
CODINGON
×
posco
K-Digital Training
스마트 팩토리 3기

구조체

구조체 ??

- 프로그래밍을 하다 보면 변수 하나로 표현하기 힘든 것이 있음.
  - ex) 학생을 표현 하려한다면 이름, 나이, 학교, 학년, 학번, 전공 등등의 다양한 특징에 대한 변수가 필요함.
  - ex) 위치를 표현하려면, x좌표, y좌표 등에 대한 변수가 필요함.

구조체

```
struct Position {
    int x = 0;
    int y = 0;
};

Position p;

p.x = 3;
p.y = 5;

p = {3, 5}
```

구조체

```
struct Position {
    int x = 0;
    int y = 0;
};

Position p;
Position *pp = &p;

pp->x = 3;
pp->y = 5;

int a = pp->x + pp->y;
```

클래스

객체지향 프로그래밍

필요한 데이터와 코드를 묶어 하나의 객체로 만들고
이 객체들 간에 상호작용을 하도록 프로그램을 만드는 방식

(실제 세계를 모델링하여 소프트웨어를 개발하는 방법)

절차지향 프로그래밍

순차적인 처리가 중요시 되며 프로그램 전체가 유기적으로 연결되도록 만드는 프로그래밍 기법

객체지향 프로그래밍

- 장점
  - 코드 재사용에 용이
  - 유지보수 용이

- 단점
  - 처리속도가 느림 -> but, 사람이 인지할 정도의 속도 X
  - 설계가 복잡함

절차지향 프로그래밍

- 장점
  - 컴퓨터의 처리구조와 유사해 실행속도가 빠름

- 단점
  - 유지보수가 어려움
  - 실행 순서가 정해져 있으므로 코드의 순서가 바뀌면 동일한 결과를 보장하기 어려움

객체 ??

- 실생활에서 우리가 인식할 수 있는 사물

- 객체(object)는 상태와 동작을 가지고 있다.
  - 객체의 상태는 객체의 특징 값(속성)
  - 객체의 동작은 객체가 취할 수 있는 동작

클래스 ??

- 객체 지향 프로그래밍을 실현하기 위해 나온 개념!
  - ex) 사람이라는 객체를 표현 하려한다면, 이름, 성별, 나이, 상황에 따른 행동 등을 정의할 수 있는 것들이 필요함.

클래스 예제

```
#include <iostream>

using namespace std;

class student {
private:
    char * name;
    int age;
public:
    void ShowInfo();
    void SetInfo(char * _name, int _age);
};

void student::ShowInfo()
{
    cout << "이름: " << name << ", 나이: " << age << endl;
}
```

```
int main()
{
    student stu;

    stu.SetInfo("홍길동", 24);
    stu.ShowInfo();

    return 0;
}
```

클래스

```
class Position {
    int x = 0; // 필드
    int y = 0;

    public : // 접근 제어자
        Position() { } // 생성자

        void printXY() { // 메소드
                std::cout << x << " " << y;
        }
};
```

```
int main() {
        Position p; // p 객체 생성
}
```

클래스

아무것도 선언하지 않아도 컴파일러가 자동으로 생성자, 소멸자를 만들어 줌

```
class Position {
        public :
}


int main() {
        Position p; // p 객체 생성
}
```

클래스

- this
  - 클래스내에서 자기 자신을 가리키는 포인터

```
class Position {
    int x = 0;
    void setX(int x) {
        this->x = x;
    }
};
```

this 없으면 밖의 x와 안의 x는 다름

클래스의 구조

- 필드(변수) : 클래스 내에서 값을 저장하는 변수
- 메소드 : 클래스 내에 선언된 함수
- 생성자 : 객체가 생성될 때 자동으로 호출되는 메소드
- 생성자 : 객체가 소멸될 때 자동으로 호출되는 메소드

객체 예시

고양이를 표현한다고 가정

- 속성(property)
  - 이름 : 나비
  - 나이 : 1살
  - 종 : 페르시안
- 행동(method)
  - mew() : 울다
  - eat() : 먹는다

생성자

생성자

```
class Box {
public:   밖에서 접근 가능
    // Default constructor
    Box() {}

    // Initialize a Box with equal dimensions (i.e. a cube)
    explicit Box(int i) : m_width(i), m_length(i), m_height(i) // member init list
    {}

    // Initialize a Box with custom dimensions
    Box(int width, int length, int height)
        : m_width(width), m_length(length), m_height(height)
    {}

    int Volume() { return m_width * m_length * m_height; }

private:  밖에서 접근 불가능
    // Will have value of 0 when default constructor is called.
    // If we didn't zero-init here, default constructor would
    // leave them uninitialized with garbage values.
    int m_width{ 0 };
    int m_length{ 0 };
    int m_height{ 0 };
};
```

생성자

```
int main()
{
    Box b; // Calls Box()

    // Using uniform initialization (preferred):
    Box b2 {5}; // Calls Box(int)
    Box b3 {5, 8, 12}; // Calls Box(int, int, int)

    // Using function-style notation:
    Box b4(2, 4, 6); // Calls Box(int, int, int)
}
```

대괄호 == 괄호

기본 생성자

- 일반적으로 매개 변수가 없지만 기본값이 있는 매개변수를 가질 수 있다.

```
class Box {
public:
    Box() { /*perform any required default initialization steps*/}

    // All params have default values
    Box (int w = 1, int l = 1, int h = 1): m_width(w), m_height(h), m_length(l){}
...
}
```

기본 생성자

- 암시적 기본 생성자를 사용 하는 경우, 클래스 정의에서 멤버를 초기화 해야 함

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

class Box {
public:
    int Volume() {return m_width * m_height * m_length;}
private:
    int m_width { 0 };
    int m_height { 0 };
    int m_length { 0 };
};

int main() {
    Box box1; // Invoke compiler-generated constructor
    cout << "box1.Volume: " << box1.Volume() << endl; // Outputs 0
}
```

기본 생성자

- 기본이 아닌 생성자가 선언된 경우 컴파일러는 기본 생성자를 제공하지 않음

```
class Box {
public:
    Box(int width, int length, int height)
        : m_width(width), m_length(length), m_height(height){}
private:
    int m_width;
    int m_length;
    int m_height;

};

int main(){

    Box box1(1, 2, 3);
    Box box2{ 2, 3, 4 };
    Box box3; // C2512: no appropriate default constructor available
}
```

생성자를 정의한 경우 밑에서도 사용을 해줘야 한다.

생성자를 정의하지 않았으면 이렇게 사용 가능

기본 생성자

- Box 개체의 배열 초기화

```
Box boxes[3]{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
```

복사 생성자

- 동일한 형식의 개체에서 멤버 값을 복사하여 개체를 초기화

```
Box(Box& other); // Avoid if possible--allows modification of other.
Box(const Box& other);

// Additional parameters OK if they have default values
Box(Box& other, int i = 42, string label = "Box");
```

```
Box(const Box& other) {
    m_width = other.m_width;
    m_height = other.m_height;
    m_length = other.m_length;
}
```

복사 생성자

- 동일한 형식의 개체에서 멤버 값을 복사하여 개체를 초기화

```
Box(Box& other); // Avoid if possible--allows modification of other.
Box(const Box& other); 값을 바꾸지 않고 참조하여 Box에 복사 만 하기 위하여 const 사용

// Additional parameters OK if they have default values
Box(Box& other, int i = 42, string label = "Box");
```

```
Box(const Box& other) {
    m_width = other.m_width;
    m_height = other.m_height;
    m_length = other.m_length;
}
```

복사하는 이유 :
int funA (int a[1000], int b[1000]){
return a[99] + b[99];}
=> 2000개의 값을 복사해야 함

int funB (int &a[1000], int &b[1000]){
return a[99] + b[99];}
=> 2개의 주소 만 복사하면 됨

복사 생성자

- 선언과 동시에 사용해야 호출 됨

```
Box box2 = box1; // 호출됨
Box box3(box1); // 호출됨

Box box4;
box4 = box1; // 호출안됨
```

복사 생성자

- 선언과 동시에 사용해야 호출 됨

```
Box box2 = box1; // 호출됨
Box box3(box1); // 호출됨

Box box4;
box4 = box1; // 호출안됨
```

복사 생성자

- 명시적으로 선언하지 않아도 컴파일러가 자동으로 생성
- 하지만 pointer가 있는 클래스의 경우 복사된 클래스의 pointer가 복사하기 전 주소를 참조하므로 주의 필요

명시적 생성자

- 단일 매개 변수를 사용하는 생성자가 있거나 하나를 제외한 모든 매개 변수에서 기본값을 사용하는 경우 이 매개 변수 형식은 클래스 형식으로 암시적으로 변환할 수 있음

```
Box(int size): m_width(size), m_length(size), m_height(size){}

Box b = 42;
```

소멸자

소멸자 ??

- 객체가 소멸될 때 자동으로 실행되는 메소드

소멸자

```
class Person {
public:
    Person() { // 생성자
        cout << "생성자 입니다" << endl;
    }
    ~Person() { // 소멸자
        cout << "소멸자입니다" << endl;
    }
};
```

구조체 vs 클래스

구조체

```
struct Position {
// 사실 구조체 내에서도 변수만 선언할 수 있는 것은 아님.
    int x = 0;
    int y = 0;

    Position() { } // 생성자

    void printXY() { // 메소드
        std::cout << x << " " << y;
    }
};
```

구조체와 다른 점은...??

- 개념이 파생되게 된 계기가 다름
  - 구조체 : 하나의 변수만으로 표현하기 어려운 것들을 표현하기 위해
  - 클래스 : 객체지향 프로그래밍을 실현하기 위해

- 문법 상으론 거의 없음.

- 접근 제어자의 기본 값
  - 구조체의 접근 제어자의 기본 값은 public
  - 정보 은닉의 중요성에 따라 클래스의 접근 제어자의 기본 값은 private

접근 제어자

접근 제어자 ??

- 클래스의 멤버(변수, 메소드)들의 접근 권한을 지정
- public, protected, private 세 가지로 나뉨

접근 제어자

- **Public:** 어디서나 접근 가능
- **Private:** 해당 클래스 내에서만 접근 가능
- **Protected:** 해당 클래스 & 하위 클래스 내에서만 접근 가능
  ➡ 이 접근 제어자는 "상속" 이라는 개념을 배우고 자세히 알아볼 예정

getter & setter

getter, setter

- 클래스 외부에서 private 변수에 접근할 수 있도록 도와주는 메소드

- getter: 변수를 반환해주는 메소드
  - get + 변수 명 방식으로 함수 이름을 지정한다.

- setter: 변수에 값을 할당해주는 메소드
  - set + 변수 명 방식으로 함수 이름을 지정한다.

getter, setter 함수 예시

```
class Cat {
        string name;
        int age;

    public:
        Cat(string name, int age) {
            this->name = name;
            this->age = age;
        }

        // getter 함수
        string getName() {
            return name;
        }

        int getAge() {
            return age;
        }

        // setter 함수
        void setName(string name) {
            this->name = name;
        }

        void setAge(int age) {
            this->age = age;
        }
};
```