
CODINGON
×
posco
K-Digital Training
스마트 팩토리 4기

# 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;
```

# 실습 1. 구조체 사용해보기

(1) Rectangle 구조체 만들기
사각형의 가로 세로 길이를 저장하는 구조체

(2) 변수 width, height

(3) 구조체를 이용하여 변수를 생성하고, width와 height 값을 콘솔로 입력 받아서 할당

(4) width와 height 값을 이용해 넓이를 계산하여 출력

```
가로, 세로 길이를 입력하세요. 3.5 4
넓이는 : 14
```

# 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;
    }
};
```

# 클래스의 구조

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

# 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
}
```

# 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);

// 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

- 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

# 실습 1. 클래스 사용해보기

(1) Rectangle 클래스 만들기

(2) 필드(변수): width, height

(3) 생성자: width와 height 설정할 2개의 숫자를 매개변수로 받기.

(4) 메소드: width와 height를 이용하여 사각형의 넓이를 반환하는 area 메소드 만들기

(5) 객체 생성 시에 width와 height을 사용자에게 입력 받아 생성자로 넘겨주기

```
사각형의 가로와 세로 길이를 입력해주세요. (띄어쓰기로 구분) 4 5
넓이는 : 20
```

# 실습 1-1. 클래스 사용해보기

(1) 실습 1에서 만든 클래스를 다른 클래스 변수에 복사하기 (복사 생성자 이용해서)

(2) (1)에서 복사한 변수의 area()함수 호출해서 넓이 구하기

(3) 실습 1에서 만든 클래스를 기본생성자로 생성된 다른 클래스 변수에 복사하기 (= 할당 이용해서)

(4) (3)에서 복사한 변수의 area()함수 호출해서 넓이 구하기

# 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;
        }
};
```

# 실습 2. getter & setter 사용해보기

(1) 실습 1에서 만든 Rectangle 클래스에 getter와 setter 함수를 선언한다.

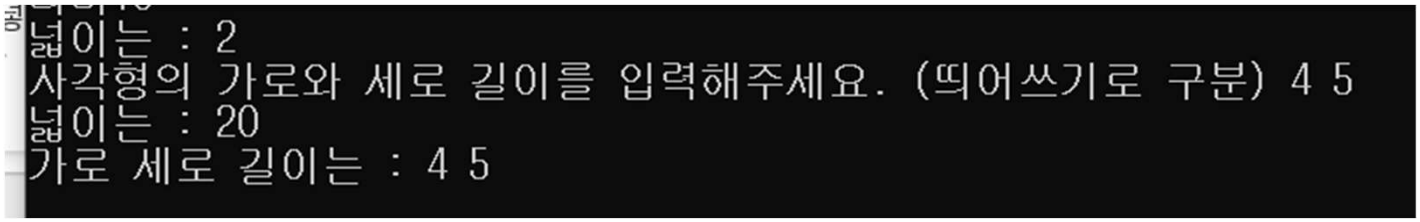
(2) 필드(변수): width, height

(3) 메소드: width와 height를 이용하여 사각형의 넓이를 반환하는 area 메소드 만들기

(4) Rectangle rect(1,2);  // 해당 코드 이용하여 클래스 선언 후 넓이 출력

(5) Setter 함수를 이용하여 사용자에게 입력 받은 width와 height를 Rectangle 클래스 필드에 저장한다.

(6) 넓이와 가로 세로 길이 출력



# 실습 3. 게임 캐릭터 생성하기

(1) Character라는 클래스 생성

(2) 필드(변수): name(이름), level(레벨), item_num(아이템 수)

(3) 생성자: 사용자에게 콘솔로 캐릭터 이름을 입력 받고, 캐릭터 생성하기 (이름은 입력 받은 값으로, 레벨, 아이템 수는 0으로 초기화)

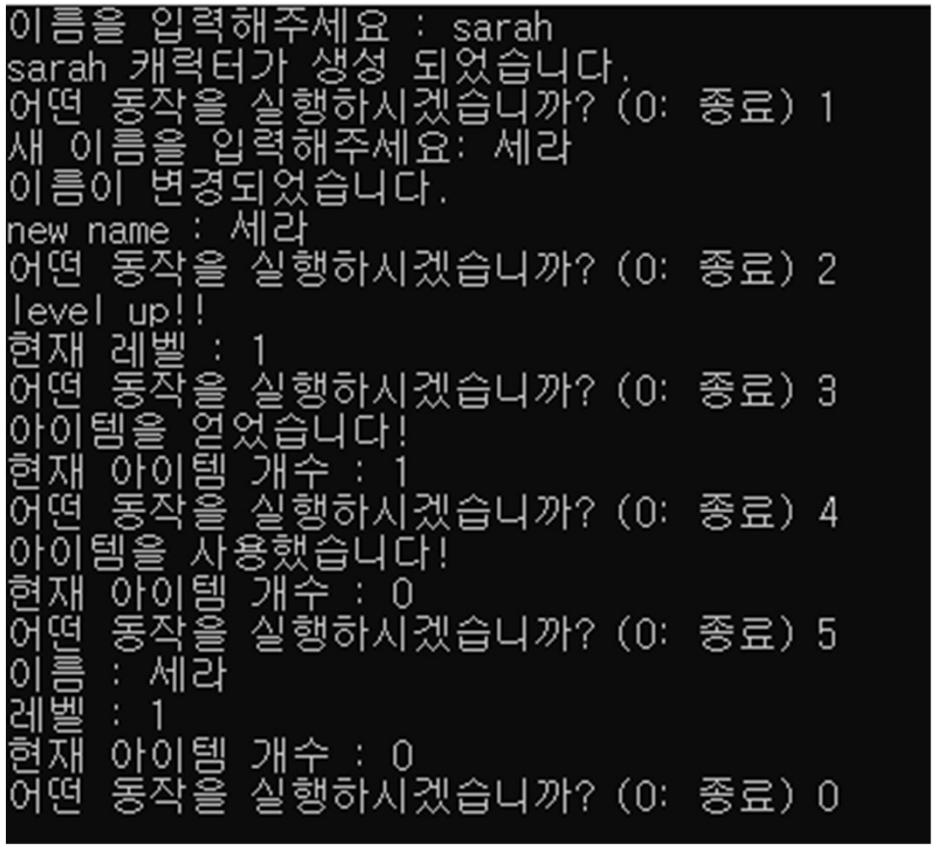
(4) 사용자에게 캐릭터를 어떻게 조작할지 입력하게 하기. (0을 입력할 때까지 입력한 번호에 해당하는 작업을 계속 하기)

(5) 위 기능은 모두 Character의 메소드로 만들어져 있어야 함

# 실습 3. 게임 캐릭터 생성하기

1을 입력하면, 이름 변경
2를 입력하면, level up (level이 1씩 올라가게)
3을 입력하면, item 줍기 (아이템 수가 1씩 증가)
4를 입력하면, item 사용 (아이템 수가 1씩 감소)
5를 입력하면, 이름, level, item을 콘솔에 출력하기
0을 입력하면, 게임 종료.

# 실습 3. 게임 캐릭터 생성하기



# [번외]실습 4. 게임 커스텀하기

실습 3에서 만든 게임을 직접 커스텀 해보세요!

예시 0. 발생 가능한 에러 처리하기 (ex. 아이템이 없으면 사용 못하게)

예시 1. 새로운 필드 및 메소드 추가 (ex. 체력, 공격하기)

예시 2. 객체끼리 상호작용 시키기 (ex. 이름 지목해서 공격)

예시 3. 저번 시간에 배운 파일 입출력을 적용해서 파일에 캐릭터들의 정보 저장하기