메모리 할당 차이점을 알 수 있다.
- C++ 문자열을 활용할 수 있다

객체 포인터

- 객체에 대한 포인터
 - C 언어의 포인터와 동일
 - 객체의 주소 값을 가지는 변수
- 포인터로 멤버를 접근할 때
 - 객체 포인터 → 멤버
 - (*객체포인터).멤버



nullptr

- Null Pointer를 의미, 즉 포인터가 아무것도 가리키고 있지 않음
- 이전에는 널 포인터를 나타내기 위해서는 NULL 매크로나 상수 0을 사용.
- 문제점
 - NULL 매크로나 상수 0을 사용하여 함수에 인자로 넘기는 경우 int 타입으로 추론

```
void func(int a) {  
    cout << "func-int" << endl << "a = " << a << endl;  
}
```

```
void func(double *p) {  
    cout << "func-double * Wn";  
}
```

```
int main() {  
    double *p = nullptr;  
    func(NULL);  
    func(p);  
}
```

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 **디폴트 생성자 실행**
 - **매개변수 있는 생성자를 호출할 수 없음**

```
Circle circleArray[3](5); // 오류
```
- 배열 소멸
 - 배열의 각 객체마다 소멸자 호출. 생성의 반대순으로 소멸

Circle 클래스의 배열 선언 및 활용

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

class Circle {
    int radius;
public:
    Circle() = default;
    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 생성, 디폴트 생성자가 없으면 오류 발생

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

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

    Circle *p;
    p = circleArray;
    for(int i=0; i<3; i++) {
        cout << "Circle " << i << "의 면적은 " << p->getArea() << endl;
        p++;
    }
}
```

```
// auto & 범위 기반 for : 배열의 각 원소 객체의 멤버 접근
for (auto obj : circleArray)
    cout << "Circle 면적 >> " << obj.getArea() << endl;
```


객체 배열 초기화

객체 배열 초기화 방법

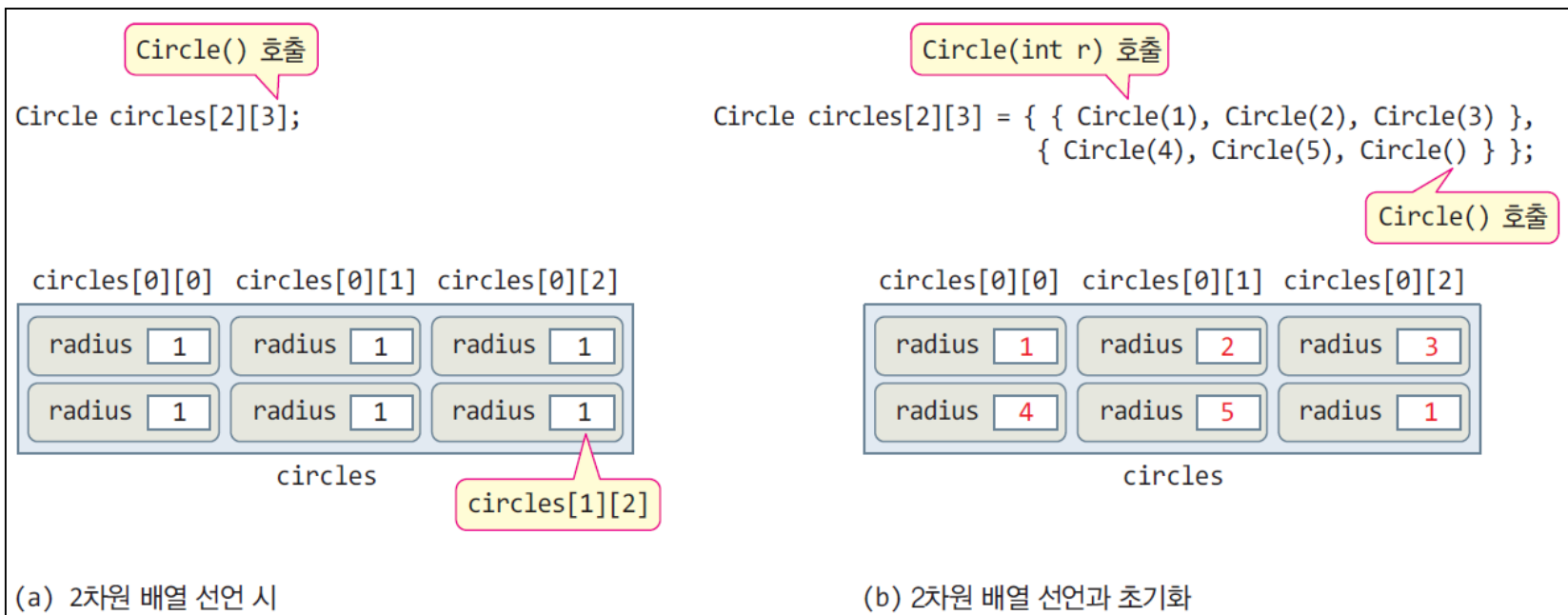
- 배열의 각 원소 객체 마다 생성자 지정

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

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

```
//array 클래스 사용  
array<Circle, 3> pArray = {Circle(40), Circle(50), Circle(20)};  
  
for(auto i=0; i<pArray.size() ;i++){ //size()함수->배열 크기 반환  
    cout<<pArray[i].getArea();  
}
```

2차원 객체 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

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

2차원 객체 배열 멤버 접근

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 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, 객체, 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
- 크기와 형 변환 필요 없음
- new/delete 연산자의 사용 형식

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

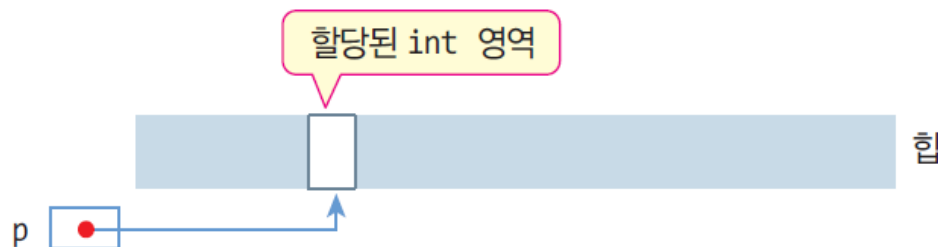
- new/delete의 사용

```
int *pInt = new int;
char *pChar = new char;
Circle *pCircle = new Circle();
```

```
delete pInt;    // 할당 받은 공간 반환
delete pChar;
delete pCircle;
```

```
if (!p) { //메모리 할당 여부 점검
    cout << "메모리를 할당할 수 없습니다.";
    return 0;
}
```

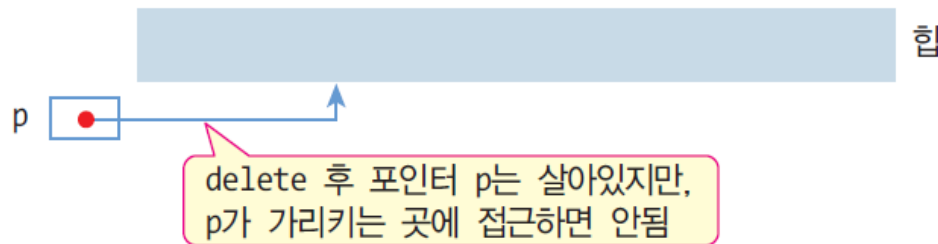
(1) `int *p = new int;`



(2) `*p = 5;`



(3) `delete p;`



(4) `p = nullptr;` //해제된 메모리를 다시 사용할 수 있는 실수 방지

delete 사용 시 주의 사항

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

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

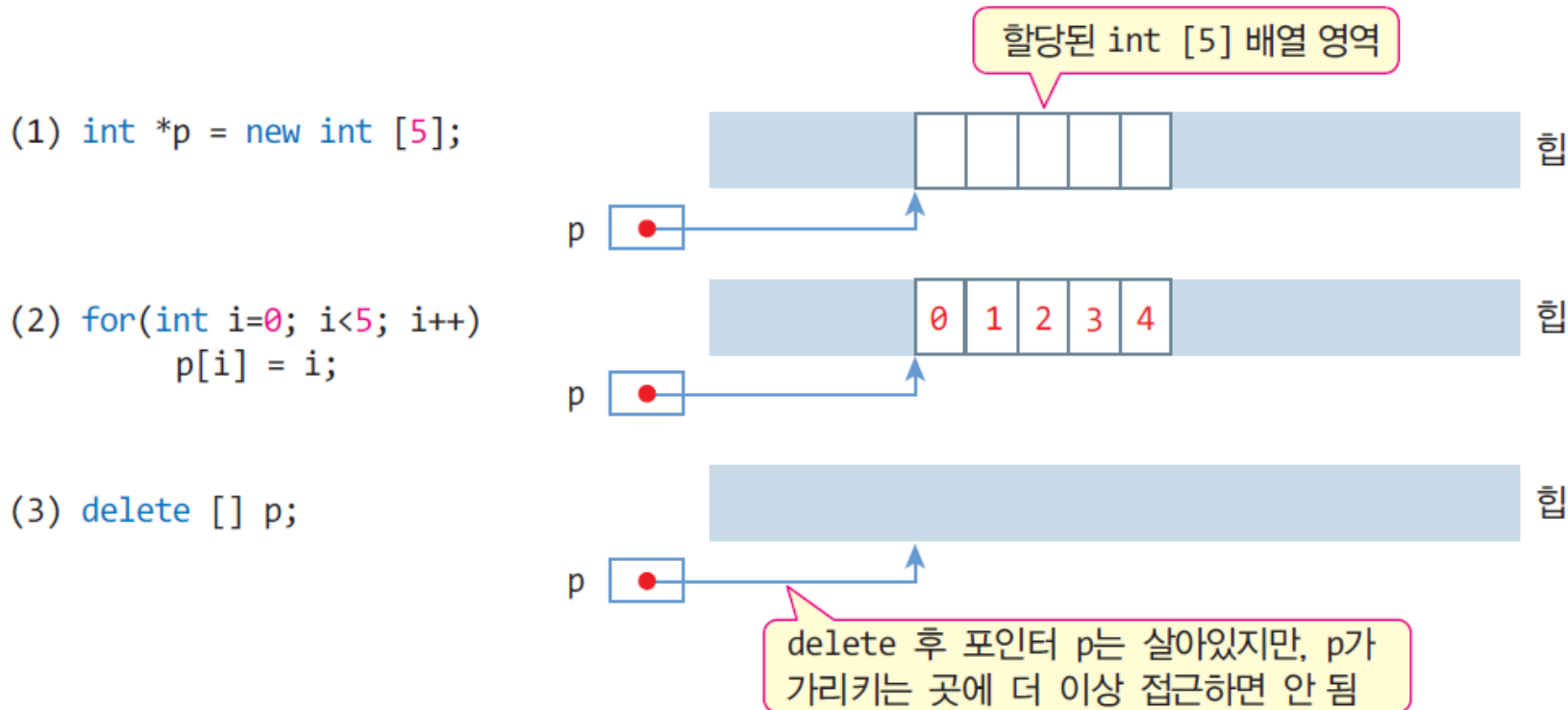
- 동일한 메모리 두 번 반환 - 오류

```
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
 - 배열 형태로 동적 생성한 것은 배열 형태로 삭제
 - 배열 크기는 생략 불가

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



동적 할당 메모리 초기화 및 delete 시 유의 사항

- 동적 할당 시 메모리 초기화

```
int *pt = new int{ 20 }; // 또는 int *pt=new int(20);
cout << "*pt = " << *pt << endl;
char *ch = new char{ 'c' }; //또는 char *ch = new char('c');
cout << "*ch = " << *ch << endl;
```

- 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
```

//유니폼 초기화를 사용하여 동적으로 할당되는 배열 초기화 가능

```
int *dm = new int[4] {3, 4, 5, 6};
```

- 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.
```

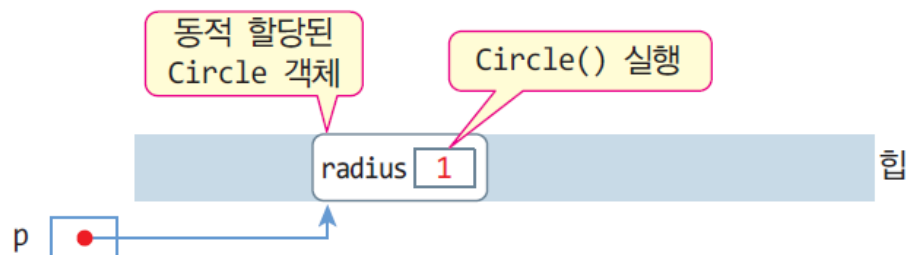
객체의 동적 생성 및 반환

- new 연산자는 객체 생성자를 호출하고 delete 연산자는 객체 소멸자 호출

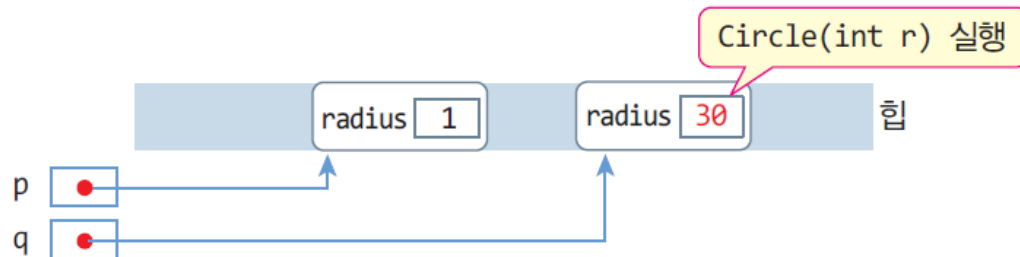
```
클래스이름 *포인터변수 = new 클래스이름;
클래스이름 *포인터변수 = new 클래스이름(생성자 매개변수리스트);
delete 포인터변수;
```

```
class Circle {
    int radius;
public:
    Circle():Circle(1){ }
    Circle(int r) : radius(r){ }
};
```

(1) Circle *p = new Circle;

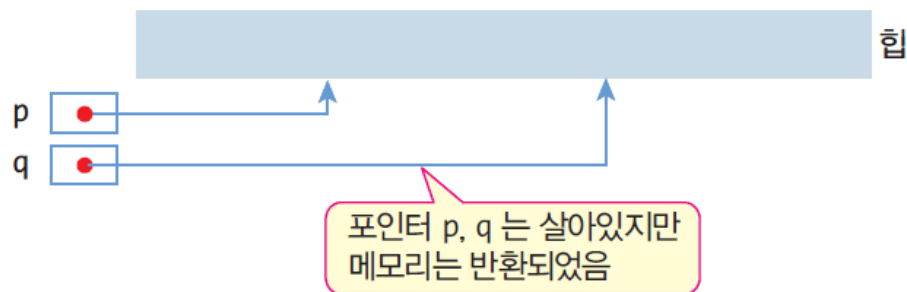


(2) Circle *q = new Circle(30);



(3) delete p;
delete q;

생성한 순서에 관계 없이 원하는
순서대로 delete 할 수 있음



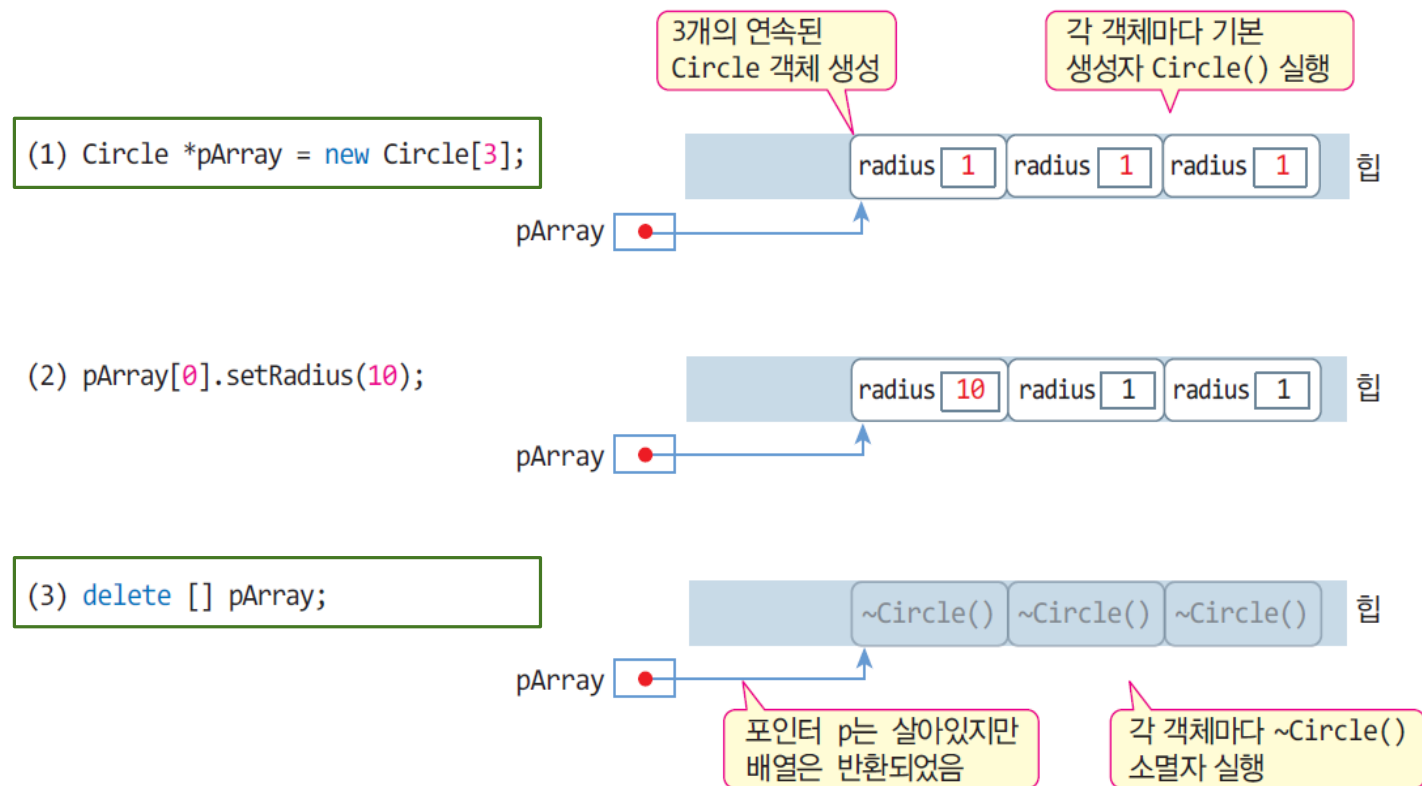
객체 배열의 동적 생성 및 반환

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

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

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

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



객체 배열의 사용, 배열의 반환과 소멸자

- 동적으로 생성된 배열도 보통 배열처럼 사용

```
Circle *pArray = new Circle[3];
```

```
pArray[0].setRadius(10);
```

```
pArray[1].setRadius(20);
```

```
pArray[2].setRadius(30);
```

```
for(int i=0; i<3; i++) {  
    cout << pArray[i].getArea();  
}
```

//유니폼 초기화를 사용하여 동적 할당 된 객체 배열 초기화

```
Circle *pArray = new Circle[3] {Circle(40), Circle(50), Circle(20)};
```

- 포인터로 배열 접근

```
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

멤버 동적 생성하기

- 클래스의 멤버도 동적 생성 가능
- 단, **생성자에서 동적 할당되어야** 하고 **소멸자에서 동적 메모리를 해제해야** 함

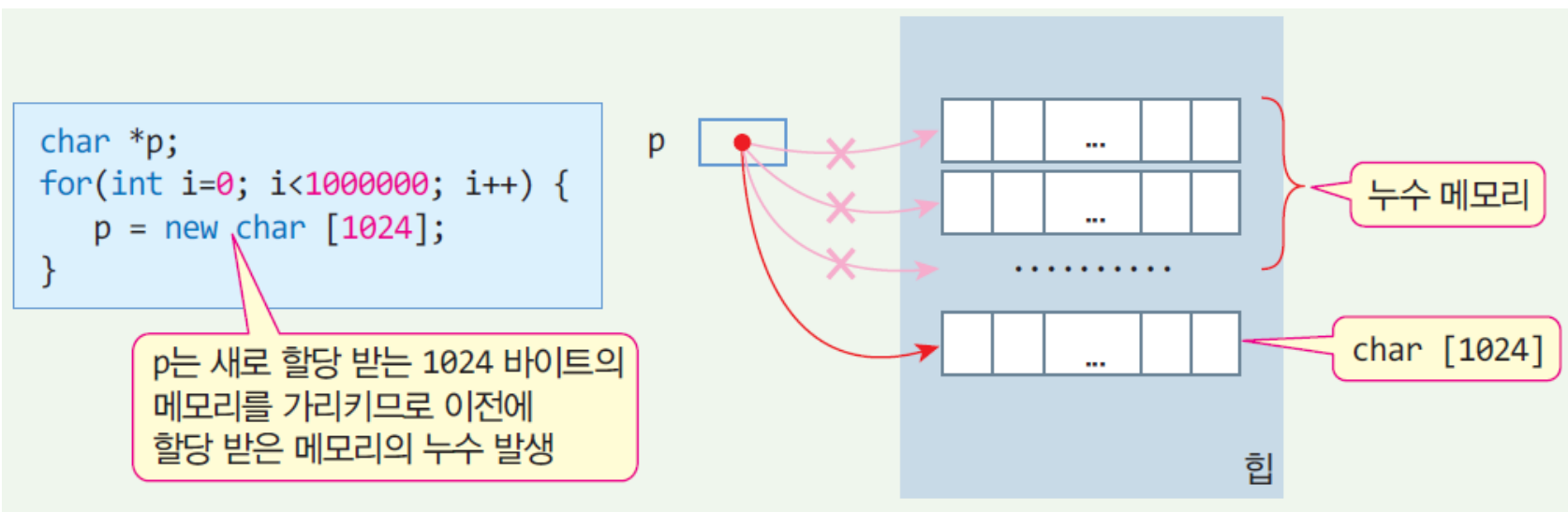
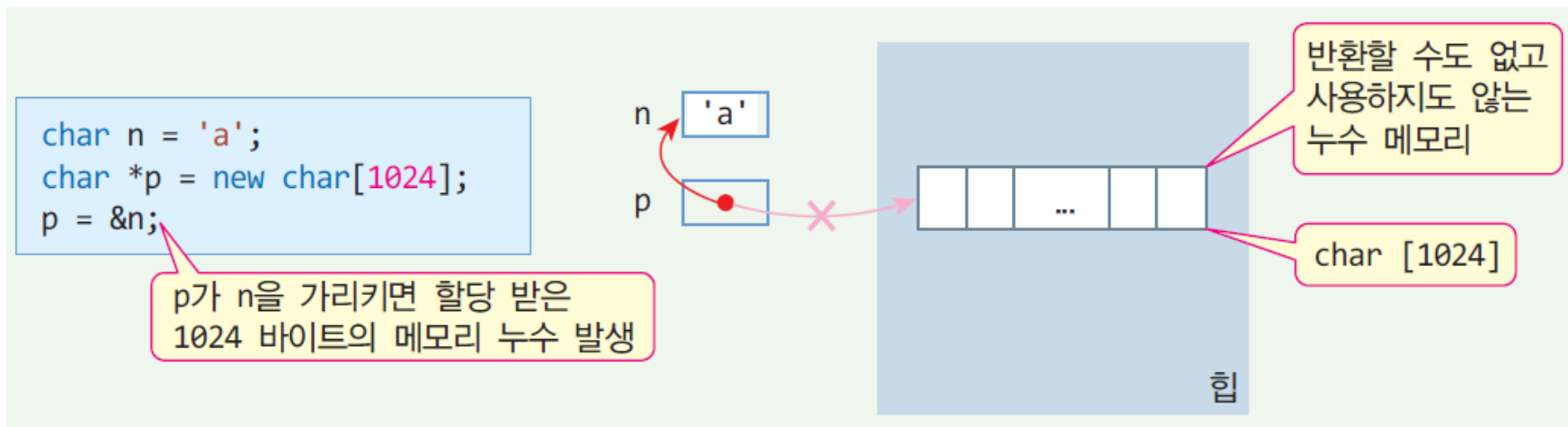
```
class Dog {
    string *name;
    int *age;
public:
    Dog(string n, int a){
        name = new string(n);
        age = new int(a);
    }
    ~Dog() {
        delete name;
        delete age;
    }
    int getAge() { return *age; }
    void setAge(int a) { *age = a; }
};
```

//생성자 초기화를 사용한 메모리 할당 & 초기화
 Dog(string n, int a) : name{ new string{n} }, age{ new int{a} } {}

//멤버 변수 동적 메모리 할당
 name = new string(n);
 age = new int(a);

```
int main() {
    Dog *p=new Dog("강아지", 2);
    cout << "강아지 나이 : " << p->getAge() << endl;
    p->setAge(5);
    cout << "강아지 나이 : " << p->getAge() << endl;
    delete p;
}
```

동적 메모리 할당과 메모리 누수



*** 프로그램이 종료되면, 운영체제는 누수 메모리를 모두 힙에 반환**

this 포인터

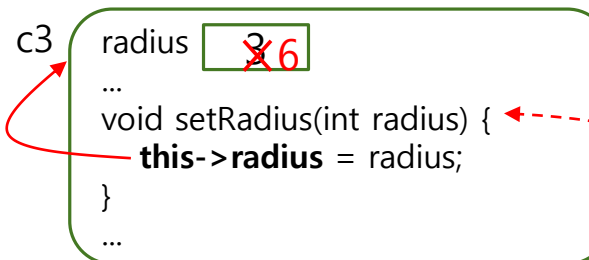
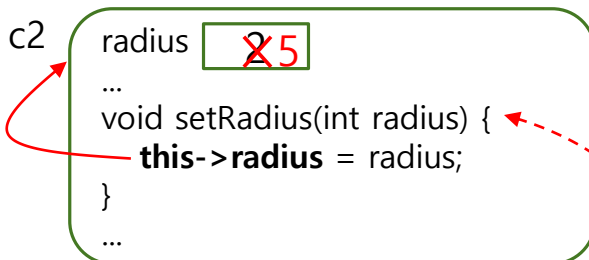
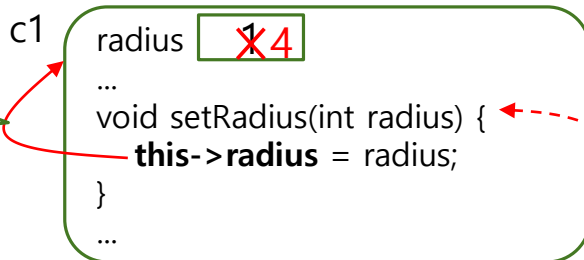
- this
 - 포인터, 객체 자신 포인터
 - 클래스의 멤버 함수 내에서만 사용
 - 개발자가 선언하는 변수가 아니고, 컴파일러가 선언한 변수
 - 컴파일러에 의해 묵시적으로 멤버 함수에 삽입 선언되는 매개 변수

```
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 **this**는 다른 객체의 **this**와 다름

this는 객체 자신에
대한 포인터



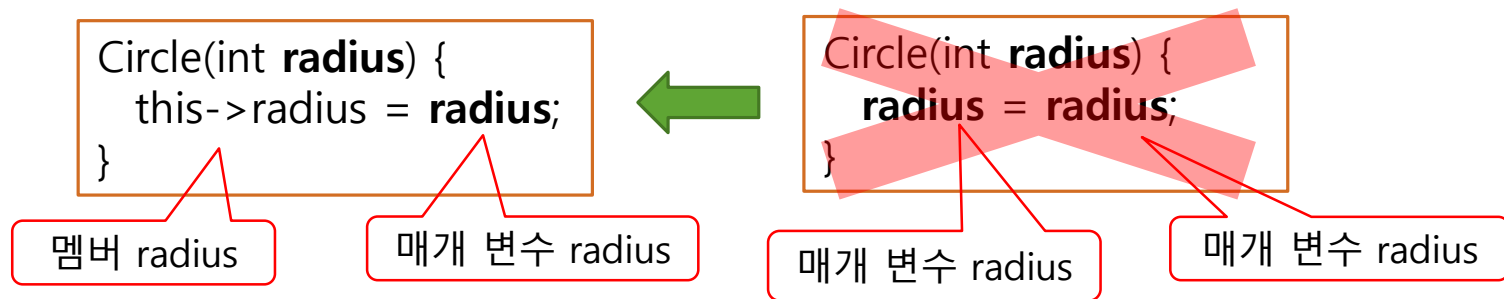
```
class Circle {
    int radius;
public:
    Circle() {
        this->radius=1;
    }
    Circle(int radius) {
        this->radius = radius;
    }
    void setRadius(int radius) {
        this->radius = radius;
    }
};
```

```
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 리턴 할 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 때문

this 포인터의 실체 – 컴파일러에서 처리

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

(a) 개발자가 작성한 클래스

컴파일러에 의해
변환

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

(b) 컴파일러에 의해 변환된 클래스

this는 컴파일러에 의해
묵시적으로 삽입된 매개 변수

Sample ob;

ob.setA(5);

컴파일러에 의해 변환

ob.setA(&ob, 5);

ob의 주소가 this
매개변수에 전달됨

(c) 객체의 멤버 함수를 호출하는 코드의 변환

스마트 포인터

- 스마트 포인터
 - 포인터처럼 동작하는 클래스 템플릿으로, 사용이 끝난 메모리를 자동으로 해제
 - 메모리 누수로부터 프로그램의 안전성 보장을 위해 제공
 - <memory>헤더 파일 필요
 - 일반 포인터와 동일하게 *, ->로 역 참조
- 종류
 - unique_ptr
 - 하나의 스마트 포인터만 객체를 소유
 - 스마트 포인터가 영역을 벗어나거나 리셋 되면 참조하던 resource 해제
 - 포인터에 대한 소유권을 이전(move)할 순 있지만 복사(copy)나 대입(assign)과 같은 공유(share)를 불허
 - shared_ptr
 - 하나의 특정 객체를 참조하는 스마트 포인터의 개수를 참조하는 스마트 포인터 – 참조 카운트
 - 참조 카운트란 해당 메모리를 참조하는 포인터가 몇개인지 나타내는 값, shared_ptr가 추가될 때 1씩 증가하고 수명이 다하면 1씩 감소, 참조 카운트가 0이 되면 메모리 자동 해제
 - weak_ptr
 - 하나 이상의 shared_ptr가 가리키는 객체를 참조할 수 있지만 reference count를 늘리지않는 스마트 포인터
 - 순환 참조를 제거하기 위해 사용
 - 순환 참조란 shared_ptr가 서로 상대방을 가리키는 것으로 reference count가 0이 되지 않아 메모리가 해제되지 않음

스마트 포인터 – unique_ptr

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

```
int main() {
    unique_ptr<int> p(new int); //unique_ptr 사용법1
    *p = 99; //포인터의 초기화
    cout << "*p = " << *p << endl;
    // unique_ptr<int> ip = p; //error
    unique_ptr<int> ap = move(p); //move()를 사용하여 포인터 이동
    // cout << "*p = " << *p << endl; //실행시간 오류
    cout << "*ap = " << *ap << endl;
```

```
    unique_ptr<double> sp = make_unique<double>(23.5); //unique_ptr 사용법2
    cout << "*sp = " << *sp << endl;
```

```
    auto asp = make_unique<int[]>(5); //unique_ptr 사용법3, make_unique<T>() 사용을 권고
    for (int i = 0; i < 5; i++) {
        asp[i] = 10 + i;
        cout << asp[i] << " ";
    }
    cout << endl;
}
```

스마트 포인터 - shared_ptr

```
#include <memory>
class Person{
    string name, tel;
public:
    Person() = default;
    Person(string n, string t) :name(n), tel(t) { };
    void display() { cout << "name = " << name << "tel = " << tel << endl; }
};

void show(shared_ptr<Person> sp) {
    sp->display(); cout << "count(sp) : " << sp.use_count() << endl;
}

int main() {
    auto p = make_shared<Person>("unique", "C++_17");
    p->display(); cout << "count(p) : " << p.use_count() << endl;

    show(p); cout << "count(p) : " << p.use_count() << endl;

    shared_ptr<Person> sp2=p; // 또는 auto sp2=p;
    sp2->display();
    cout << "count(p) : " << p.use_count() << endl;
    cout << "count(sp2) : " << sp2.use_count() << endl;
}
```

```
Microsoft Visual Studio 디버그 콘솔
name = unique tel = C++_17
count(p) : 1
name = unique tel = C++_17
count(sp) : 2
count(p) : 1
name = unique tel = C++_17
count(p) : 2
count(sp2) : 2
```

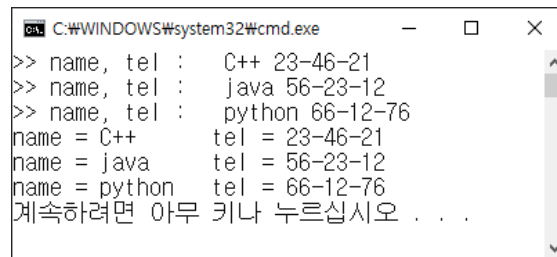
객체 포인터 배열

- 객체의 주소 값 저장이 가능한 포인터 변수로 이뤄진 배열

//3개의 객체 주소를 저장할 수 있는 객체 포인터 배열을 선언하여 Person 클래스 객체 처리

```
class Person {
    string name, tel;
public:
    Person(string n, string t) :name(n), tel(t) { };
    void display() const { cout << "name = " << name << " \t tel = " << tel << endl; }
};

int main() {
    Person *per[3]; //3개의 객체 주소를 저장할 수 있는 객체 포인터 배열 선언
    string name, tel;
    int size = sizeof(per) / sizeof(per[0]); //배열 크기 계산
    for (int i = 0; i < size; i++) {
        cout << ">> name, tel : "; cin >> name; cin >> tel;
        per[i] = new Person(name, tel); //생성된 객체의 주소를 저장
    }
    for (int i = 0; i < size; i++) {
        per[i]->display(); //(*(per+i))->display();
    }
    for (int i = 0; i < size; i++) {
        delete per[i]; //n번 객체 생성, n번 객체 소멸
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
>> name, tel : C++ 23-46-21
>> name, tel : java 56-23-12
>> name, tel : python 66-12-76
name = C++      tel = 23-46-21
name = java     tel = 56-23-12
name = python   tel = 66-12-76
계속하려면 아무 키나 누르십시오 . . .
```

c & c++ 동적 메모리 할당

```
#include <stdlib.h>
#include <iostream>
using namespace std;
```

```
class Apple {
public:
```

```
    Apple() { cout << "yummy apple" << endl; }
    ~Apple() { cout << "finished apple" << endl; }
```

```
};
```

```
int main() {
    //heap int : c style
    int *a = (int *)malloc(sizeof(int));
    *a = 100;
    free(a);
```

```
    //heap int array : c style
    int *b = (int *)malloc(sizeof(int)*3);
    b[0] = 100;
    free(b);
```

```
    //heap Apple : c style
    Apple *c = (Apple *)malloc(sizeof(Apple));
    free(c);
```

```
    //heap Apple array : c style
    Apple *d = (Apple *)malloc(sizeof(Apple)*3);
    free(d);
```

```
}
```

```
PS C:\yanges\lecture\lecture_src\cpp> g++ cpptest.cpp
PS C:\yanges\lecture\lecture_src\cpp> ./a
```

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

```
class Apple {
public:
```

```
    Apple() { cout << "yummy apple" << endl; }
    ~Apple() { cout << "finished apple" << endl; }
```

```
};
```

```
int main() {
    //heap int : cpp style
    int *a = new int;
    *a = 100;
    delete a;
```

```
    //heap int array : cpp style
    int *b = new int[3];
    b[0] = 100;
    delete []b;
```

```
    //heap Apple : c++ style
    Apple *c = new Apple;
    delete c;
```

```
    //heap Apple array : c++ style
    Apple *d = new Apple[3];
    delete []d;
```

```
}
```

```
yummy apple
finished apple
yummy apple
yummy apple
finished apple
finished apple
finished apple
```

c++ & safer c++ 동적 메모리 할당

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

```
class Apple {
public:
    Apple() { cout << "yummy apple" << endl; }
    ~Apple() { cout << "finished apple" << endl; }
};

int main() {
    //heap int : c++ style
    cout << "== unique_ptr<int> ==" << endl;
    unique_ptr<int> a = make_unique<int>();

    //heap int array : c++ style
    cout << "== vector<int> ==" << endl;
    vector<int> b(3);
    b[0]=55;
    cout << " b.at(0) or b[0]=" << b[0] << endl;

    //heap Apple : c++ style : memory leak 해결
    cout << "== unique_ptr<Apple> ==" << endl;
    unique_ptr<Apple> c = make_unique<Apple>();

    //heap Apple array : c++ style
    cout << "== vector<Apple> ==" << endl;
    vector<Apple> c(3);
}
```

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

```
class Apple {
public:
    Apple() { cout << "yummy apple" << endl; }
    ~Apple() { cout << "finished apple" << endl; }
};

int main() {
    //heap int : cpp style
    int *a = new int;
    *a = 100;
    delete a;

    //heap int array : cpp style
    int *b = new int[3];
    b[0] = 100;
    delete []b;

    //heap Apple : c++ style
    Apple *c = new Apple;
    delete(c);

    //heap Apple array : c++ style
    Apple *d = new Apple[3];
    delete [] d;
}
```

```
yummy apple
finished apple
yummy apple
yummy apple
yummy apple
finished apple
finished apple
finished apple
```

c++ 문자열 – string 클래스

- string 클래스 -> c++ 문자열처리는 string 클래스 사용을 권고
 - C++ 표준 라이브러리, <string> 헤더 파일에 선언
 - 가변 크기의 문자열
 - 다양한 문자열 연산을 실행하는 연산자와 멤버 함수 포함
 - 문자열, 스트링, 문자열 객체, string 객체 등으로 혼용
 - null 문자로 끝나지 않음

- 문자열 생성

```
string str; // 빈 문자열을 가진 스트링 객체
string address1("강원도 춘천시 한림대학길");
string address2("강원도 춘천시 한림대학길", 6); //address2="강원도";
string copyAddress(address1);
```

```
getline(cin, str, 'c'); //getline() 문자열 입력 함수, 세번째 인수는 구분 기호(생략하면 \n), 문자 'c'에서 입력 중지
```

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

```
auto string1="Hello cpp"; //const char*
auto string2="Hello cpp"s; //str::string
```


c++ 문자열 - string 클래스

- string 객체의 동적 생성과 소멸
 - 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
```

- 숫자 <-> 문자열 변환

//숫자-> 문자열로 변환

```
cout<<to_string(130)<<endl;
cout<<to_string(3.14)<<endl;
```

//문자열 -> 숫자로 변환

```
cout<<stoi("1234aa")<<endl; //int, 오류 아님, 1234로 변환
cout<<stol("3456")<<endl; //long, stol("a3456") -> error
cout<<stod("12.34")<<endl; //double
cout<<stof("34.45")<<endl; //float
```

c++ 문자열 – string 클래스

```
#include <string>
#include <locale> //문자를 다루는 함수, isdigit(), toupper(), isalpha(), islower()
using namespace std;
int main() {
    string s1 = "C++ programming", s2="language";
    cout << "문자열 비교 : " << s1.compare(s2) << endl;
    cout << "문자열 연결 : " << s1.append(s2) << endl;
    s1="C++ programming";
    cout << "문자열 삽입 : " << s1.insert(3, "really") << endl;
    cout << "문자열 대치 : " << s1.replace(3, 6, "study") << endl; //3부터 6개의 문자를 study로 대치
    cout << "문자열 길이 : " << s1.length() << endl; cout << "문자열 길이 : " << s1.size() << endl;
    cout << "문자열 삭제 : " << s1.erase(0, 4) << endl; //문자열 일부분 삭제, 0부터 4개 문자 삭제
    s1.clear(); //문자열 전체 삭제
    cout << "문자열 삭제 : " << s1 << endl;
    s1 = "C++ programming";
    cout << "문자열 검색 : " << s1.find("p") << endl; //문자나 문자열 검색, 없으면 -1반환
    cout << "문자열 추출 : " << s1.substr(2,4) << endl; //문자열 일부 추출, 인덱스 2에서 4개의 문자 반환
    cout << "문자열 추출 : " << s1.substr(2) << endl; //문자열 일부 추출, 인덱스 2에서 끝까지 반환
    cout << "문자열의 각 문자 다루기 : " << s1.at(5) << endl; //인덱스 5에 있는 문자 반환
    cout << "문자 다루기 : " << static_cast<char>(toupper('a')) << endl;
    if (isdigit(s1.at(6))) cout << "숫자" << endl;
    else if (isalpha(s1.at(6))) cout << "문자" << endl;
    else cout << "해당없음" << endl;
    return 0;
}
```

string_view 클래스 - c++ 17

- string_view 클래스
 - 다양한 타입의 문자열 처리
 - 임시 객체 생성없이 문자열 사용, 읽기전용
 - 단, 언어 표준이 C++17인 경우만 동작

```
#include <string_view> //string_view 클래스
```

```
//string_view 클래스 사용
string_view extractExt(string_view filename) {
    return filename.substr(filename.find('.')); //확장자만 반환
}
```

```
int main() {
    string filename = "C:\\temp\\file.string";
    //data() : string_view 타입 문자열 -> string 타입 문자열 생성
    //cout << extractExt(filename).data()<< endl;
    cout << extractExt(filename)<< endl;

    const char* cfilename = "C:\\temp\\file.cstring";
    cout << extractExt(cfilename)<< endl;

    cout << extractExt("C:\\temp\\file.literal")<< endl;
    return 0;
}
```

```
#pragma warning(disable : 4996)
#include <iostream>
#include <string>
#include <cstring>
using namespace std;
```

```
//다양한 타입의 문자열 처리를 위해 함수 오버로딩
string extractExt(const string& filename) {
    return filename.substr(filename.find('.'));
}
```

```
char* extractExt(char* filename) {
    char *p = nullptr;
    char ext[10];
    p = strchr(filename, '.');
    strcpy(ext, p+1);
    return ext; //확장자만 반환
}
```

```
int main() {
    string filename = "C:\\temp\\file.string";
    cout << extractExt(filename)<< endl;

    const char* cfilename = "C:\\temp\\file.cstring";
    cout << extractExt(cfilename)<< endl;

    cout << extractExt("C:\\temp\\file.literal")<< endl;
    return 0;
}
```

실습 1. 동적 객체 생성과 반환 – new, delete

```
#include <iostream>
using namespace std;
class Color {
    int red, green, blue;
    string color;

public:
    Color() : Color(0, 0, 0, "black" ){ }
    Color(int r, int g, int b, string c) : red(r), green(g), blue(b), color(c) { }
    ~Color() { cout << color << " 객체 소멸" << endl; }
    void setColor(int r, int g, int b, string c);
    void show() const;
};
```

```
int main() {
    Color *p=new Color;
    Color *q = new Color(0, 255, 0, "green");
    p->show();
    q->show();
    //생성한 순서에 관계 없이 원하는 순서대로 delete 할 수 있음
    delete q;
    delete p;
    return 0;
}
```

실습 2. 동적 객체 생성과 반환 - unique_ptr

```
#include <iostream>
#include <memory>
#include <string>
using namespace std;
class Person{
    string name;
    string tel;
public:
    Person() = default;
    Person(string n, string t) :name(n), tel(t) { };
    void display() const{ cout << "name = " << name << "tel = " << tel << endl; }
};

int main() {
    auto p = make_unique<Person>("unique", "C++_17");
    p->display();
}
```

실습 3. 객체 멤버 동적 생성 – new, delete

```
class Matrix{
private:
    int rowSize, colSize;
    int **ptr;
public:
    Matrix(int rowSize, int colSize);
    ~Matrix();
    void setup();
    void print();
};

Matrix ::Matrix(int r, int c) : rowSize(r), colSize(c){
    ptr = new int *[rowSize];
    for (int i = 0; i < rowSize; i++) {
        ptr[i] = new int[colSize];
    }
}

void Matrix ::setup(){
    for (int i = 0; i < rowSize; i++){
        for (int j = 0; j < colSize; j++){
            ptr[i][j] = rand() % 5 + 1;
        }
    }
}
```

```
Matrix::~Matrix(){
    for (int i = 0; i < rowSize; i++){
        delete[] ptr[i];
    }
    delete[] ptr;
}

void Matrix::print(){
    for (int i = 0; i < rowSize; i++){
        for (int j = 0; j < colSize; j++) {
            cout << setw(5) << ptr[i][j];
        }
        cout << endl;
    }
    cout << endl;
}

int main()
{
    cout << "matrix1" << endl;
    Matrix matrix1(3, 4);
    matrix1.setup();
    matrix1.print();
}
```

실습 4. 객체 멤버 동적 생성 – unique_ptr

```
include <iostream>
#include <memory>
#include <string>
using namespace std;
class Person{
    unique_ptr<string> name;
    unique_ptr<string> tel;
public:
    Person();
    Person(string n, string t);
    ~Person();
    void display() const;
};

Person::~Person() {
    cout << *name << " 객체 해제" << endl;
}
```

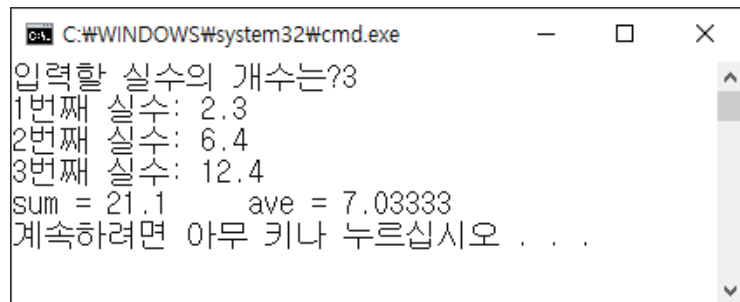
```
Person::Person() :Person("null", "null") {}
Person::Person(string n, string t) {
    name = make_unique<string>(n);
    tel = make_unique<string>(t);
}

void Person::display() const {
    cout << "name = " << *name << " \t tel = " << *tel << endl;
}

int main() {
    unique_ptr<Person> p=make_unique<Person>("java", "1116-1");
    p->display();
    return 0;
}
```

활용1. 실수 형 배열의 동적 할당

- 사용자로부터 배열 크기를 입력 받아 실수 배열을 동적 할당 받고 입력 받은 값으로 초기화한다.
- 배열 전체 합과 평균을 출력한다.



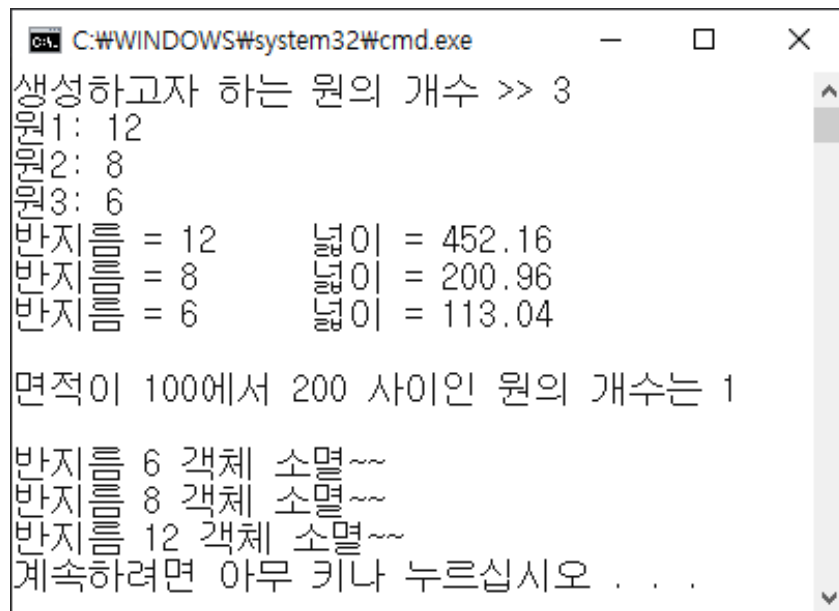
```
C:\WINDOWS\system32\cmd.exe
입력할 실수의 개수는?3
1번째 실수: 2.3
2번째 실수: 6.4
3번째 실수: 12.4
sum = 21.1      ave = 7.03333
계속하려면 아무 키나 누르십시오 . . .
```


활용 2. 객체 배열 동적 생성

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

class CircleManage {
    Circle *p; //동적으로 할당된 배열의 시작 주소 저장
    int size;
public:
    CircleManage();
    ~CircleManage();
    void CircleWrite();
};
```

- ❖ 입력 받은 원의 개수만큼 객체 배열을 생성하고
면적이 100~200사이의 원의 개수를 출력하시오 –
new, delete 사용



```
C:\WINDOWS\system32\cmd.exe
생성하고자 하는 원의 개수 >> 3
원1: 12
원2: 8
원3: 6
반지름 = 12      넓이 = 452.16
반지름 = 8       넓이 = 200.96
반지름 = 6       넓이 = 113.04

면적이 100에서 200 사이인 원의 개수는 1

반지름 6 객체 소멸~~
반지름 8 객체 소멸~~
반지름 12 객체 소멸~~
계속하려면 아무 키나 누르십시오 . . .
```

활용 3. 객체 배열 동적 할당

- Fraction 객체 배열을 동적 할당하여 분수를 처리하는 프로그램을 완성하시오 – 스마트 포인터 사용

```
#include <iostream>
using namespace std;
#ifdef FRACTION_H
#define FRACTION_H
class Fraction
{
    int numer; //분자
    int denom; //분모
public:
    Fraction(int num, int den); //매개변수가 있는 생성자
    Fraction(); //디폴트 생성자
    ~Fraction();

    int getNumer() const; //접근자
    int getDenom() const;
    void setNumer(int num); //설정자
    void setDenom(int den);
    void print() const;

private:
    void normalize(); //gcd() 함수를 사용하여 기약분수로 처리
    int gcd(int n, int m); //분모와 분자의 최대 공약수
};
#endif
```

```
int main()
{
    unique_ptr<Fraction[]> frac;
    int cnt;
    int nume, demon;

    cout << "생성하고자 하는 분수의 개수는 >> ";
    cin >> cnt;

    frac = make_unique<Fraction[]>(cnt);

    for (int i = 0; i < cnt; i++) {
        cout << "분자와 분모값을 입력하세요 >>";
        cin >> nume >> demon;
        frac[i].setNumer(nume);
        frac[i].setDenom(demon);
    }

    for (int i = 0; i < cnt; i++) {
        frac[i].print();
    }
```

활용 4. 문자열

- 다음과 같이 문자열을 처리하는 프로그램
 - 특정 문자 제거
 - 문자열 교체

```

C:\WINDOWS\system32\cmd.e...
문자열 입력: adsfwe etfga erekjdf
삭제하고자 하는 문자 입력 : e
삭제 후 문자열
adsfw tfga rkjdf

an old 문자열을 a new로 교체 후 문자열
This is a new string

계속하려면 아무 키나 누르십시오 . . .
  
```

```

string removeChar(string strg, char c); //문자 제거
string findAndReplace(string strg1, string strg2, string strg3); //문자열 교체
  
```

```

int main()
{
    string strg;
    char c;
    cout << "문자열 입력: "; getline(cin, strg);
    cout << "삭제하고자 하는 문자 입력 : "; cin >> c;
    cout << "삭제 후 문자열 " << endl;
    cout << removeChar(strg, c)<<endl<<endl;

    string strg1("This is an old string");
    string strg2("an old");
    string strg3("a new");
    strg1 = findAndReplace(strg1, strg2, strg3);
    cout << strg2<<" 문자열을 " <<strg3 << "로 교체 후 문자열 " << endl;
    cout << strg1<<endl<<endl;

    return 0;}
  
```

Q & A

- "C++ 객체포인터와 배열 동적 생성"에 대한 학습이 끝났습니다.
- 모든 내용을 이해 하셨나요?
- 아직 이해가 안되는 내용이 있다면 다시 한번 복습하시기 바랍니다.
- 질문은 한림 SmartLEAD 쪽지 또는 e-mail 또는 전화상담을 이용하시기 바랍니다.
- 5주 과제와 퀴즈(제한 기간내 2회 응시 가능)가 있습니다.
- 수고하셨습니다.^^