Structs and Classes

프로그래밍 입문(2)

Topics

- typedef
- 구조체(struct)
- Scope and Namespace
- **클래스(class)**

클래스(class)

- 객체지향 프로그래밍 지원을 위해 C++에서 새롭게 도입된 것.
- C++에서는 구조체와 매우 유사.
- 필드는 멤버 변수(member variable)이라고도 불림.
- public:에 주의.
  - public하게 사용할 수 있다는 의미.

```
class Person {
    public:
        int id; //member variable
        string name;
        int age;

}; //;를 잊지 말 것.
```

클래스(class)

- 클래스(Person)를 데이터형으로 선언된 변수는 **객체*(Object)*** 또는 Person의 **인스턴스*(Instance)***라고 부릅니다.

- 클래스를 선언하는 시점에는 클래스가 어떤 것인지 알려주기만 합니다.

- 실제 메모리가 할당되고 초기화되는 것은 변수가 선언되어 이 클래스의 인스턴스가 생성되는 시점입니다.

  - `Person p;`

```
class Person {

    public:
        int id; //member variable
        string name;
        int age;

}; //;를 잊지 말 것.


            Person p;
```

public vs. private

- private과 public은 클래스 내부에 접근하는 것을 제어할 수 있는 키워드.

- private - 외부에서 접근 불가.

- public - 외부에서 접근 가능.

- 오른쪽에서 멤버 변수들은 외부에서는 접근이 불가능함.

- e.g.) `Person p;`
  `p.name = "John";` → X

```
class Person {
    private:
        int id;
        string name;
        int age;

public:
    Person() {
        id = 0;
        name = "";
        age = 0;
    }
```

생성자
(Constructor)

- 생성자는 클래스의 변수가 선언될 때 객체를 인스턴스화하기 위해 실행됨.

- 생성자의 이름은 반드시 클래스의 이름과 일치해야 함.

- 기본 생성자는 아무런 매개변수가 없는 것.

- 실제 정의하지 않아도 암시적으로 생성되고 실행됨.

```
class Person {
    private:
        int id;
        string name;
        int age;

public:
    Person() {
        id = 0;
        name = "";
        age = 0;
    }
```

생성자와 매개변수

- 생성자는 기본적으로 반환값이 없는 함수와 유사.

- 매개변수를 받아 사용할 수 있음.

- 생성자 안에서는 private으로 선언된 멤버 변수도 접근이 가능하므로, 값을 대입하는 것이 가능.

```
public:
    Person() {
        id = 0;
        name = "";
        age = 0;
    }

    Person(string name, int age) {
        id = 0;
        this->name = name;
        this->age = age;
    }
```

this 포인터

- 두번째 생성자에서 this 포인터를 사용.
- 포인터이므로 참조를 위해 ->를 사용하였음.
- this 포인터는 현재 클래스의 인스턴스를 가리킴.
- Person 클래스의 변수 p1, p2를 선언하였다면 Person내부의 this는,
  - p1을 위해 실행될 때는 p1 인스턴스를,
  - p2를 위해 실행될 때는 p2 인스턴스를 가리키게 됨.

```
public:
    Person() {
        id = 0;
        name = "";
        age = 0;
    }

    Person(string name, int age) {
        id = 0;
        this->name = name;
        this->age = age;
    }

//잘못된 생성자.
Person(int id, string name, int age) {
    id = id;
    name = name;
    age = age;
}
```

멤버 변수 초기화

```
class Person {
    private:
        const int id;
        string name;
public:
    Person(int id, string name) : id(id) {
        this->name = name;
        cout << "1st Constructor!" << endl;
    }
```

- 멤버 변수가 const로 선언되면 처음 생성하면서 값을 초기화 해야 함.
    - 이전의 생성자에서 사용한 방식(id = 0)은 실제로는 변수 id를 선언 → 값을 대입한 것으로 취급됨 (`const int id = 0`이 아님).
- 따라서 실제로 선언과 동시에 초기화를 하기 위한 방법이 필요함.
- 생성자 옆에 콜론(:)을 붙이고, 그 뒤에 초기화를 위한 리스트를 작성할 수 있음 - 여러개는 쉼표(,)로 구분함.

기본값을 이용한 생성자

```
public:
    Person2(int id = 0, string name = "", int age = 1) {
        this->id = id;
        this->name = name;
        this->age = age;
    }
```

```
Person2 p1 (111);
Person2 p2 (123, "Another Person");
```

- 함수와 마찬가지로 매개변수에 기본값을 지정할 수 있음.
- Person2 클래스는 생성자가 하나 뿐이지만, 다양한 형태의 매개변수를 받아들여 객체를 생성할 수 있음.
- 함수 기본값과 마찬가지로 오른쪽부터 생략 가능함.

생성자 위임

```
Person(int id, string name) : id(id) {
    this->name = name;
    cout << "1st Constructor!" << endl;
}
Person(int id, string name, int age) : Person(id, name) {
    this->age = age;
    cout << "2nd Constructor!" << endl;
}

Person mcCree(123, "Jesse McCree", 37);
```

1

2

- 생성자 위임을 통해 복잡한 코드를 반복해서 사용하지 않고도 다양한 생성자 지원이 가능함.

- 매개변수 세 개를 이용해 생성자를 호출하면 우선 매개변수 두 개로 첫번째 생성자를 호출한 다음, 두번째 생성자의 코드가 실행됨.

멤버 함수
(Member Function)

- 클래스에 소속되어 정의된 함수.
- 멤버 변수와 마찬가지로 '.'연산자를 이용해 호출할 수 있음.
- 항상 클래스나 클래스의 인스턴스와 연관지어 실행되게 됨.
  - printName()만으로는 사용불가.
- 클래스 내부의 멤버 변수에 접근 가능 - private이라도.

```
class Person {
    public:
        int id;
        string name;
        int age;

        void printName() {
            cout << "My name is ";
            cout << name << endl;
        }
};

//멤버함수 호출.
Person dva = { 1, "Song Hana", 20 };
dva.printName();
```

객체의 동적할당

```
Person* p = new Person(101, "Dying Man", 99);
p->print();
delete p;
```

- 클래스의 포인터로 객체를 선언하고 동적으로 할당하는 것이 가능함.

- 이전에 배운 포인터 사용과 유사.

- 생성자를 위해 매개변수를 넘기는 것과, 포인터이므로 멤버 참조를 위해 ->를 사용한다는 점에 주의.

- delete로 할당된 메모리를 반환해야 함.

소멸자

```
~Person() {
    cout << "Destructor!! - " << name << endl;
}
```

- 클래스 이름에 ~를 붙여 정의.
- 객체의 사용이 끝나는 시점에 자동으로 호출됨.
  - 객체를 포인터로 선언하여 동적할당을 하고, delete로 삭제하는 경우.
  - 함수 등의 실행이 종료되어 객체가 소멸하는 경우.
- 만약 멤버 변수의 초기화에서 동적으로 메모리를 할당하는 경우 등 자원을 반환해야 하는 경우가 있다면, 소멸자에서 처리를 해주어야 함.

객체의 생명주기

```
Person mcCree(123, "Jesse McCree", 37);
```

- 클래스 선언 자체는 단지 클래스의 형태를 명시만 할 뿐임.

- 실제 클래스의 변수인 객체를 선언할 때, 이 객체에 메모리가 할당됨.

- 객체가 선언되면 우선 멤버변수들이 생성되고 생성자가 실행됨.

- 멤버 초기화는 멤버변수들의 생성시점이 생성자 내부코드 실행시점보다 빠르기 때문에 필요함.

- 이후 객체의 사용이 종료되면, 소멸자가 실행됨.

Summary

- 클래스 정의
- 멤버 변수와 멤버 함수
- 생성자와 소멸자