



#8. 클래스와 객체

C++ Programming Tutor

8주차 수업 안내



- 클래스와 객체
 - 클래스와 객체의 기본
 - 클래스의 정의
 - 객체의 생성과 사용
 - 생성자와 소멸자
 - 접근 제어
 - 정적 멤버
 - 클래스와 객체 자세히 살펴보기
 - 멤버 함수
 - 객체의 배열
 - 객체의 동적인 생성
 - 클래스에 넣을 수 있는 다른 것들
 - This 포인터

클래스의 정의



Point 구조체를 Point 클래스로 바꿔보자!

클래스의 정의



Point 구조체?

그게 뭐죠? 먹는 건가요?

클래스의 정의



예전에 구조체 배울 당시에...

```
struct Point {  
    int x; // x 좌표  
    int y; // y 좌표  
};
```

이런거 배웠잖아요!

클래스의 정의



이제 이걸 클래스로 바꿔볼까요?

클래스의 정의



```
class Point {  
public:  
    // 멤버 변수들  
    int x, y;  
  
    // 멤버 함수  
    void Print() {  
        std::cout << "(" << x << ", " << y << ")\\n";  
    }  
};
```

구조체에 비해서 뭐가 달라졌죠?

class라는 키워드를 사용해서 정의!

접근 제어와 관련된 키워드(public)를 사용!

멤버 함수 역시 정의!



클래스의 정의

```
class Point {  
public:  
    // 멤버 변수들  
    int x, y;  
  
    // 멤버 함수  
    void Print() {  
        std::cout << "(" << x << ", " << y << ")\\n";  
    }  
};
```

클래스의 정의를 의미

접근 제어와 관련된 키워드

클래스의 정의 안에서 정의된 함수는 멤버 함수가 됨

멤버 변수(Member Variables)

구조체에서 멤버라고 불리던 것이
클래스에서 멤버 변수에서 불리는 것 뿐
(클래스에는 멤버 함수도 있기 때문)

멤버 함수(Member Functions)

클래스의 멤버로써 존재하는 함수

접근 제어(Access Control)

클래스의 외부에서 멤버에 접근할 수 있는지를 설정
정보 은닉과 관련

객체의 생성과 사용



클래스는 만들었는데, 이거 어찌 써요?



객체의 생성과 사용



객체를 생성해서 사용하면 되겠지...



객체의 생성과 사용



그런데 객체는 어떻게 생성하고 사용하죠?

객체의 생성과 사용



```
int main() {
    // 객체를 생성
    Point pt;

    // pt를 초기화
    pt.x = 100;
    pt.y = 200;

    // pt의 내용을 출력
    pt.Print();

    return 0;
}
```

객체의 생성과 사용



근데요... 제가 생각해 봤는데요...

일반 함수랑 멤버 함수랑 뭐가 다르죠?

멤버 함수는 기본적으로 일반 함수랑 같아요
(오버로드도 할 수 있고, 디폴트 인자도 사용할 수 있고...)



다만 클래스의 멤버라서 이런 차이점이 있어요

멤버 함수를 호출하기 위해서는 객체의 이름을 명시!

멤버 함수 안에서는 객체의 이름을 명시하지 않고

멤버 변수에 접근 가능!

멤버 함수 안에서는 외부에서 접근을

거부한 멤버에도 접근 가능!

객체의 생성과 사용



저는 더러우면 생활을 못해서...
클래스 안도 깨끗하게 쓰고 싶어요
그래서 멤버 함수를 밖에서 정의하고 싶은데...



객체의 생성과 사용



```
class Point {  
    // 멤버 변수  
    int x, y;  
  
    // 멤버 함수  
    void Print();  
};  
  
void Point::Print() {  
    std::cout << "(" << x << ", " << y << ")\\n";  
}
```

멤버 함수를 클래스 밖에서 정의하고 싶다고요?

다음 두 가지만 하세요!

클래스 정의 안에서는

멤버 함수의 원형만 남겨두고!

클래스 정의 밖에서는

범위 지정 연산자(:)를 사용해서 함수를 정의!

범위 지정 연산자(Scope Resolution Operator)란?

void Print()

보통의 함수 Print()를 의미

void Point::Print()

Point 클래스의 멤버 함수 Print()를 의미

즉, Point 클래스의 멤버 함수라는 것을 명시하는 연산자!

객체의 생성과 사용



Q : 객체를 사용해서 초기화나 대입은 어떻게 하나요?

A : 구조체랑 똑같은데요?



객체의 생성과 사용



```
int main() {  
    Point pt1, pt2;  
    pt1.x = 100; pt1.y = 100;  
    pt2.x = 200; pt2.y = 200;  
  
    // pt1을 사용해서 새로운 pt3를 초기화  
    Point pt3 = pt1;  
    pt3.Print();  
  
    // pt2을 pt3에 대입  
    pt3 = pt2;  
    pt3.Print();  
  
    return 0;  
}
```

객체의 초기화(Initialization)와 대입(Substitution)이 다른 개념이라는 건 아시나요?

```
Point pt3 = pt1;
```

객체를 정의하면서 '초기화'

```
pt3 = pt2;
```

이미 생성한 객체에 '대입'

왜 알아야 될까요? 그건 뒤에서...

생성자(Constructor)와 소멸자(Destructors)는
특별한 기능을 가진 멤버 함수!

생성자는 객체를 생성할 때 자동적으로 호출되는 함수!
소멸자는 객체가 소멸될 때 자동으로 호출되는 함수!



생성자와 소멸자



오... 짜네요!!!

근데 어디에 쓰죠?



일반적으로 생성자에서 하는 일은
객체가 제대로 동작할 수 있게 ‘준비’하는 일!

멤버 변수 초기화를 한다거나...

(정수형 변수를 0으로 초기화)

(포인터 변수를 NULL로 초기화)

동적 메모리 할당을 한다거나...

반대로 소멸자에서 하는 일은
‘정리’에 해당하는 일!

할당한 동적 메모리를 해제하거나...
객체가 파일을 열어서 사용했다면
그 파일을 닫아주거나...

생성자와 소멸자



읽기 귀찮은 이들을 위해 정리하면...

생성자에서는 '준비'를 하고, 소멸자에서는 '정리'를 한다!



생성자를 사용하는 방법?

한 클래스에는...

생성자가 없을 수도 있고 1개 이상 존재할 수도 있다

가장 간단한 형태를 띠고 있는

'디폴트 생성자(Default Constructor)'

생성자와 소멸자



```
class Point {  
public:  
    int x, y;  
  
    // 생성자  
    Point();  
};  
  
Point::Point() {  
    x = 0;  
    y = 0;  
}
```

생성자와 소멸자



멤버 함수와 비슷하게 생겼는데, 조금 이상하네...



생성자의 특징

생성자는 클래스와 동일한 이름!

생성자는 반환 값이 없다!

생성자와 소멸자



```
int main() {  
    Point pt;  
  
    return 0;  
}
```

Point 객체를 생성할 때 생성자가 자동으로 호출!

```
Point pt;  
pt.x = 0;  
pt.y = 0;
```

생성자와 소멸자



아무런 인자가 없는 생성자를 ‘디폴트 생성자’라고 부름!

그럼 인자가 있는 생성자는?

또 복사 생성자는?



생성자와 소멸자



```
class Point {  
public:  
    int x, y;  
  
    // 생성자  
    Point(int _x, int _y);  
};  
  
Point::Point(int _x, int _y) {  
    x = _x;  
    y = _y;  
}
```

생성자와 소멸자



```
int main() {  
    Point pt(3, 5);  
  
    return 0;  
}
```

역시 Point 객체를 생성할 때 생성자가 자동으로 호출!

```
Point pt;  
pt.x = 3;  
pt.y = 5;
```

복사 생성자(Copy Constructor)는
다른 객체로부터 값을 복사해서 초기화하는 데 사용!

생성자와 소멸자



```
class Point {  
public:  
    int x, y;  
  
    // 복사 생성자  
    Point(const Point& pt);  
};  
  
Point::Point(const Point& pt) {  
    x = pt.x;  
    y = pt.y;  
}
```

생성자와 소멸자



```
int main() {  
    Point pt1(100, 100), pt2(200, 200);  
  
    // pt1을 사용해서 새로운 pt3을 초기화  
    Point pt3 = pt1;  
    pt3.Print();  
  
    // pt2을 pt3에 대입  
    pt3 = pt2;  
    pt3.Print();  
  
    return 0;  
}
```

생성자와 소멸자



복사 생성자는 자신과 동일한 타입의 객체에 대한
레퍼런스를 인자로 받는 생성자!

`Point(Point& pt);`

`Point(const Point& pt);`

두 가지 모두 복사 생성자이지만...
`const Point&`의 형식을 권장!

생성자와 소멸자



```
Point pt3 = pt1;
```

복사 생성자가 호출(초기화)

```
pt3 = pt2;
```

복사 생성자가 호출되지 않음(대입)

생성자와 소멸자



초기화와 대입을 구별할 줄 알아야 되는 이유?
컴퓨터 내부적으로는 완전히 다른 연산이기 때문!



생성자와 소멸자



Q : 근데요...

복사 생성자를 만들지 않아도 초기화 잘 되잖아요?

그런데 굳이 이거 왜 쓰는 거죠?



생성자와 소멸자



A : 그건...

1 : 1 복사 말고 다른 방식으로 복사하고 싶어서요



`strcpy()` : 문자열 복사 함수

문자열의 포인터만 복사하는 경우

같은 문자열을 공유

얕은 복사(Shallow Copy)

새로운 문자열을 하나 만들어서 복사해주는 경우

다른 문자열을 가지게 됨

깊은 복사(Deep Copy)

얕은 복사(Shallow Copy)인 경우

```
String::String(const String& s) {  
    p = s.p;  
}
```

깊은 복사(Deep Copy)인 경우

```
String::String(const String& s) {  
    p = new char[s.size() + 1];  
    strcpy(p, s.p);  
}
```

생성자와 소멸자



Q : 생성자 요거요거... 참 편해서 좋네요!
근데 쓰지 않아도 프로그램은 잘 되잖아요?
배우기도 귀찮은데 그냥 안 쓸래요...



생성자와 소멸자



A : '반드시' 생성자가 필요할 때가 있지롱!

멤버 변수 중에...

const 속성을 가진 것이 있거나 레퍼런스 타입이 있다면!

const 속성을 가진 변수와 레퍼런스 변수는
'반드시 초기화되어야 한다'는 공통점이 존재!

생성자와 소멸자



일반 변수라면 변수를 정의할 때 초기화하면 되는데...

멤버 변수는 어디서 해야 하지?

객체를 생성하자 마자 초기화하면 될까?

생성자와 소멸자



```
class Test {  
public:  
    const int maxCount;  
    int& ref;  
    int sample;  
};  
  
void main() {  
    Test test;  
  
    test.maxCount = 100;  
    test.ref = test.sample;  
}
```

생성자와 소멸자



설명	파일	줄	열	프로젝트
✖ 1 error C2512: 'Test' : 사용할 수 있는 적절한 기본 생성자가 없습니다.	test.cpp	11	1	Test
✖ 2 error C3892: 'test' : const인 변수에 할당할 수 없습니다.	test.cpp	13	1	Test
📄 3 IntelliSense: 식이 수정할 수 있는 lvalue여야 합니다.	test.cpp	13	4	Test



생성자와 소멸자



그렇다면 생성자에서 초기화하면 될까?

생성자와 소멸자



```
class Test {  
public:  
    const int maxCount;  
    int& ref;  
    int sample;  
  
    Test();  
};  
  
Test::Test() {  
    maxCount = 100;  
    ref = sample;  
}
```

생성자와 소멸자



설명	파일	줄	열	프로젝트
✖ 1 error C2758: 'Test::maxCount' : 생성자 기본/멤버 이니셜라이저 목록에 초기화해야 합니다.	test.cpp	12	1	Test
✖ 2 error C2758: 'Test::ref' : 생성자 기본/멤버 이니셜라이저 목록에 초기화해야 합니다.	test.cpp	12	1	Test
✖ 3 error C2166: l-value가 const 개체를 지정합니다.	test.cpp	13	1	Test
✖ 4 IntelliSense: "Test::Test()" 이(가) 다음에 대한 이니셜라이저를 제공하지 않습니다.	test.cpp	12	14	Test
✖ 5 IntelliSense: 식이 수정할 수 있는 lvalue여야 합니다.	test.cpp	13	4	Test



생성자와 소멸자



아... 슈ㅂ... 뭐 어쩌라고!!!

생성자와 소멸자



그럴 때는 생성자의 초기화 리스트를 사용하세요!

생성자와 소멸자



```
class Test {  
public:  
    const int maxCount;  
    int& ref;  
    int sample;  
  
    Test();  
};  
  
Test::Test() : maxCount(100), ref(sample) {  
    sample = 200;  
}
```

생성자와 소멸자



```
int main() {
    Test test;

    std::cout << "test.maxCount = "
        << test.maxCount << "\n";
    std::cout << "test.ref = "
        << test.ref << "\n";

    return 0;
}
```

생성자와 소멸자



```
C:\Windows\system32\cmd.exe
test.maxCount = 100
test.ref = 200
계속하려면 아무 키나 누르십시오 . . .
```



반드시 초기화해야 하는 멤버 변수들은
생성자의 초기화 리스트를 사용해서 초기화한다!

```
Test::Test() : maxCount(100), ref(sample) {  
    sample = 200;  
}
```

콜론(:)을 찍어준 후에 멤버 변수의 이름을 적고...
이어지는 괄호 안에 초기 값을 넣어주고...

생성자와 소멸자



생성자의 초기화 리스트는요...

인자가 있는 생성자에서도 사용할 수 있어요!

(전달하는 인자를 사용해서 초기화할 수도 있죠!)

생성자와 소멸자



```
class Test {  
public:  
    const int maxCount;  
    int& ref;  
    int sample;
```

```
    Test(int count, int& num);  
};
```

```
Test::Test(int count, int& num)  
: maxCount(count), ref(num) {  
    sample = 200;  
}
```

생성자와 소멸자



```
int main() {
    int number = 400;
    Test test(300, number);

    std::cout << "test.maxCount = "
        << test.maxCount << "\n";
    std::cout << "test.ref = "
        << test.ref << "\n";

    return 0;
}
```

생성자와 소멸자



```
C:\Windows\system32\cmd.exe
test.maxCount = 300
test.ref = 400
계속하려면 아무 키나 누르십시오 . . .
```



생성자와 소멸자



소멸자는 객체가 소멸할 때 자동적으로 호출되는 함수!

소멸자의 이름은 생성자의 이름 앞에 ~를 붙인 형태!

예를 들어 Point 클래스의 소멸자는 ~Point()

소멸자의 주요 용도는 객체가 사용한 자원의 '정리'!

생성자와 소멸자



```
class Test {  
public:  
    int* arr;  
  
    Test(int arraySize);  
    ~Test();  
};
```

생성자와 소멸자



```
Test::Test(int arraySize) {  
    arr = new int[arraySize];  
}  
    
```

```
Test::~Test() {  
    delete[] arr;  
    arr = NULL;  
}
```

생성자와 소멸자



```
void main() {
    int size;
    std::cout << "몇 개의 정수를 입력? : ";
    std::cin >> size;

    Test test(size);

    for (int i = 0; i < size; i++)
        std::cin >> test.arr[i];

    for (int i = size - 1; i >= 0; i--)
        std::cout << test.arr[i] << " ";
}
```

생성자와 소멸자



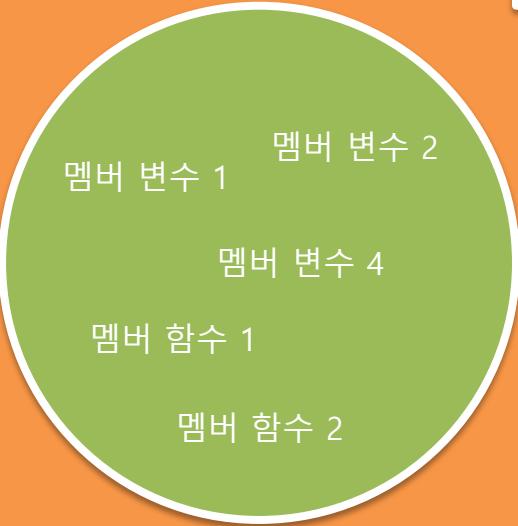
A screenshot of a Windows command-line interface (cmd.exe) window. The window title bar reads "C:\Windows\system32\cmd.exe". Inside the window, the following interaction is visible:

```
몇 개의 정수를 입력? : 5
1 3 5 7 9
9 7 5 3 1
계속하려면 아무 키나 누르십시오 . . .
```

The window has a standard Windows-style title bar with minimize, maximize, and close buttons. The main area is a black terminal window with white text.

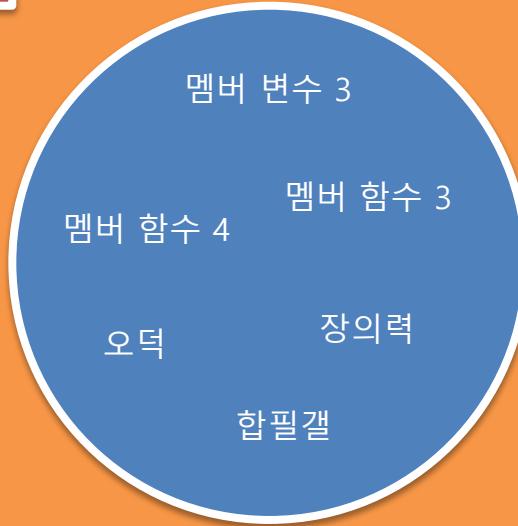
클래스에서 말하는 접근 제어는
어떤 멤버를 외부에 보이게 할 것인지 지정하는 작업!

클래스



멤버 변수 1
멤버 변수 2
멤버 변수 4
멤버 함수 1
멤버 함수 2

외부에 공개할 멤버들



멤버 변수 3
멤버 함수 3
멤버 함수 4
오덕
장의력
합필갤

외부에 숨길 멤버들

접근 권한과 관련한 키워드

public : 외부에서의 접근을 허용한다.

protected, private : 외부에서 접근할 수 없다.

protected와 private의 차이점은
상속을 배우고 나서...

```
class AccessControl {  
public:  
    char publicData;  
    void publicFunc() { }  
protected:  
    int protectedData;  
    void protectedFunc() { }  
private:  
    float privateData;  
    void privateFunc() { }  
};
```

```
int main() {
    AccessControl ac;

    ac.publicData = 'A';
    ac.publicFunc();
    ac.protectedData = 100;
    ac.protectedFunc();
    ac.privateData = 4.5f;
    ac.privateFunc();

    return 0;
}
```

접근 제어



오류 목록				
	설명	파일	줄	열
	프로젝트			
✖ 1	error C2248: 'AccessControl::protectedData' : protected 멤버('AccessControl' 클래스에서 선언)에 액세스할 수 없습니다.	test.cpp	20	1
✖ 2	error C2248: 'AccessControl::protectedFunc' : protected 멤버('AccessControl' 클래스에서 선언)에 액세스할 수 없습니다.	test.cpp	21	1
✖ 3	error C2248: 'AccessControl::privateData' : private 멤버('AccessControl' 클래스에서 선언)에 액세스할 수 없습니다.	test.cpp	22	1
✖ 4	error C2248: 'AccessControl::privateFunc' : private 멤버('AccessControl' 클래스에서 선언)에 액세스할 수 없습니다.	test.cpp	23	1
✖ 5	IntelliSense: 멤버 "AccessControl::protectedData" (선언됨 줄 8)에 액세스할 수 없습니다.	test.cpp	20	7
✖ 6	IntelliSense: 함수 "AccessControl::protectedFunc" (선언됨 줄 9)에 액세스할 수 없습니다.	test.cpp	21	7
✖ 7	IntelliSense: 멤버 "AccessControl::privateData" (선언됨 줄 11)에 액세스할 수 없습니다.	test.cpp	22	7
✖ 8	IntelliSense: 함수 "AccessControl::privateFunc" (선언됨 줄 12)에 액세스할 수 없습니다.	test.cpp	23	7



main() 함수는 '외부'가 된다는 점!

클래스에게 있어서 '내부'는 멤버 함수 뿐!

다른 모든 함수들은 '외부'

근데 접근 권한은요...

외부에서 접근할 수 있느냐를 설정한 것이거든요...

클래스의 내부인 멤버 함수에서는

접근 권한에 상관 없이 모든 멤버에 접근할 수 있지요!

```
class AccessControl {  
public:  
    char publicData;  
    void publicFunc() { }  
protected:  
    int protectedData;  
    void protectedFunc() { }  
private:  
    float privateData;  
    void privateFunc() { }  
  
public:  
    void AccessAllMembers();  
};
```

```
void AccessControl::AccessAllMembers() {  
    publicData = 'A';  
    publicFunc();  
    protectedData = 100;  
    protectedFunc();  
    privateData = 4.5f;  
    privateFunc();  
}
```

private로 지정한 멤버에 접근할 수는 없지만
public으로 지정한 멤버 함수를 통해서 간접적으로 접근 가능!

클래스에서 모든 멤버 변수는 protected 혹은 private로!
(약속된 부분[인터페이스]에 멤버 변수를 놓지 말라는 뜻)



멤버 변수가 공개되면...

아무 때나, 아무 곳에서나 멤버 변수의 값 변경 가능!

집에서 TV를 시청하고 있는데
골목을 지나가던 사람이 리모콘을 사용해서
다른 채널을 변경...

근데 하필이면 변경한 채널이 19금 방송이라면!?

접근 제어



저도 이런짓은 맨정신으로 못해요.

따라서 직접 멤버 변수를 노출하는 대신에...

접근자를 사용해서 간접적으로 멤버 변수를 노출하는
방법을 사용할 필요가 있다!

접근자는 단순히 public 접근 권한을 가진 멤버 함수로
외부에서 멤버 변수에 접근하는 것을 도와주는
혹은 감시하는 역할을 한다

```
class Point {  
public:  
    // 접근자  
    void SetX(int value) { x = value; }  
    void SetY(int value) { y = value; }  
    int GetX() { return x; }  
    int GetY() { return y; }  
  
private:  
    int x, y;  
};
```

```
int main() {
    Point pt;

    // pt의 x, y 좌표를 대입
    pt.SetX(100);
    pt.SetY(100);

    // pt의 x, y 좌표를 각각 출력
    std::cout << "pt.x = " << pt.GetX() << "\n";
    std::cout << "pt.y = " << pt.GetY() << "\n";

    return 0;
}
```



```
C:\Windows\system32\cmd.exe
pt.x = 100
pt.y = 100
계속하려면 아무 키나 누르십시오 . . .
```

멤버 변수 x, y의 값을 읽고 쓰게 해주는

멤버 함수들을 접근자라고 한다

(읽고 : Get, 쓰고 : Set)

정적 멤버(Static Members)는 모든 객체가 공유하는 멤버!

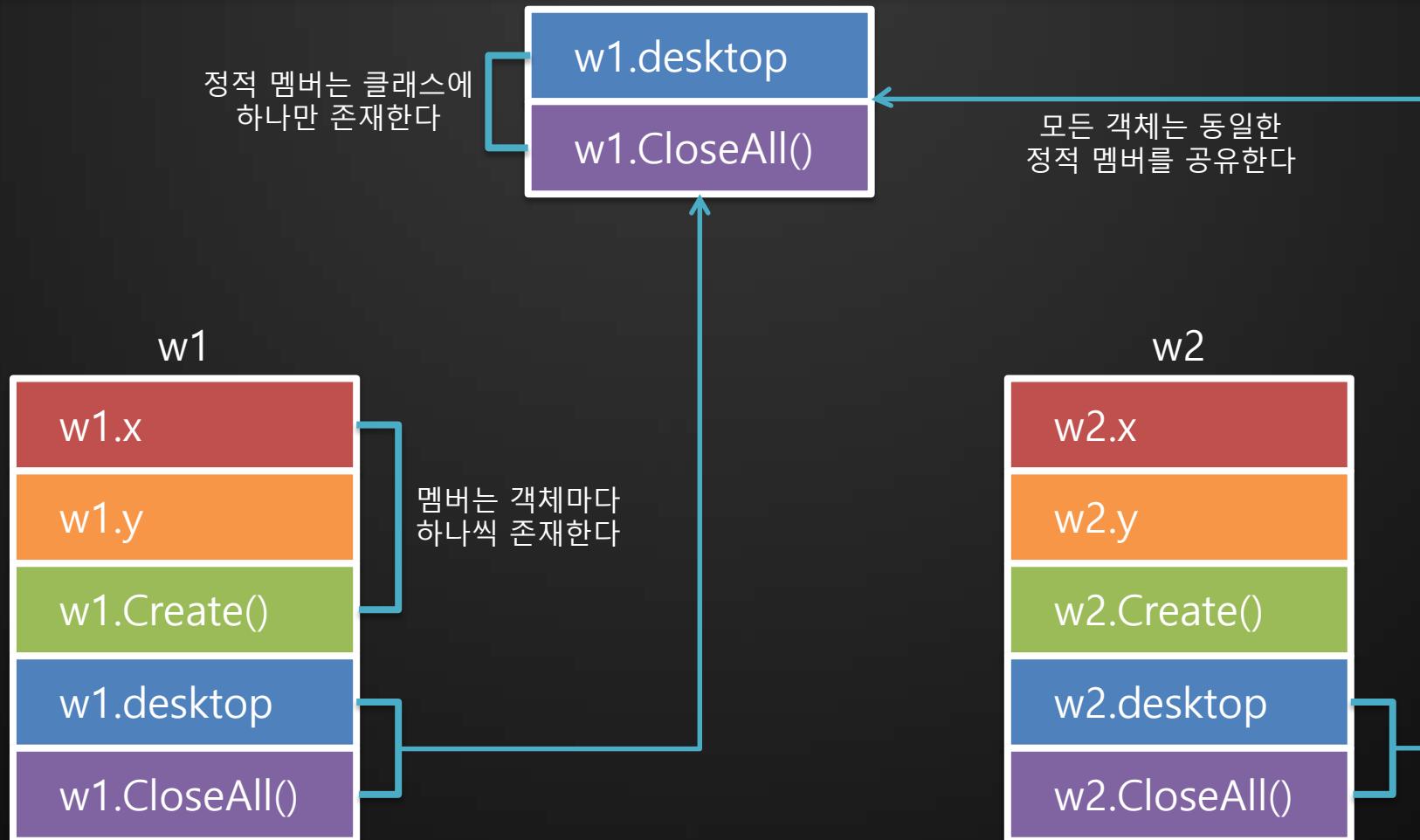
멤버 변수와 멤버 함수들은 객체마다 하나씩 존재
또한 객체는 저마다 자신의 멤버를 사용...

하지만 정적 멤버 변수와 정적 멤버 함수는
클래스에 오직 하나만 존재!

이렇게 생긴 클래스가 있다면...

```
class Window {  
    int x, y;  
    void Create();  
  
    static char desktop[20];  
    static void CloseAll();  
};
```

정적 멤버



w1.desktop == w2.desktop == Window::desktop
(모든 객체의 desktop은 동일한 변수 Window::desktop을 의미한다)

정적 멤버



Q : 정적 멤버 별로 안 어렵네요. 예제나 줘 보시죠.



정적 멤버



```
class Number {  
public:  
    Number(int num);  
    ~Number();  
    // 정적 멤버  
    static int number_count;  
    static void Print();  
  
private:  
    int number;  
};  
  
// 정적 멤버 변수  
int Number::number_count = 0;
```

정적 멤버



```
void Number::Print() {  
    std::cout << "Number 객체 수 = "  
        << number_count << endl;  
}
```

```
Number::Number(int num) : number(num) {  
    // 숫자 객체의 수 증가  
    number_count++;  
}  
    
```

```
Number::~Number() {  
    // 숫자 객체의 수 감소  
    number_count--;  
}
```

정적 멤버



```
void main() {
    Number::Print();

    // 객체 생성
    Number one(1);

    Number::Print();

    Number two(2);
    Number three(3);

    Number::Print();
}
```

정적 멤버

A screenshot of a Windows command-line interface (cmd.exe) window. The window title is "C:\Windows\system32\cmd.exe". Inside the window, the following text is displayed:

```
Number 객체 수 = 0
Number 객체 수 = 1
Number 객체 수 = 3
계속하려면 아무 키나 누르십시오 . . .
```

The text shows three separate outputs of the static variable "객체 수" (object count) from three different instances of the class, demonstrating that static members belong to the class itself rather than individual objects.

정적 멤버는 객체의 소유가 아니라 클래스의 소유!

정적 멤버 함수에서는 일반 멤버에 접근할 수 없다

(number가 어느 클래스의 멤버 변수인지 알 수 없기 때문)

```
void Number::Print() {  
    number = 100; // 실패  
  
    std::cout << "Number 객체 수 = "  
        << number_count << "\n";  
}
```

멤버 함수에서 다룰 내용?

인라인 함수

Const 함수

멤버 함수의 오버로딩



인라인 함수(Inline Function)는
새로운 종류의 함수가 아니고 함수의 속성!
(멤버가 아닌 함수도 인라인이 될 수 있지만
여기서는 인라인 멤버 함수를 기준으로 설명)

인라인 함수의 기본 개념은 함수를 호출하는 대신에
함수의 내용을 그대로 옮겨 놓는 것!

멤버 함수



```
// 인라인 함수
inline void Func() {
    std::cout << "내가 인라인이라니!\n";
}

int main() {
    // 인라인 함수를 호출
    Func();

    return 0;
}
```

Func() 함수가 인라인인 경우에 컴퓨터는
main() 함수를 다음과 같은 소스 코드로 인식

```
int main() {  
    // 인라인 함수를 호출  
    std::cout << "내가 인라인이라니!\n";  
  
    return 0;  
}
```

Q : 함수의 내용을 그대로 옮기는 게 어디에 좋나요?

A : 일반적인 함수라면...

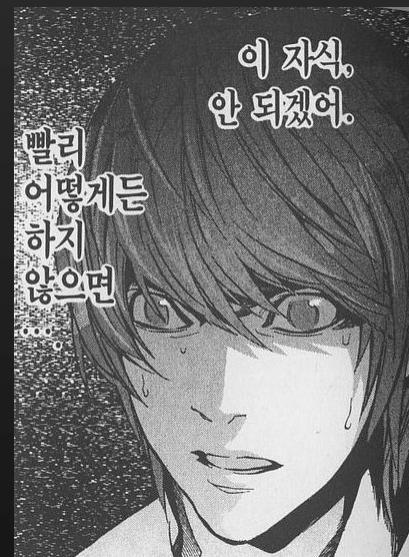
실행의 흐름을 옮기는 작업, 인자를 복사하는 과정에서
많은 부하(Overhead)가 발생하지만
인라인 함수를 사용하면 이런 작업들이 불필요!

멤버 함수



Q : 올ㅋ! 그러면 앞으로 모든 함수에 inline을 붙여야겠네요?

A : 천만에 말씀! 부하를 줄일 수는 있지만 다른 단점이 존재
(무엇이 단점일까? 맞추면 보상 제공ㅋ)



우리가 인라인 함수로 만들고 싶다고 해서
항상 인라인 함수가 되는 것은 아니다

우리는 인라인 함수를 만들고 싶다고 요청할 뿐,
정말로 인라인 함수로 만들지는 컴퓨터가 결정!



멤버 함수를 인라인으로 만드는 방법?

클래스의 내부에 정의한 멤버 함수들은
모두 자동으로 인라인 함수!

클래스의 외부에 정의한 멤버 함수는
함수의 정의 앞에 `inline` 키워드를 추가!

멤버 함수



일단 클래스의 내부에 정의한 멤버 함수들부터!

멤버 함수



```
class Number {  
public:  
    void SetNumber(int value) {  
        if (value < 0)            number = 0;  
        else if (value > 100)   number = 100;  
        else                      number = value;  
    }  
  
private:  
    int number;  
};
```

멤버 함수



우리가 보기엔 이렇겠지만...

```
Number num;  
num.SetNumber(84);  
num.SetNumber(127);
```

컴퓨터가 보기엔...

```
Number num;
```

```
// num.SetNumber(84);
int temp1 = 84;
if (value < 0)          num.number = 0;
else if (value > 100)    num.number = 100;
else                      num.number = temp1;
// num.SetNumber(127);
int temp2 = 127;
if (value < 0)          num.number = 0;
else if (value > 100)    num.number = 100;
else                      num.number = temp2;
```

멤버 함수



이번에는 클래스의 외부에서 정의한 멤버 함수를 인라인으로!

멤버 함수



```
class Number {  
public:  
    void SetNumber(int value);  
}  
  
private:  
    int number;  
};  
  
inline void Number::SetNumber(int value) {  
    if (value < 0)            number = 0;  
    else if (value > 100)     number = 100;  
    else                      number = value;  
}
```

함수에도 Const가 있다?

변수의 경우라면...

내용을 변경하지 못하게 하려고 Const 속성을 부여

함수에 Const 속성을 부여하는 것은 어떤 의미를 가지고 있을까?

Const 함수의 자격 요건

멤버 변수의 값을 변경하지 않는 멤버 함수!



멤버 함수를 Const로 만드는 것의 의미

1. 다른 개발자가 “아, 이 함수는 멤버 변수의 값을 변경하지 않는구나”라고 생각하게 만든다
2. 실수로 멤버 변수의 값을 바꾸려고 하면, 컴퓨터가 오류 메시지를 통해서 알려준다
3. Const 객체를 사용해서 이 함수를 호출할 수 있다

Const 객체라는 것은 다음과 같이 정의한 객체

```
const Point pt(100, 100);
```

기본 타입의 변수가 Const라면...

그 변수의 값을 바꿀 수가 없다

객체가 Const인 경우에는...

해당 객체의 멤버 변수의 값을 바꿀 수가 없다

Const 객체의 멤버 변수에 접근해서 값을 바꾸려고 하면
오류 메시지가 발생!

멤버 변수의 값을 바꾸는 멤버 함수를 호출하는 경우에도
오류 메시지가 발생!

멤버 변수의 값을 바꾸지 않는
Const 멤버 함수를 호출하는 것은 허용!

멤버 함수



```
class CellPhone {  
public:  
    int bell_mode;  
  
    void Call(int quick_num);  
    void ShowRecentCall() const;  
};
```

멤버 함수



```
void main() {
    const CellPhone myPhone;

    myPhone.bell_mode = 3; // 오류
    myPhone.Call(1);      // 오류

    myPhone.ShowRecentCall();
}
```

멤버 함수



Q : 멤버 함수의 오버로딩은
일반 함수의 오버로딩과 뭐가 달라요?
A : 다른 거 없는데요?



멤버 함수



```
class Number {  
public:  
    void Offset(int num_offset);  
    void Offset(const Number& num);  
    void SetNumber(int value) {  
        if (value < 0)            number = 0;  
        else if (value > 100)    number = 100;  
        else                      number = value;  
    }  
  
private:  
    int number;  
};
```

멤버 함수



```
void Number::Offset(int num_offset) {  
    SetNumber(number + num_offset);  
}
```

```
void Number::Offset(const Number& num) {  
    Offset(num.number);  
}
```

객체의 배열



일단 생성자를 고려하지 않고 객체의 배열을 생성

```
class Number {  
public:  
    Number() { number = 0; }  
    Number(int num) : number(num) { }  
    void Print() {  
        std::cout << number << "n";  
    }  
  
private:  
    int number;  
};
```

객체의 배열



```
int main() {  
    Number num[3];  
  
    for (int i = 0; i < 3; i++) {  
        num[i].Print();  
    }  
  
    return 0;  
}
```

객체의 배열



객체의 배열을 정의할 때 각 객체들은
디폴트 생성자로 초기화된다

객체의 배열



다른 생성자를 호출하게 만들고 싶다면 배열의 초기화를 사용!

```
void main() {  
    Number num[3] = {  
        Number(1),  
        Number(2),  
        Number(3)  
    };
```

```
for (int i = 0; i < 3; i++) {  
    num[i].Print();  
}
```

객체의 배열



A screenshot of a Windows command prompt window titled "cmd C:\Windows\system32\cmd.exe". The window contains the following text:

```
1  
2  
3  
계속하려면 아무 키나 누르십시오 . . .
```



객체의 동적인 생성



```
class Number {  
public:  
    Number() { number = 0; }  
    Number(int num) : number(num) { }  
    Number(const Number& num) {  
        number = num.number;  
    }  
    void Print() {  
        std::cout << number << "n";  
    }  
  
private:  
    int number;  
};
```

객체의 동적인 생성



```
void main() {  
    Number num(72);  
    Number* num1 = new Number();  
    Number* num2 = new Number(99);  
    Number* num3 = new Number(num);  
  
    num1->Print(); // (*p1).Print();와 동일  
    num2->Print(); // (*p2).Print();와 동일  
    num3->Print(); // (*p3).Print();와 동일  
  
    delete num1;  
    delete num2;  
    delete num3;  
    num1 = num2 = num3 = 0;  
}
```

객체의 동적인 생성



Q : 객체를 동적으로 생성할 때
생성자와 소멸자는 언제 호출되나요?

A : 정적으로 생성할 때랑 똑같아요...

객체의 동적인 생성



는 함정!!!



생성자는 **new** 연산자를 사용해서 객체를 생성할 때 호출!
소멸자는 **delete** 연산자를 사용해서 해제할 때 호출!

```
void main() {  
    Number num(72); // 생성자 : 정적인 생성  
  
    // 생성자 : 동적인 생성  
    Number* num1 = new Number(99);  
  
    delete num1; // 소멸자 : 동적인 생성  
} // : 소멸자 : 정적인 생성
```

객체의 동적인 생성



정리하면...

정적인 생성

생성자의 호출

객체를 정의할 때

동적인 생성

new 연산자로 할당할 때

소멸자의 호출

객체를 정의한 함수가 끝날 때

delete 연산자로 해제할 때

클래스에 넣을 수 있는 다른 것들



열거체부터 알아봐요!

클래스의 정의 안에 열거체를 넣는 것은
클래스 밖에서 정의하는 것과 다르지 않다

다만...

클래스에 넣을 수 있는 다른 것들



열거체의 심볼을 클래스의 외부에서 참조하고 싶다면
영역 지정 연산자(:)를 사용!



클래스에 넣을 수 있는 다른 것들



```
class Count {  
public:  
    enum { MIN = 0, THREE = 3, EIGHT = 8, MAX = 10};  
  
    void Print() {  
        std::cout << num << endl;  
    }  
    void SetCount(int count);  
  
private:  
    int num;  
};
```

클래스에 넣을 수 있는 다른 것들



```
inline void Count::SetCount(int count) {  
    if (count < MIN)          num = MIN;  
    else if (count > MAX)    num = MAX;  
    else                      num = count;  
}  
  
void main() {  
    Count cnt;  
  
    cnt.SetCount(Count::EIGHT);  
}
```

클래스에 넣을 수 있는 다른 것들



typedef도 다르지 않아요!



클래스에 넣을 수 있는 다른 것들



```
class Count {  
public:  
    enum { MIN = 0, THREE = 3, EIGHT = 8, MAX = 10};  
    typedef int Count_Type;  
  
    void Print() {  
        std::cout << num << endl;  
    }  
  
    void SetCount(Count_Type count);  
  
private:  
    Count_Type num;  
};
```

클래스에 넣을 수 있는 다른 것들



```
inline void Count::SetCount(Count_Type count) {  
    if (count < MIN)          num = MIN;  
    else if (count > MAX)    num = MAX;  
    else                      num = count;  
}  
  
void main() {  
    Count cnt;  
  
    cnt.SetCount(Count::EIGHT);  
}
```

This 포인터



멤버 함수에는 This 포인터가 숨어 있지요!
근데 이게 뭐죠?



This 포인터는 자기 자신을 가리킨다!

예를 들어 pt라는 객체를 사용해서 멤버 함수를 호출했다면
This 포인터는 pt 객체를 가리키게 된다



This 포인터



```
class WhoAmI {  
public:  
    int id;  
  
    WhoAmI(int id_arg);  
    void ShowYourSelf() const;  
};
```

This 포인터



```
WhoAmI::WhoAmI(int id_arg) {  
    id = id_arg;  
}
```

```
void WhoAmI::ShowYourSelf() const {  
    std::cout << "(ID = " << id  
        << ", this = " << this << ")\\n";  
}
```

This 포인터



```
void main() {
    WhoAmI obj1(1), obj2(2), obj3(3);

    obj1.ShowYourSelf();
    obj2.ShowYourSelf();
    obj3.ShowYourSelf();

    std::cout << "&obj1 = " << &obj1 << "\n";
    std::cout << "&obj2 = " << &obj2 << "\n";
    std::cout << "&obj3 = " << &obj3 << "\n";
}
```

This 포인터



```
C:\Windows\system32\cmd.exe
<ID = 1, this = 0017FB0>
<ID = 2, this = 0017FBC4>
<ID = 3, this = 0017FBB8>
&obj1 = 0017FB0
&obj2 = 0017FBC4
&obj3 = 0017FBB8
계속하려면 아무 키나 누르십시오 . . .
```

가상의 함수
obj1.ShowYourSelf(); → WhoAmI* const this = &obj1;

↓

```
void WhoAmI::ShowYourSelf() const {
    std::cout << "(ID = " << id
        << ", this = " << this << ")\\n";
}
```

This 포인터는 멤버 함수에만 있는 것!

This 포인터는 멤버 함수를 소유한 객체를 가리키는 것이 임무

그런데...

정적 멤버 함수는 객체의 소유가 아니기 때문에 This 포인터 없음!
(멤버가 아닌 함수에 This 포인터가 없는 것도 같은 이유)

This 포인터



```
class Test {  
public:  
    static void TestThis();  
};  
  
void Test::TestThis() {  
    std::cout << "this = " << this << endl;  
}
```

This 포인터



오류 목록				
	설명	파일	줄	열
프로젝트				
Test	error C2355: 'this' : 비정적 멤버 함수 안에서만 참조할 수 있습니다.	test.cpp	9	1
Test	IntelliSense: 'this'는 비정적 멤버 함수 내에서만 사용할 수 있습니다.	test.cpp	9	30



This 포인터



멤버 함수 안에서 정의한 변수나 매개변수의 이름이
멤버 변수와 동일한 경우에는 멤버 변수에 접근 불가능

하지만...

this 포인터를 사용하면 이런 상황도 멤버 변수에 접근 가능!

This 포인터



```
class Test {  
public:  
    Test(int id);  
  
private:  
    int id;  
};
```

```
Test::Test(int id) {  
    this->id = id;  
}  
↓  
멤버 변수 id를 의미  
↓  
매개 변수 id를 의미
```

8주차 레포트



- 얕은 복사(Shallow Copy)와 깊은 복사(Deep Copy)에 대해서...
- 생성자, 소멸자의 접근 권한을 private로 사용할 때는 언제일까?
- This 포인터의 타입은 무엇일까?
- 정말로 멤버 함수는 객체마다 있는 것일까?

이에 대한 자신의 생각을 A4 종이 1장 이내로 작성하여 다음주에 제출.
(가장 논리적인 답안을 한 학생 1명에게 보상 제공)

8주차 숙제



- 당구공 클래스

- class Ball {
public:
 int _x, _y; // 공의 위치
 int _radius; // 공의 반지름

- Ball(int x, int y); // 기본 생성자

- Ball(const Ball& b); // 복사 생성자

- };

- 기본 생성자와 복사 생성자를 구현

- 당구공 클래스에 접근 권한 설정

- 외부에서 접근을 막아야 하는 멤버와 접근을 허용해야 하는 멤버 골라서 알맞은 접근 권한을 설정

8주차 숙제



- 당구공 클래스에 접근자 추가
 - 당구공의 크기가 800×400 인데 당구공의 위치가 (800, 400) 밖으로 나가는 경우 발생
 - 당구공 클래스(Ball)에 접근자를 추가해서 _x, _y의 값이 (800, 400)을 넘어설 수 없도록 수정
- 당구공 클래스 최적화
 - 게임에 사용하는 당구공의 크기는 모두 같음
 - _radius를 정적 멤버 변수로 고치기
- 당구공 4개 생성하기
 - 나란히 놓여져 있는 당구공 객체 4개를 배열에 보관하는 코드를 작성

8주차 숙제



- Color 클래스 스스로 만들기
 - 색상의 정보를 보관하고 RGB 요소를 개별적으로 변경하는 것도 가능
 - 2개의 파일이 주어짐
 - Color 클래스의 정의를 포함하고 있는 Color.h 파일
 - Color 클래스의 사용하는 방법을 보여주는 Example.cpp 파일
 - 두 파일을 참조해서 Color.cpp 파일을 완성하는 것이 여러분의 몫
(두 파일은 C++ 튜터 클럽에 업로드 예정)

8주차 숙제



9주차 수업 안내



- 상속과 포함
 - 포함
 - 왜 포함을 사용해야 할까
 - 객체를 멤버로 갖는 클래스
 - 상속
 - 문서 저장 클래스
 - HTML 문서 저장 클래스
 - 클래스간의 형변환
 - 접근 제어
 - 다중 상속



THANK YOU!