

학번:

이름:

1. 기말시험이다. 중요 내용을 잠깐 정리해보도록 하자.

C++ 언어의 패러다임 중에서 객체지향에 관한 내용을 다룬 수업이었다. C++ 언어에서는 키워드 `class`를 이용하여 사용자가 새로운 자료형을 정의할 수 있다. C++ 언어는 `class` 안에 모델링 하고자 하는 객체의 속성을 나타내는 멤버변수와 함께 객체의 동작을 표현하는 멤버함수를 프로그램할 수 있다. C++ 언어에서는 `struct`를 사용하여도 `class`와 같은 프로그램을 만들 수 있지만 `struct`는 모든 멤버변수와 함수가 `public`인 경우에 사용하는 것이 바람직하다. `Class`에서는 특별한 경우를 제외하고는 멤버변수를 항상 `private`으로 만들어 `class` 밖에서의 직접적인 접근을 제한하는 것이 원칙이다. 이를 정보은폐라는 말로 표현하기도 한다.

새로운 자료형은 왜 만드는 것인가? 단순히 하나의 대상만을 `class`로 모델링하는 것이 구현하고자 하는 프로그램에서 가치있는 일인가 생각해봐야 한다. `Class`는 단독으로 만들어 사용하려고 만들기 보다는 상속(`inheritance`)이라는 개념을 구현하기 위한 목적으로 만든다. `Class A`와 `B`사이에 “A is a B.” 라는 문장이 성립한다면 A를 B의 자식 클래스로 만들 수 있다. 여러 클래스를 상속 관계로 구성하는 최종 목적은 다형성(`polymorphism`)을 구현하기 위한 것이다. 다형성이란, 하나의 명령으로 객체마다 특성에 맞는 행동이 구현됨을 의미한다.

C++의 `class`를 배웠으니 구상하고 있는 게임에 적용해 보고자 한다. 실제로는 각 게임을 구현할 때마다 기획에 맞는 다양한 `class`들이 필요할 것이다. 일단 간단하게 시작하기로 하자. 내가 구상하고 있는 게임에는 `player`와 `monster`가 있다. 이들을 `class` 상속 관계로 구성하고 싶다. 그런데 이 두 `class`는 모두 이동 가능(`movable`)하고 화면에 그려질 수 있다(`renderable`)는 특성을 갖는다. 따라서 이들의 공통 속성을 갖는 `GameObject`를 부모 `class`를 만들기로 한다. `GameObject` `class`는 객체로 만들어 질 클래스는 아니고 다형성을 구현하기 위한 공통의 부모 클래스로 만들면 적당할 것이다. 다음과 같이 만들어야 할 클래스와 멤버변수 그리고 멤버함수를 표로 정리해 보았다.

클래스	변수		함수	
GameObject	Point	struct Point { int x, int y };	void move(x, y) void render() void show()	x, y 위치로 이동한다 화면에 출력한다 자신의 정보를 출력한다.
Player	HP level	체력을 나타내는 정수 레벨을 나타내는 정수		
Monster	HP	체력을 나타내는 정수		

[표] 필요 클래스 간단 정리(초기 버전)

시작은 단순하고 보잘 것 없어 보이지만 모든 것이 처음부터 완벽할 수는 없는 법이다. 문제를 따라가며 프로그램을 시작해 보자.

[주의] 문제가 요구하는 답만을 답지에 적을 것.

문제를 간단히 하기 위해서 파일 분리는 하지 않고 하나의 소스파일만을 사용한다.

[문제1] 표의 세 class 간의 상속 관계를 그림으로 표현하라. (10)

[문제2] class GameObject를 프로그램하라. (20)

GameObject는 객체로 만들 수 없으므로 추상 클래스(abstract class)로 만들어야 한다.

GameObject는 struct Point를 private 멤버로 갖는다.

move(x, y) 함수를 이용하여 Point 멤버변수의 값을 변경한다.

render() 함수는 순수 가상함수이다.

show() 함수는 GameObject의 현재 위치를 다음과 같이 화면에 표시한다.

[화면출력] 현재 위치는 12, 10입니다.

[문제3] class Player를 프로그램하라. (20)

멤버변수 HP와 level은 생성자의 인수로 초기화한다.

render() 함수는 단순히 화면에 그려진다고 출력한다.

[화면출력] 플레이어를 그립니다.

show() 함수는 GameObject의 show()를 이용하여 다음과 같이 출력한다.

[화면출력] 현재 위치는 20, 12입니다.

HP는 100, level은 33입니다.

Monster는 Player와 유사하므로 쉽게 만들 수 있을 것이다.

지금까지 프로그램이 잘 되었다면 다음의 main 프로그램이 문제없이 컴파일되고 실행되어야 한다.

```
//-----
int main()
//-----
{
    system( "chcp 949" );           // 커맨트창에 한글이 출력안되는 경우 사용

    cout << endl;
    GameObject* objs[2];

    objs[0] = new Player( Point( 1, 2 ), 100, 33 );
    objs[1] = new Monster( Point( 1, 0 ), 3 );

    for ( int i = 0; i < 2; ++i )
        objs[i]->render();

    cout << endl;

    for ( int i = 0; i < 2; ++i )
        objs[i]->show();
}
```

위 프로그램이 실행된 후의 출력화면은 다음과 같을 것이다.

```
C:\Windows\system32\cmd.exe
활성 코드 페이지: 949

플레이어를 그립니다
몬스터를 그립니다

현재 위치는 1, 2입니다.
HP는 100, level은 33입니다.
현재 위치는 1, 0입니다.
HP는 3입니다.
Press any key to continue . . .
```

Player와 Monster를 heap에 동적으로 생성한 후 부모 클래스인 GameObject의 포인터로 객체를 관리하고 있다. GameObject의 멤버함수 show()와 render()를 실행시켰는데 원하는 대로 Player와 Monster의 멤버함수가 수행되는 것을 볼 수 있다. 다형성이 제대로 구현된 것이다. 다형성을 구현하면 프로그램을 읽기 쉽고 주요 부분의 코드가 간단하게 된다. 또한, 향후 요구사항 변경에 의하여 GameObject를 상속받는 새로운 객체가 추가되는 경우에도 다형성을 이용하여 구현한 코드는 수정할 필요가 없다는 점이 중요하다. 프로그램의 유지보수에 이점이 있다는 말이다.

현재 프로그램은 객체를 new로 생성하고 있다. new로 만든 객체는 반드시 delete로 소멸시켜야 한다. 객체가 제대로 소멸되는가를 확인하기 위하여 소멸자가 호출될 때 화면에 객체가 소멸된다고 화면에 출력하기로 하자.

[문제4] 소멸자를 프로그램하라. (추가되는 부분만 적을 것) (20)

메인 프로그램에서 다음과 같이 객체를 소멸하여야 하며 소멸자가 제대로 호출되면 다음과 같은 실행화면을 볼 수 있을 것이다.

```
//-----
int main()
//-----
{
    // 위의 프로그램과 같으므로 중간 부분을 생략함
    // main()이 끝나기 전에 아래의 두 줄로 객체를 지운다.

    delete objs[0];
    delete objs[1];
}
```

```
현재 위치는 1, 0입니다.
HP는 3입니다.
플레이어 소멸
몬스터 소멸
Press any key to continue . . .
```

[실행화면] 위부분 생략

GameObject의 move(x, y) 함수는 x, y 위치로 객체를 옮기는 함수이다. 그런데 player의 현재 위치 x, y에서 원하는 Point의 위치만큼을 이동하는 기능이 필요하여 += 연산자를 오버로딩하기로 하였다.

[문제5] class GameObject에 += 연산자를 오버로딩하라. (추가되는 부분만 적을 것) (15)

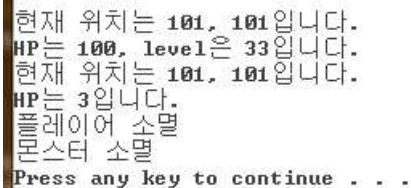
+= 연산자의 리턴값은 void이다.

main()에 추가된 부분이 컴파일되고 실행되어야 한다. 실행된 결과는 그림을 참고한다.

```
//-----
int main()
//-----
{
    // 위의 프로그램과 같으므로 중간 부분을 생략함
    cout << endl;
    for ( int i = 0; i < 2; ++i )
        *objs[i] += Point( 100, 100 );    // += 연산자 오버로딩

    for ( int i = 0; i < 2; ++i )
        objs[i]->show();

    delete objs[0];
    delete objs[1];
}
```



```
현재 위치는 101, 101입니다.
HP는 100, level은 33입니다.
현재 위치는 101, 101입니다.
HP는 3입니다.
플레이어 소멸
몬스터 소멸
Press any key to continue . . .
```

[실행화면] 위부분 생략

GameObject의 show() 함수는 화면에 현재 좌표를 출력하고 있다. 이렇게 단순히 출력하는 기능은 출력 연산자를 오버로딩하여 구현할 수도 있다.

[문제6] GameObject의 << 연산자를 오버로딩하라. (추가되는 부분을 정확히 적을 것) (15)

다음의 코드가 컴파일되고 실행되어야 한다.

아래의 main()은 지금까지 문제와 관련된 전체 소스이다.

```
//-----
int main()
//-----
{
    system( "chcp 949" );    // 커맨트창에 한글이 출력안되는 경우 사용
```

```

cout << endl;
GameObject* objs[2];

objs[0] = new Player( Point( 1, 2 ), 100, 33 );
objs[1] = new Monster( Point( 1, 0 ), 3 );

for ( int i = 0; i < 2; ++i )
    objs[i]->render();

cout << endl;

for ( int i = 0; i < 2; ++i )
    objs[i]->show();

cout << endl;
for ( int i = 0; i < 2; ++i )
    *objs[i] += Point( 100, 100 );

for ( int i = 0; i < 2; ++i )
    objs[i]->show();

cout << endl ;
cout << *objs[0] << *objs[1] << endl;    // 출력 연산자 오버로딩 부분

delete objs[0];
delete objs[1];
}

```

```

C:\Windows\system32\cmd.exe
활성 코드 페이지: 949
플레이어를 그립니다
몬스터를 그립니다
현재 위치는 1, 2입니다.
HP는 100, level은 33입니다.
현재 위치는 1, 0입니다.
HP는 3입니다.
현재 위치는 101, 101입니다.
HP는 100, level은 33입니다.
현재 위치는 101, 101입니다.
HP는 3입니다.
현재 위치는 101, 101입니다.
현재 위치는 101, 101입니다.
플레이어 소멸
몬스터 소멸
Press any key to continue . . .

```

아주 작은 프로그램이지만 C++ 언어에서 class와 관련된 내용은 이것으로 충분하다
 하지만 아직 구상한 게임 프로그램은 시작도 하지 않았다.
 앞으로 공부해야할 것들이 정말 많을 것임을 알 수 있다. 시간 많은 겨울 방학이 기대된다.

- 앞으로는 즐거운 프로그램 하세요.-

```

#include <iostream>
using namespace std;

struct Point {
    int x;
    int y;

    Point( int a, int b ) : x( a ), y( b ) { }
};

class GameObject {
    Point p;
public:
    GameObject( Point p ) : p( p ) { }
    virtual ~GameObject() { }
    virtual void move( int x, int y ) { p.x = x; p.y = y; }
    virtual void show() { cout << "현재 위치는 " << p.x << ", " << p.y << "입니다. " << endl; }
    virtual void render() = 0;

    void operator+=( const Point& q ) {
        move( p.x+q.x, p.y+q.y );
    }
    friend ostream& operator<<( ostream& os, const GameObject& p );
};

ostream& operator<<( ostream& os, const GameObject& g )
{
    cout << "현재 위치는 " << g.p.x << ", " << g.p.y << "입니다. " << endl;
    return os;
};

class Player : public GameObject {
    int HP, level;
public:
    Player( Point p, int h, int l ) : GameObject( p ), HP( h ), level( l ) { }
    virtual ~Player() { cout << "플레이어 소멸" << endl; }
    virtual void show() {
        GameObject::show();
        cout << "HP는 " << HP << ", level은 " << level << "입니다. " << endl;
    };
    virtual void render() {
        cout << "플레이어를 그립니다" << endl;
    }
};

class Monster : public GameObject {
    int HP;
public:
    Monster( Point p, int h ) : GameObject( p ), HP( h ) { }
    virtual ~Monster() { cout << "몬스터 소멸" << endl; }
    virtual void show() {
        GameObject::show();
        cout << "HP는 " << HP << "입니다. " << endl;
    }
};

```

```

    };
    virtual void render() {
        cout << "몬스터를 그립니다" << endl;
    }
};

//-----
int main()
//-----
{
    system( "chcp 949" );           // 커맨트창에 한글이 출력안되는 경우 사용

    cout << endl;
    GameObject* objs[2];

    objs[0] = new Player( Point( 1, 2 ), 100, 33 );
    objs[1] = new Monster( Point( 1, 0 ), 3 );

    for ( int i = 0; i < 2; ++i )
        objs[i]->render();

    cout << endl;

    for ( int i = 0; i < 2; ++i )
        objs[i]->show();

    cout << endl;
    for ( int i = 0; i < 2; ++i )
        *objs[i] += Point( 100, 100 );

    for ( int i = 0; i < 2; ++i )
        objs[i]->show();

    cout << endl ;
    cout << *objs[0] << *objs[1] << endl;

    delete objs[0];
    delete objs[1];
}

```