ROBIT
ROBOT SPORT GAME TEAM

C++ & Qt

18th 이선경

Basic-C Language

ROBIT ROBOT SPORT GAME TEAM

▶ main 함수

❖ main() 함수

- C++ 프로그램의 실행을 시작하는 함수
  - main() 함수가 종료하면 C++ 프로그램 종료
- main() 함수의 C++ 표준 모양

```
int main() { // main()의 리턴 타입 int
    .....
    return 0; // 0이 아닌 다른 값으로 리턴 가능
}
```

```
void main() { // 표준 아님
    .....
}
```

- main()에서 return문 생략 가능

```
int main() {
    .....
    // return 0; // 개발자의 편리를 위해 return 문 생략 가능
}
```

basic

namespace

- 이름(identifier) 충돌이 발생하는 경우
  - 여러 명이 서로 나누어 프로젝트를 개발하는 경우
  - 오픈 소스 혹은 다른 사람이 작성한 소스나 목적 파일을 가져와서 컴파일 하거나 링크하는 경우
  - 해결하는데 많은 시간과 노력이 필요
- namespace 키워드
  - 이름 충돌 해결
    - 2003년 새로운 C++ 표준에서 도입
  - 개발자가 자신만의 이름 공간을 생성할 수 있도록 함
    - 이름 공간 안에 선언된 이름은 다른 이름공간과 별도 구분
- 이름 공간 생성 및 사용

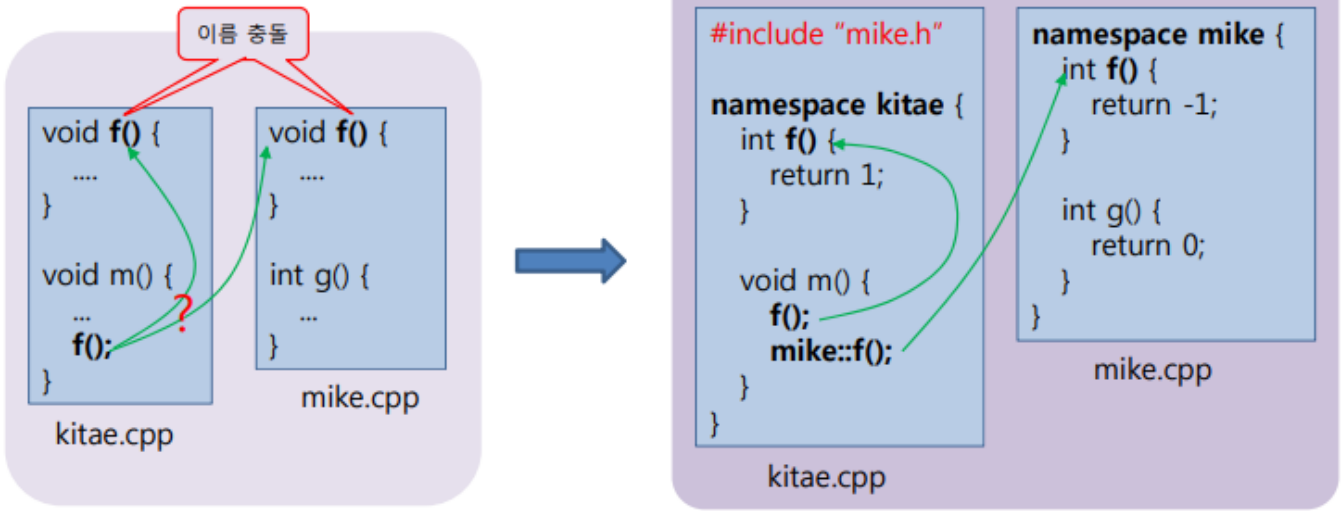
```
namespace kitae { // kitae 라는 이름 공간 생성
   ..... // 이 곳에 선언된 모든 이름은 kitae 이름 공간에 생성된 이름
}
```

  - 이름 공간 사용
    - 이름 공간 :: 이름

basic

ROBIT ROBOT SPORT GAME TEAM

- namespace



(a) kitae와 mike에 의해 작성된 소스를 합치면 f() 함수의 이름 충돌. 컴파일 오류 발생

(b) 이름 공간을 사용하여 f() 함수 이름의 충돌 문제 해결

basic

▶ cout (≒ printf)

❖ cout과 << 연산자 이용

```
std::cout << "Hello₩n"; // 화면에 Hello를 출력하고 다음 줄로 넘어감
std::cout << "첫 번째 맛보기입니다.";
```

❖ cout 객체

- **스크린** 출력 장치에 연결된 **표준** C++ 출력 스트림 객체
- <iostream> 헤더 파일에 선언
- std 이름 공간에 선언: **std::cout**으로 사용

❖ << 연산자

- 스트림 삽입 연산자(stream insertion operator)
  - C++ 기본 산술 시프트 연산자(<<)가 스트림 삽입 연산자로 재정의됨
  - ostream 클래스에 구현됨
  - 오른쪽 피연산자를 왼쪽 스트림 객체에 삽입
  - cout 객체에 연결된 화면에 출력
- 여러 개의 << 연산자로 여러 값 출력

```
std::cout << "Hello₩n" << "첫 번째 맛보기입니다.";
```

❖ 문자열 및 기본 타입의 데이타 출력

- bool, char, short, int, long, float, double 타입 값 출력

```
int n=3;
char c='#';
std::cout << c << 5.5 << '-' << n << "hello" << true;
```

#5.5-3hello1

- 연산식뿐 아니라 함수 호출도 가능

```
std::cout << "n + 5 =" << n + 5;
std::cout << f(); // 함수 f()의 리턴값을 출력한다.
```

❖ 다음 줄로 넘어가기

- '₩n'이나 endl 조작자 사용

```
std::cout << "Hello" << '₩n';
std::cout << "Hello" << std::endl;
```

basic

▶ cin (≒ scanf)

- cin
  - 표준 입력 장치인 키보드를 연결하는 C++ 입력 스트림 객체
- >> 연산자
  - 스트림 추출 연산자(stream extraction operator)
    - C++ 산술 시프트 연산자(>>)가 <iostream> 헤더 파일에 스트림 추출 연산자로 재정의됨
    - 입력 스트림에서 값을 읽어 변수에 저장
  - 연속된 >> 연산자를 사용하여 여러 값 입력 가능

```
cout << "너비와 높이를 입력하세요>>";
cin >> width >> height;
cout << width << '₩n' << height << '₩n';
```

```
너비와 높이를 입력하세요>>23 36
23
36
```

Basic-C Language

ROBIT ROBOT SPORT GAME TEAM

▶ 변수 정의

```
// 변수의 정의
#include <iostream>

int main() {
  int i;
  char c;
  double d;
  float f;

  return 0;
}
```

C에서 배운 기본 변수 자료형과 정의가 동일.

```
int arr[10];
int *parr = arr;

int i;
int *pi = &i;
```

포인터의 정의 역시 동일.

```
/* 변수는 변수 사용 직전에 선언해도 된다.*/
#include <iostream>

int main() {
  int sum = 0;

  for (int i = 1; i <= 10; i++) {
    sum += i;
  }

  std::cout << "합은 : " << sum << std::endl;
  return 0;
}
```

C와 다르게 for문, if문 등의
반복문, 비교문 안에서도 변수 선언이 가능.

Basic-C Language

ROBIT
ROBOT SPORT GAME TEAM

▶ 반복문,비교문

```cpp
// C++ 의 for 문
#include <iostream>

int main() {
  int i;

  for (i = 0; i < 10; i++) {
    std::cout << i << std::endl;
  }
  return 0;
}
```

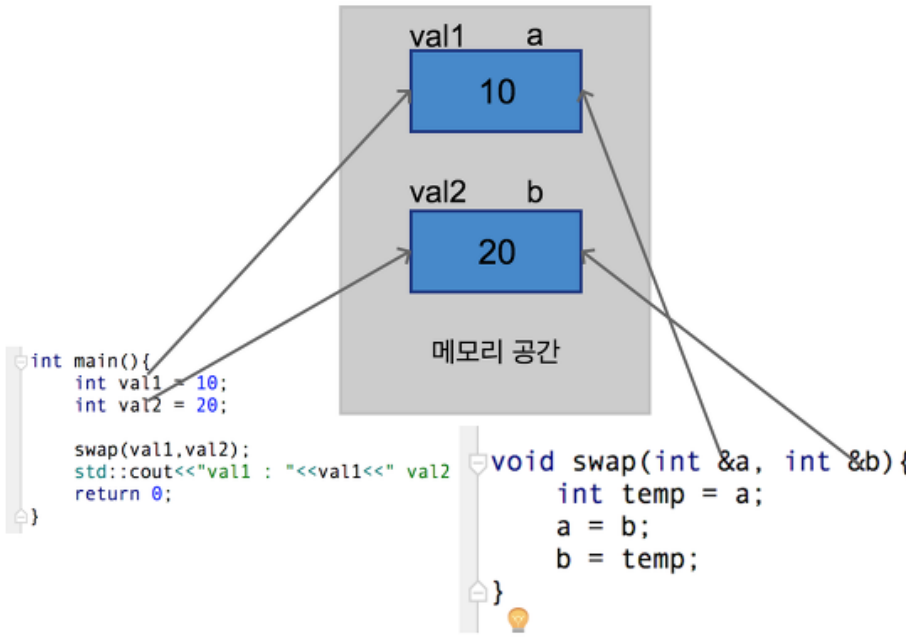
```cpp
/* 행운의 숫자 맞추기*/
#include <iostream>

int main() {
  int lucky_number = 3;
  std::cout << "내 비밀 수를 맞추어 보세요~" << std::endl;

  int user_input;  // 사용자 입력

  while (1) {
    std::cout << "입력 : ";
    std::cin >> user_input;
    if (lucky_number == user_input) {
      std::cout << "맞추셨습니다~~" << std::endl;
      break;
    } else {
      std::cout << "다시 생각해보세요~" << std::endl;
    }
  }
  return 0;
}
```

for, if, while, switch-case 등 대부분의 반복, 비교문이 C와 동일하게 동작.

Basic

ROBIT ROBOT SPORT GAME TEAM

▶ 참조자(reference)




```cpp
#include<iostream>

using namespace std;

int main()
{
    int num1 = 10;
    int &num2 = num1;    // num1을 참조하는 num2
                         // num1과 num2는 같은 메모리 공간 주소를 가짐

    cout << num1 << endl;
    cout << num2 << endl;

    cout << &num1 << endl;      // num1의 메모리 주소 값
    cout << &num2 << endl;      // num2의 메모리 주소 값
}
```

변수라고 하는 것은 할당된 메모리 공간에 붙여진 이름이다.
우리는 이 이름을 가지고 해당 메모리 공간에 접근이 가능해진다.
그러면 참조자는 무엇일까? 참조자는 할당된 하나의 메모리 공간에 다른 이름을 붙이는 것을 말한다.

Basic

ROBIT
ROBOT SPORT GAME TEAM

▶ 참조자(reference)

```
1   #include<iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       int &num1 = 2;    //상수를 참조 불가능
8
9       int &num1;       //참조하는 값이 없는 경우 선언 불가능
10
11      int &num1 = NULL;        // NULL 값도 참조 불가능
12  }
```

포인터와는 다르게 특정 상수 값을 참조할 수 없음. (참조하는 값이 없어도 정의 안됨.)
NULL 값 또한 참조 할 수 없다.
포인터와 비교해 할 수 있는 기능은 축소되었지만 이름 그대로 참조(reference) 기능에 집중됨.

Basic

ROBIT
ROBOT SPORT GAME TEAM

▶ 참조자(reference)

```
int a = 10;
int &another_a = a; // another_a 는 이제 a 의 참조자!

int b = 3;
another_a = b; // ??
```

포인터와 다르게 한 번 정의된 참조자는 다른 것을 또 참조할 수 없다.

실습-reference

ROBIT
ROBOT SPORT GAME TEAM

- 참조자(reference)

```cpp
#include <iostream>

int main() {
  int a = 3;
  int& another_a = a;

  another_a = 5;
  std::cout << "a : " << a << std::endl;
  std::cout << "another_a : " << another_a << std::endl;

  return 0;
}
```

a와 another a는 항상 같은 값을 반환.
(이는 a나 another a를 다른 값으로 바꿔도 마찬가지.)

Basic

ROBIT ROBOT SPORT GAME TEAM

- Call by reference VS value

pass by reference

cup =

fillCup( )

pass by value

cup =

fillCup( )

대표적인 예시로 함수의 매개변수 전달 방식이 있음.
C에서 포인터를 통해 주소를 전달하는 방식이 Call by reference.
그냥 변수 값 자체를 전달하면 Call by value이다.

Basic

ROBIT ROBOT SPORT GAME TEAM

- Call by reference VS value

```cpp
#include<iostream>

using namespace std;

void Swap(int a, int b)      // Call-by-value
{                            // 외부 값을 변화시킬 수 없다.
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int num1 = 5;
    int num2 = 10;

    Swap(num1, num2);        // 값의 변화가 없다.

    cout << num1 << endl;
    cout << num2 << endl;
}
```

```cpp
#include<iostream>

using namespace std;

void Swap(int &a, int &b)      // Call-by-reference
{                              // 외부 값을 변화 가능
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int num1 = 5;
    int num2 = 10;

    Swap(num1, num2);        // 값이 바뀐다.

    cout << num1 << endl;
    cout << num2 << endl;
}
```

C언어

절차 지향

C++

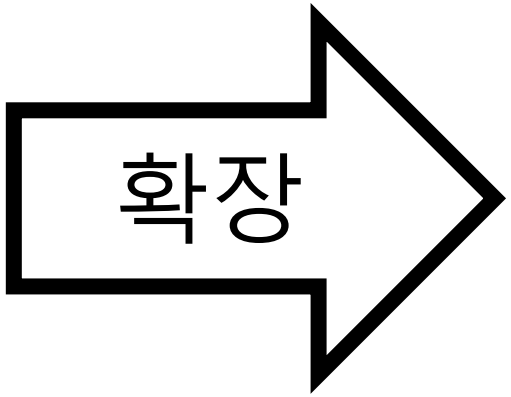
객체 지향

C언어

구조체

키워드로 Struct 사용

멤버로 여러 형식의 변수 포함 가능



C++

class

키워드로 class 사용

멤버로 여러 형식의 변수와
함수 포함 가능

Class 변수 = 객체(object)

데이터 접근에 제한을 둘 수 있음

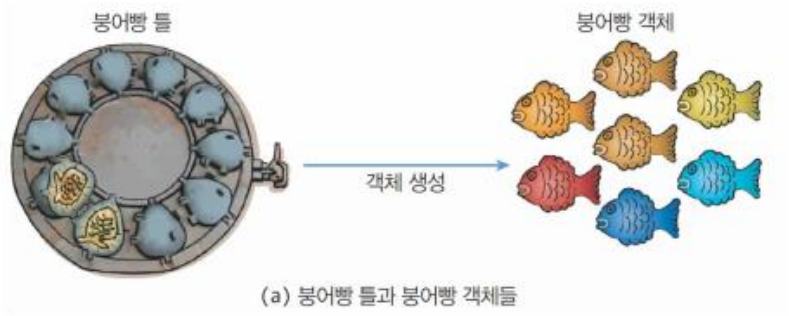
C++

객체(object)

C++

Class : 객체

↳ 설계도 : 제품



(a) 붕어빵 틀과 붕어빵 객체들

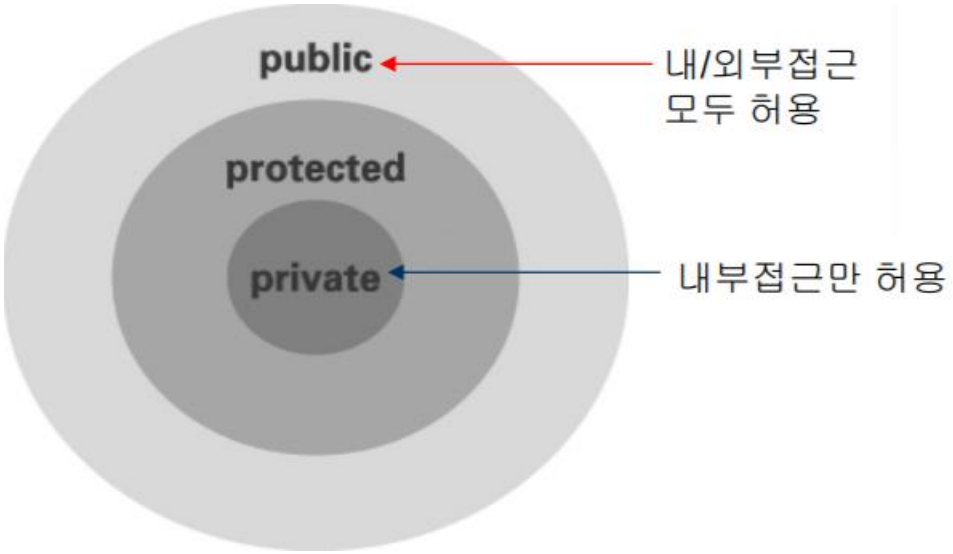
cf)

- 클래스
  - 객체를 만들어내기 위해 정의된 설계도, 틀
  - 클래스는 객체가 아님. 실체도 아님
  - 멤버 변수와 멤버 함수 선언

- 객체
  - 객체는 생성될 때 클래스의 모양을 그대로 가지고 탄생
  - 멤버 변수와 멤버 함수로 구성
  - 메모리에 생성, 실체(instance)라고도 부름
  - 하나의 클래스 틀에서 찍어낸 여러 개의 객체 생성 가능
  - 객체들은 상호 별도의 공간에 생성

C++

Class 멤버 접근 제어



<접근 지정자>

Public : 공용부분
객체 외부에서 사용될 수 있는 자료, 함수를 정의함

Private :전용부분
함부로 변경되어서는 안될 자료와 객체 외부에서
호출되어서는 안될 멤버함수를 정의함.
private 영역에서 정의된 자료형이나 함수는
해당 객체 내부의 멤버함수만이 사용 가능.
외부에 대해 자료의 정보가 은폐됨.
클래스 정의 시 키워드 private이 생략되면
Public이 나오기 전까지의 부분을 전용멤버로 간주.

C++

객체 멤버 접근

```
#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
}
```

Class 선언부 -hpp or h

ROBIT
ROBOT SPORT GAME TEAM

변수와, 함수의 구조를 정의함.

```
class 클래스명
{
[private:]
자료 선언;
함수 선언;
public:
자료 선언;
함수 선언;
};
```

ex

```
class Car
{
    private:
        int wheels;
        int price;
    public:
        void setWheels(int _wheels);
        void setPrice(int _price);
        int getWheels(void);
        int getPrice(void);
};
```

Class 구현부 -cpp

실제로 실행되는 함수의 내용 등을 표현함.

```
자료형 클래스명::함수명(매개변수)
{
   …함수 내용 …
}
```

cf) ❖ 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; }
   ....
};
```

ex

```
#include "car.h"
void Car::setWheels(int _wheels) {
        this->wheels = _wheels;
}
void Car::setPrice(int _price) {
        this->price = _price;
}
int Car::getWheels(void) {
        return this->wheels;
}
int Car::getPrice(void) {
        return this->price;
}
```

C++

Main 함수에서의 Class 이용

```
int main(void)
{
    Class명 객체명;
    return 0;
}
```

ex

```
int main(int argc, char* argv[ ])
{
        Car myCar;
        myCar.setWheels(4);
        myCar.setPrice(10000);
        std::cout << "My car has " << myCar.getWheels() << " wheels."<< std::endl;
        return 0;
}
```

Class

overloading

- 함수 중복
  - 동일한 이름의 함수가 공존
    - 다형성
    - C 언어에서는 불가능
  - function overloading
  - 함수 중복이 가능한 범위
    - 보통 함수들 사이
    - 클래스의 멤버 함수들 사이
    - 상속 관계에 있는 기본 클래스와 파생 클래스의 멤버 함수들 사이
- 함수 중복 성공 조건
  - 중복된 함수들의 이름 동일
  - 중복된 함수들의 매개 변수 타입이 다르거나 개수가 달라야 함
  - 리턴 타입은 함수 중복과 무관

Class
overloading

```
int sum(int a, int b, int c) {
   return a + b + c;
}

double sum(double a, double b) {
   return a + b;
}

int sum(int a, int b) {
   return a + b;
}
```

성공적으로 중복된 sum() 함수들

```
int main(){
   cout << sum(2, 5, 33);

   cout << sum(12.5, 33.6);

   cout << sum(2, 6);
}
```

중복된 sum() 함수 호출.
컴파일러가 구분

```
int sum(int a, int b) {
   return a + b;
}
double sum(int a, int b) {
   return (double)(a + b);
}
```

함수 중복 실패

```
int main() {
   cout << sum(2, 5);
}
```

?

컴파일러는 어떤 sum() 함수를 호출하는지 구분할 수 없음

Class

overloading

- 동일한 이름을 사용하면 함수 이름을 구분하여 기억할 필요 없고, 함수 호출을 잘못하는 실수를 줄일 수 있음

```
void msg1() {
   cout << "Hello";
}
void msg2(string name) {
   cout << "Hello, " << name;
}
void msg3(int id, string name) {
   cout << "Hello, " << id << " " << name;
}
```

(a) 함수 중복하지 않는 경우

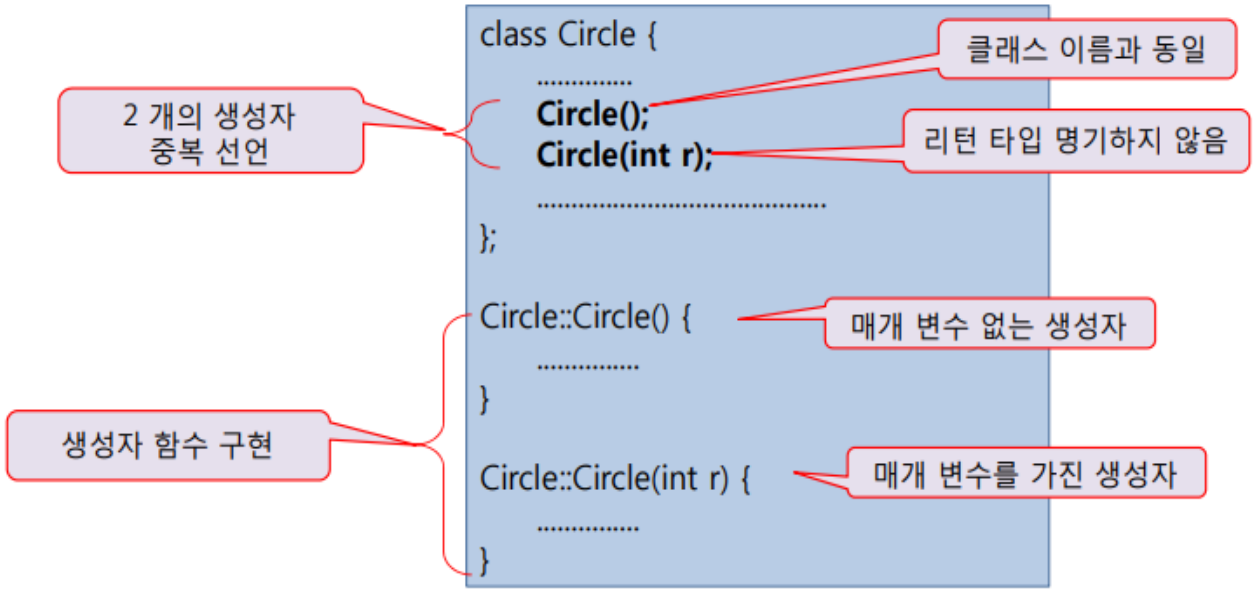
```
void msg() {
   cout << "Hello";
}
void msg(string name) {
   cout << "Hello, " << name;
}
void msg(int id, string name) {
   cout << "Hello, " << id << " " << name;
}
```

(b) 함수 중복한 경우

함수 중복하면 함수 호출의 편리함.오류 가능성 줄임

Class
생성자

- 생성자(constructor)
  - 객체가 생성되는 시점에서 자동으로 호출되는 멤버 함수
  - 클래스 이름과 동일한 멤버 함수



- 생성자의 목적
  - 객체가 생성될 때 객체가 필요한 초기화를 위해
    - 멤버 변수 값 초기화, 메모리 할당, 파일 열기, 네트워크 연결 등
- 생성자 이름
  - 반드시 클래스 이름과 동일
- 생성자는 리턴 타입을 선언하지 않는다
  - 리턴 타입 없음. void 타입도 안됨
- 객체 생성 시 오직 한 번만 호출
  - 자동으로 호출됨. 임의로 호출할 수 없음. 각 객체마다 생성자 실행
- 생성자는 중복 가능
  - 생성자는 한 클래스 내에 여러 개 가능
  - 중복된 생성자 중 하나만 실행
- 생성자가 선언되어 있지 않으면 기본 생성자 자동으로 생성
  - 기본 생성자 - 매개 변수 없는 생성자
  - 컴파일러에 의해 자동 생성

Class

생성자

▷멤버 변수 초기화 방법

```
class Point {
   int x, y;

public:
   Point();
   Point(int a, int b);
};
```

(1) 생성자 코드에서 멤버 변수 초기화

```
Point::Point() { x = 0; y = 0; }
Point::Point(int a, int b) { x = a; y = b; }
```

(2) 생성자 서두에 초기값으로 초기화

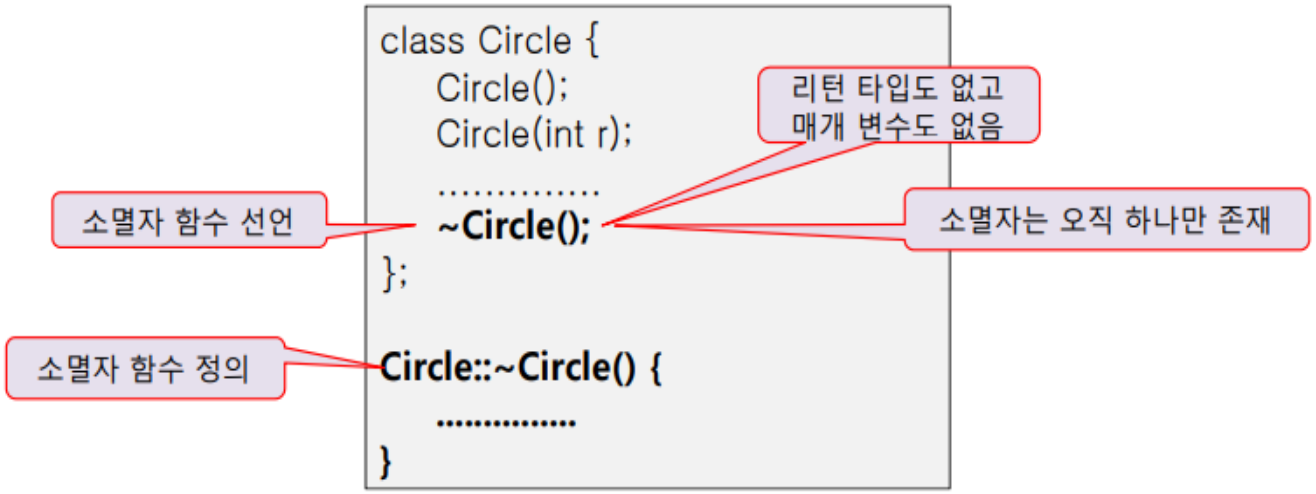
```
Point::Point() : x(0), y(0) { // 멤버 변수 x, y를 0으로 초기화
}
Point::Point(int a, int b)    // 멤버 변수 x=a로, y=b로 초기화
   : x(a), y(b) {          // 콜론(:) 이하 부분을 밑줄에 써도 됨
}
```

(3) 클래스 선언부에서 직접 초기화

```
class Point {
   int x=0; y=0; // 클래스 선언부에서 x, y를 0으로 초기화
public:
   ...
};
```

Class
소멸자

- 소멸자
  - 객체가 소멸되는 시점에서 자동으로 호출되는 함수
    - 오직 한번만 자동 호출, 임의로 호출할 수 없음
    - 객체 메모리 소멸 직전 호출됨

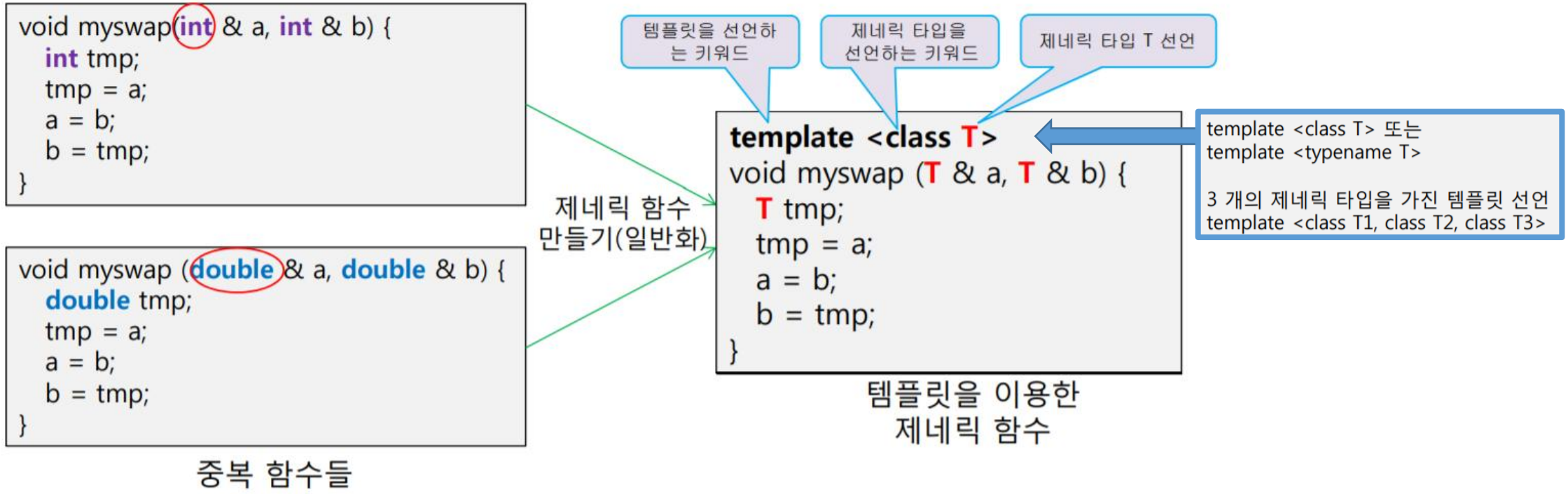


- 소멸자의 목적
  - 객체가 사라질 때 마무리 작업을 위함
  - 실행 도중 동적으로 할당 받은 메모리 해제, 파일 저장 및 닫기, 네트워크 닫기 등
- 소멸자 함수의 이름은 클래스 이름 앞에 ~를 붙인다.
  - 예) Circle::~Circle() { ... }
- 소멸자는 리턴 타입이 없고, 어떤 값도 리턴하면 안됨
  - 리턴 타입 선언 불가
- 중복 불가능
  - 소멸자는 한 클래스 내에 오직 한 개만 작성 가능
  - 소멸자는 매개 변수 없는 함수
- 소멸자가 선언되어 있지 않으면 기본 소멸자가 자동 생성
  - 컴파일러에 의해 기본 소멸자 코드 생성
  - 컴파일러가 생성한 기본 소멸자 : 아무 것도 하지 않고 단순 리턴

C++

▶ 템플릿

- 변수나 매개 변수의 자료형만 다르고, 코드 부분이 동일한 함수, 클래스를 일반화 시킬 수 있다.



C++

ROBIT
ROBOT SPORT GAME TEAM

▶ STL(Standard Template Library)

- 표준 템플릿 라이브러리
  - 컨테이너 - 템플릿 클래스
    - 데이터를 담아두는 자료 구조를 표현한 클래스
    - 리스트, 큐, 스택, 맵, 셋, 벡터
  - iterator - 컨테이너 원소에 대한 포인터
    - 컨테이너의 원소들을 순회하면서 접근하기 위해 만들어진 컨테이너 원소에 대한 포인터
  - 알고리즘 - 템플릿 함수
    - 컨테이너 원소에 대한 복사, 검색, 삭제, 정렬 등의 기능을 구현한 템플릿 함수
    - 컨테이너의 멤버 함수 아님

〈표 10-1〉 STL 컨테이너의 종류

컨테이너 클래스	설명	헤더 파일
vector	가변 크기의 배열을 일반화한 클래스	<vector>
deque	앞뒤 모두 입력 가능한 큐 클래스	<deque>
list	빠른 삽입/삭제 가능한 리스트 클래스	<list>
set	정렬된 순서로 값을 저장하는 집합 클래스. 값은 유일	<set>
map	(key, value) 쌍을 저장하는 맵 클래스	<map>
stack	스택을 일반화한 클래스	<stack>
queue	큐를 일반화한 클래스	<queue>

〈표 10-2〉 STL iterator의 종류

iterator의 종류	iterator에 ++ 연산 후 방향	read/write
iterator	다음 원소로 전진	read/write
const_iterator	다음 원소로 전진	read
reverse_iterator	지난 원소로 후진	read/write
const_reverse_iterator	지난 원소로 후진	read

〈표 10-3〉 STL 알고리즘 함수들

copy	merge	random	rotate
equal	min	remove	search
find	move	replace	sort
max	partition	reverse	swap

C++

ROBIT
ROBOT SPORT GAME TEAM

▶ STL(Standard Template Library)

- vector

: 동적 할당 필요 없이 사용할 수 있는 가변 길이 배열!

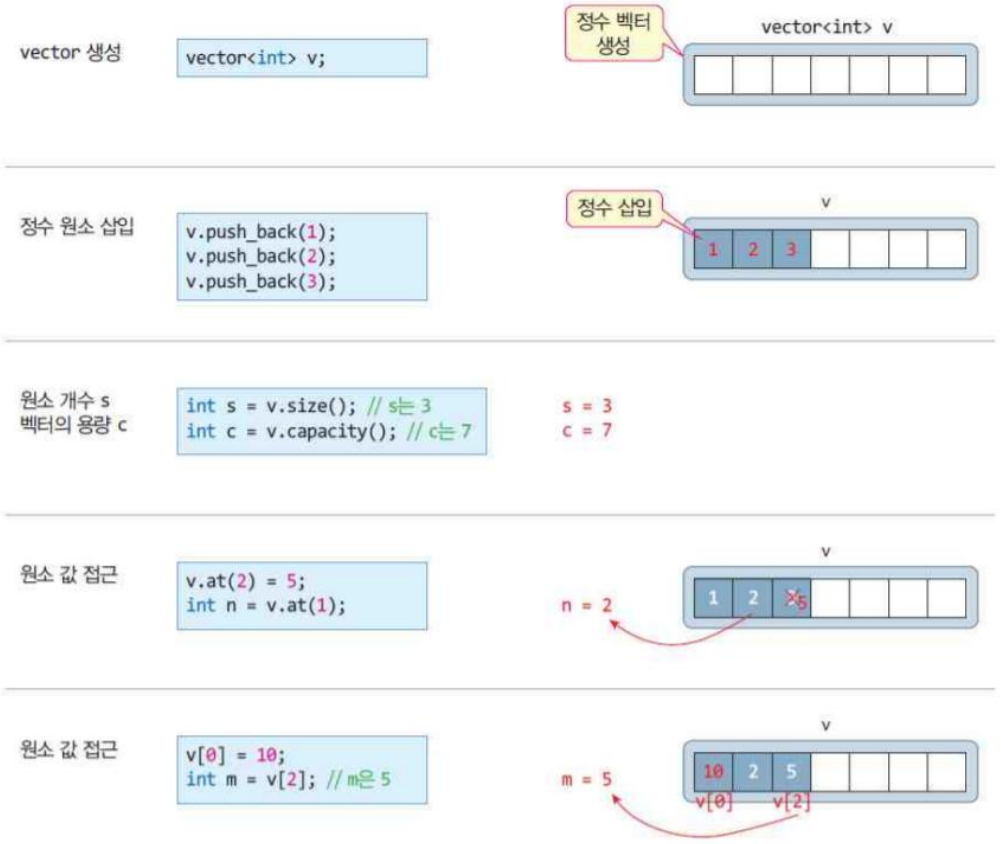
- 가변 길이 배열을 구현한 제네릭 클래스
  - 개발자가 벡터의 길이에 대한 고민할 필요 없음
- 원소의 저장, 삭제, 검색 등 다양한 멤버 함수 지원
- 벡터에 저장된 원소는 인덱스로 접근 가능
  - 인덱스는 0부터 시작

C++

▶ STL(Standard Template Library)

- vector

멤버와 연산자 함수	설명
push_back(element)	벡터의 마지막에 element 추가
at(int index)	index 위치의 원소에 대한 참조 리턴
begin()	벡터의 첫 번째 원소에 대한 참조 리턴
end()	벡터의 끝(마지막 원소 다음)을 가리키는 참조 리턴
empty()	벡터가 비어 있으면 true 리턴
erase(iterator it)	벡터에서 it가 가리키는 원소 삭제. 삭제 후 자동으로 벡터 조절
insert(iterator it, element)	벡터 내 it 위치에 element 삽입
size()	벡터에 들어 있는 원소의 개수 리턴
operator[]()	지정된 원소에 대한 참조 리턴
operator=()	이 벡터를 다른 벡터에 치환(복사)



C++

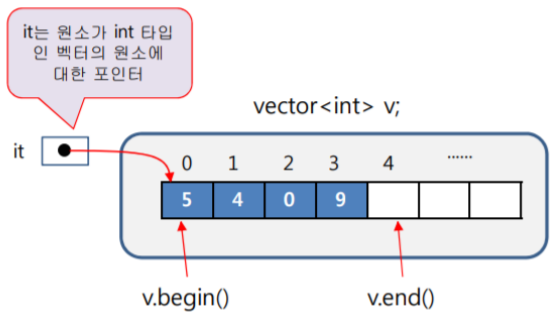
ROBIT ROBOT SPORT GAME TEAM

▶ STL(Standard Template Library)

- vector

- ❖ iterator란?
  - ▪ 반복자라고도 부름
  - ▪ 컨테이너의 원소를 가리키는 포인터
- ❖ iterator 변수 선언
  - ▪ 구체적인 컨테이너를 지정하여 반복자 변수 생성

```
vector<int>::iterator it;
it = v.begin();
```




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

int main() {
   vector<int> v; // 정수 벡터 생성
   v.push_back(1);
   v.push_back(2);
   v.push_back(3);

   vector<int>::iterator it; // 벡터 v의 원소에 대한 포인터 it 선언

   for(it=v.begin(); it != v.end(); it++) { // iterator를 이용하여 모든 원소 탐색
      int n = *it; // it가 가리키는 원소 값 리턴
      n = n*2; // 곱하기 2
      *it = n; // it가 가리키는 원소에 값 쓰기
   }

   for(it=v.begin(); it != v.end(); it++)  // 벡터 v의 모든 원소 출력
      cout << *it << ' ';
   cout << endl;
}
```

```
2 4 6
```

C++

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
    - new로 할당 받은 메모리 반환
    - 객체의 동적 소멸 - 소멸자 호출 뒤 객체를 힙에 반환

C++

동적 할당

- 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
```

C++

동적 할당

▶ 정수

```
#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
```

▶ 정수형 배열

```
#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
}
```

C++

동적 할당

▶ 객체

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

```
#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
```

C++

동적 할당

▶ 객체 배열

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

- 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)

C++

동적 할당

- 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.
```

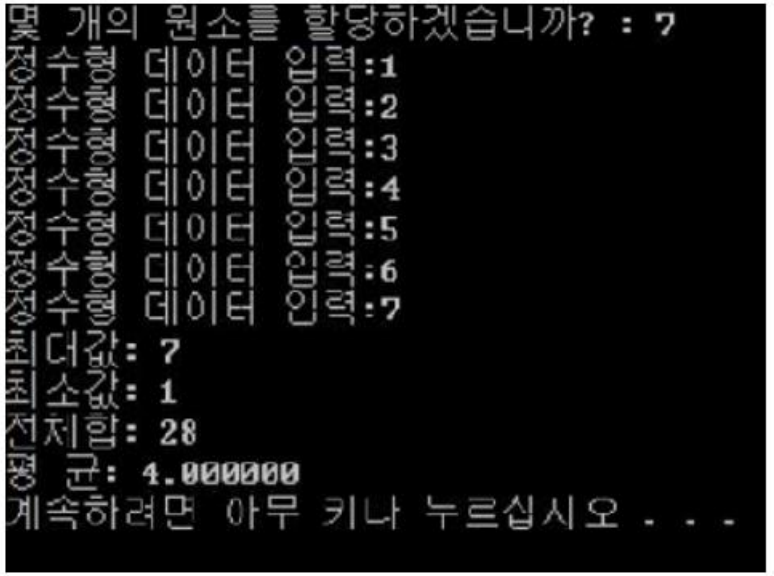
과제

ROBIT
ROBOT SPORT GAME TEAM

▶ 과제 1

: 다음 과제를 C++로, Class를 이용하여 작성하시오
모든 변수와 함수를 클래스 멤버변수, 멤버함수로 작성할 것
(코드에서 생성하는 클래스 객체는 1개)

동적할당을 이용하여 배열의 값들의 최대, 최소, 전체 합, 평균을 구하시오.



과제에서 각각의 클래스는

헤더파일과 소스 파일로 따로 나누어

작성할 것

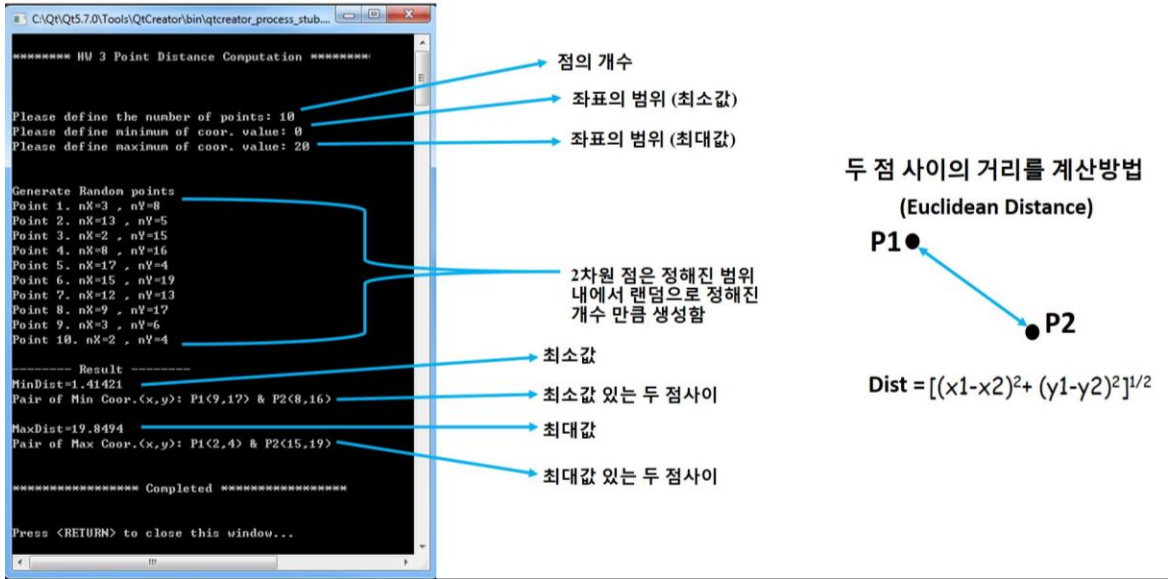
과제

ROBIT ROBOT SPORT GAME TEAM

▶ 과제 2

점들의 집합에서 임의의 두 점 사이의 거리를 계산할 때, 최소값과 최대값을 구하고 그것에 해당하는 각각의 두 점을 구하는 클래스를 작성하시오.

- 메모리 동적할당을 이용함
- structure를 이용하여 2차원 좌표를 구현함
- 2차원 점은 정해진 범위 내에서 랜덤으로 정해진 개수 만큼 생성함
- 2차원 좌표의 범위와 점의 개수는 "cin"으로 입력



과제에서 각각의 클래스는
헤더파일과 소스 파일로 따로 나누어
작성할 것

과제

▶ 과제 3

과제에서 각각의 클래스는
헤더파일과 소스 파일로 따로 나누어
작성할 것

ROBIT
ROBOT SPORT GAME TEAM

몬스터 사냥
플레이어 클래스, 몬스터 클래스를 각각 생성하고 플레이어가 몬스터를 사냥하는 스크립트를 작성.

명령어를 입력 받아 플레이어가 이동, 공격, 스테이터스 표기를 실행

```
Type Command(A/U/D/R/L/S)
D
Y Position -1 moved!
Type Command(A/U/D/R/L/S)
R
X Position 1 moved!
Type Command(A/U/D/R/L/S)
S
HP:50
MP:10
Position:1,-1
Type Command(A/U/D/R/L/S)
U
Y Position 1 moved!
Type Command(A/U/D/R/L/S)
D
Y Position -1 moved!
Type Command(A/U/D/R/L/S)
L
X Position -1 moved!
Type Command(A/U/D/R/L/S)
S
HP:50
MP:10
Position:0,-1
Type Command(A/U/D/R/L/S)
A
공격실패!
Type Command(A/U/D/R/L/S)
A
공격실패!
Type Command(A/U/D/R/L/S)
S
HP:50
MP:8
Position:0,-1
Type Command(A/U/D/R/L/S)
```

```
Type Command(A/U/D/R/L/S)
A
공격 성공!
남은 체력:40
Type Command(A/U/D/R/L/S)
A
공격 성공!
남은 체력:30
Type Command(A/U/D/R/L/S)
A
공격 성공!
남은 체력:20
Type Command(A/U/D/R/L/S)
A
공격 성공!
남은 체력:10
Type Command(A/U/D/R/L/S)
A
공격 성공!
남은 체력:0
Monster Die!!
```

```
Type Command(A/U/D/R/L/S)
S
HP:50
MP:0
Position:0,-1
Type Command(A/U/D/R/L/S)
A
MP 부족!


...Program finished with exit code 0
Press ENTER to exit console.
```

공격을 해서 몬스터의 HP가 0이 되면 프로그램 종료.
공격 한 번당 MP 1소모, MP가 부족한 상태에서 공격하면
프로그램 종료.

과제

ROBIT
ROBOT SPORT GAME TEAM

▶ 과제 3

```
class Player
{


    public:
        int HP,MP,x,y;
        Player();
        Player(int x,int y);
        void Attack(Monster &target);
        void Show_Status();
        void X_move(int move);
        void Y_move(int move);
};
```

몬스터와 캐릭터 클래스는 위와 같이 구성.

```
class Monster
{

    public:
        int HP,x,y;
        Monster();
        Monster(int x,int y,int HP);
        int Be_Attacked();
};
```

```
Player player(0,0);
Monster monster(5,4,50);
```

몬스터와 캐릭터 초기값은 위와 같이 설정.

과제

ROBIT
ROBOT SPORT GAME TEAM

다음날 출근(7시) 전까지 예시 참고하여 GIT 제출

https://github.com/lee-sunkyoung/robit_intern_LSK
자신의 이름으로 패키지 생성 후 개인톡으로 링크 제출

레포트 작성 시 **알아보기 쉽도록 작성할 것.**

- 목차 혹은 제목, 번호 붙이기 필수
- 코드 복사 붙여넣기 금지
- 실행 결과 화면 첨부, 설명 작성 후 pdf로 제출.

*** 제출 형식 틀리거나 기한 지키지 못한 과제는 사유불문 채점 안함.**