C++ 프로그래밍 [수행과제10] 성명 : 김민석 학번: 20145124

1. 클래스의 구조

클래스는 구조가

class 클래스명 {
    멤버변수;
    생성자() { } //소멸자() { }
    메소드() { }
}; 로 되어 있고 C언어에서 JAVA에서 쓰이는 class (객체와 다형성 등)를 C언어에서도 쓰기 위해 새롭게 추가된? 느낌의 개념이라고 생각하면 된다. 기본적으로 객체 내부에서만 쓰이는 private으로 선언되어 있으며 객체를 통해 다른 곳에서 사용하고 싶다면 public : 밑에 선언하면 된다.

2. 생성자의 구조 및 사용방법

클래스 안에서 선언이 되는 생성자와 소멸자는 우리가 직접 만든 class로 만들어진 '객체'가 생성되고 소멸할 때의 명령 부분으로써 class의 멤버 함수 이다. 또는 public:을 선언해주고 사용해야 한다. 생성자는 객체가 '생성'될 때, 즉 선언부분에 있으면 초기값을 지정해주거나 필요한 초기화 절차를 실행하는 함수이다. 반대로 소멸자는 객체가 '소멸'할 때 호출되어, 생성자와 다르게 값을 지정하거나 중복정의가 불가능하다.

생성자의 형태는 '클래스명() { }'이고, 소멸자의 형태는 '~클래스명() { }' 이다.

예를 들자면

```
class test {
public:
    test() { ~ } // 생성자, 객체 실행시 ~ 부분 실행. 매개 변수를 받을 수 있다.
    ~test() { ~ } // 소멸자, 객체 소멸시 ~ 부분 실행
```

3. 메소드의 구조 및 사용방법

메소드는 생성자/소멸자와 같이 class 내부에서 선언되며 원형은 class 내부에, 그리고 내용은 외부에 선언할 수 있다. 또한 함수라고도 불리며, 그 객체의 '동작'을 나타내는 부분이다.

예를 들어 -

```
class car {
public:
    // 생성자, 소멸자
    void move(int x) {
        std::cout << x << "만큼 이동" << endl;
    }
};
```

이 있다면 메소드 move는 car 클래스로 만들어지는 모든 객체에 들어가 있는 '동작' 이라고 할 수 있다.

4. 객체 생성과 멤버 접근 방법

객체를 생성하는 방법은 다양한데 가장 기본은 '클래스명 객체명;' 이다. 이때는 default 생성자가 호출이 되며 JAVA에서처럼 new 키워드를 쓰고 싶을땐 다음과같이 객체를 동적으로 생성하면 된다.

```
Car * c = new Car(10);
c -> setSpeed(5);
:
delete c;
```

또는 Car c(10, "1단"); 생성자 호출

아니면 Car c = Car(10, "1단"); 임시 객체 처럼 선언해도 된다.

5. 클래스 멤버 함수의 외부 정의 방법

함수(메소드)의 '외부' 정의라는 것은 기본적으로 class 내에 구현되는 함수는 원형이자 '내부' 함수이다.
이를 class 내부에서 원형, 즉 그러할 함수가 있을 것만 보여주고 그 안에서 이루어지는 작업 등은 외부에서 할 수 있게 정의하는 방법이다. class 내에는 함수의 원형 '반환형 함수이름(매개변수);' 까지(끝에 ;세미콜론 필수) 쓰고
class 외부에서 '반환형 클래스명:: 함수명(매개변수) { 내부실행문; }' 를 쓰면 된다.

ex) class T {
~ // 생성자, 멀버함수 등
int getScore();
}

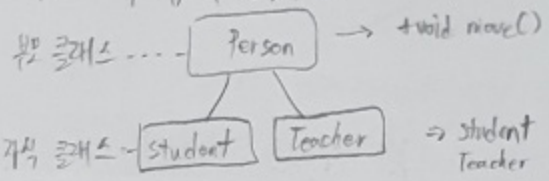
int T::getScore() { return score; } 이다.

* 여기 '::' 는 범위지정연산자(Scope Operator)로써 이함수가 어디 내부에 존재해 있는지 알수있게 해준다.

6. 상속을 적용하는 방법

상속은 유전자와 같이 생각하면 되는데, 상속을 해주고 받는 클래스가 정해지며 상속을 받는 클래스는 상속을 해주는 클래스의 속성(멤버변수)과 기능(메소드 등)을 쓸 수있게 된다. 이때 상속을 받는 클래스를 자식 클래스, 해주는 클래스를 부모클래스라고도 한다. 이를 통해 클래스를 재사용(Recycling) 할 수 있고 연관된 클래스간의 공통된 코드를 줄일 수 있다.

JAVA에서는 'class 자식클래스명 extends 부모클래스명 { }' 의 형식이었다면
C++ 에서는 'class 〃 : public 〃 { }' 의 형식이다.



⇒ Student S; S->move() 가능!
Teacher t; t->move()

7. 가상함수의 사용방법

가상함수는 JAVA의 Interface처럼 한 클래스에 있는 함수를 파생클래스가 구현하고 싶을 때 쓰이며 함수의 반환형 앞에 virtual을 붙임으로써 '가상' 함수임을 보인다. 이처럼 함수를 다시 정의하는 방법은 함수 재정의과 가상함수의 방법이 있는데 함수 재정의는 반환형, 이름, 매개변수가 같은 함수를 대놓고써 원래 가지고 있는 함수를 쓸수있는 기회가 없지만 가상함수에선 본래 함수의 존재가 의미없어져서 호출하려해도 '동적 바인딩'이 일어나 파생클래스에서 구현한 함수가 호출된다.
동적바인딩이 있으므로 특별히 부모클래스에서 어떤 동작을 수행하지 않는다면, 그냥 '순수가상 함수'로 구현해놓고 내용은 자식클래스에서 구현하면 된다. 이때 순수가상함수는 'virtual 반환형 함수명(매개변수) = 0;' 으로 하면 된다.

8. 예외처리를 사용하는 방법

예외처리는 알고리즘이나 코드가 복잡한 프로그래밍 과정에서 오류가 날 것 같은 곳에 쓸으로써 오류를 사용자가 원하는 대로 처리하는 방법이다. 사용법은 'try { ~내용; throw (예외 데이터 값); } catch (대응하는 반환형 매개변수) { ~예외시 처리문 }'으로 되어 있다.

throw는 항상 try문 내부에 있어야 하며 throw로 던져하게 될 예외시 대응값에 맞는 catch문(중복선언 가능)이 없다면 프로그램을 강제종료시키니 조심하자

ex)
```
int main() {
    int n;

    try {
        cout << "함수 입력" << endl;
        cin >> n;
        if(n <= 0) throw (-1);
        cout << "입력한 수 : " << n << endl;
    }
    catch(const int i) {
        cout << "Error code : " << i << endl;
    }
}
```

생성자, 멤버변수 접근, 외부 함수 예시 등 추가 설명

```
class Student {
    int score; // 기본적으로 private
public:
    Student(int s) : score(s) {
        // 그외에 추가 초기화 내용부
    } // 소멸자 제거
    void setScore(int s);
    int getScore();
} // Student 클래스

void Student::setScore(int s) { // 외부 함수 예시
    score = s;
}
int Student::getScore() {
    return score;
}
```

생성자는 메개변수로 객체 생성시 그 객체의 멤버 변수의 값을 지정할 수 있고 ':'을 통해 바로 값을 넣을 수 있으며 추가 선언부에서도 초기화 등을 할 수 있다.

일반적으로 private (객체 내에서만 사용 가능)인 멤버 변수들을 main에서 객체를 통해 가져오고 값을 바꿀 수 없는 것을 설정자(set), 접근자(get)이라는 JAVA와 같이 함수를 통해 접근하고 값을 바꿀 수 있다.

범위지정연산자 (::) 덕에 그 클래스 내부에 선언된 private 멤버를 볼 수 있다.

C언어에서 없던 class가 C++에선 나온건데 그럴거면 구조체(struct)처럼 하나만 없어보이는데 왜 쓰이는 것일까?

1. 절차 지향 대신 객체 지향을 통해 더 다른 프로그램을 만들기 위해서
2. 구조체는 기본적으로 public 이고 객체는 기본적으로 private. 즉, 캡슐화가 쉽다.
3. 구조체가 가질 수 없는 메소드의 존재 등