

객체지향프로그래밍 LAB #12

<기초문제>

1. 아래의 프로그램을 작성하시오. (/*구현*/ 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

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

class Base {
protected: //Base type
    void print_base() { cout << "Base" << endl; }
};

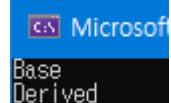
// Base type | 상속 type | Derived type
// private   | 상관없음   | 접근 불가( Base함수로 접근)
// protected | 상관없음   | private
// public     | private/protected | private
// public     | public      | public

class Derived : private Base {
public:
    void print_derived() {
        /*구현*/
        cout << "Derived" << endl;
    }
};

int main() {
    Base base;
    Derived derived;

    /* 구현 */

    return 0;
}
```



```
Microsoft Windows [Version 6.0.6002.18005]
(c) 2009 Microsoft Corporation. All rights reserved.

C:\>

Base
Derived
```

2. 아래의 프로그램을 작성하시오. (/*구현*/ 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

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

// 함수 오버로딩: int    sum(int x, int y),
//                double sum(double x, double y)
//                float  sum(float x, float y, float z)
// 함수 오버라이딩 (상속의 특수한 경우 사용)
//void Text::append(string _extra)
//void Fancy::append(string _extra)

class Text {
private:
    string text;
public:
    Text(string _t) : text(_t) {}
    /*구현*/ //get() 함수 virtual 로 구현
    virtual void append(string _extra) { text += _extra;}
};


class FancyText : public Text {
private:
    // string text; 접근이 안됨, Base Class에서 private
    string left_brac;
    string right_brac;
    string connector;
public:
    // initialization list는 생성자를 호출할 수 있게 해준다.
    FancyText(string _t, string _lb, string _rb, string _con) :
        Text::Text(_t), left_brac(_lb), right_brac(_rb), connector(_con) {}
    /*구현*/ //override 키워드 사용한 get() 함수 구현, main 함수 참고하여 출력화면처럼
되도록 구현
    /*구현*/ //override 키워드 사용한 append() 함수 구현
};

class FixedText : public Text {
public:
    FixedText() : Text::Text("FIXED") {}
    /*구현*/ //override 키워드 사용한 append() 함수 구현, main 함수 참고하여 출력화면처럼
되도록 구현
};

int main() {
    Text t1("Plain");
    t1.append("A");
    cout << t1.get() << endl;

    FancyText t2("Fancy", "<<", ">>", "***");
    t2.append("A");
    cout << t2.get() << endl;
}
```

```
FixedText t3;  
t3.append("A");  
cout << t3.get() << endl;  
t1 = t2; // Base <- Derived 가능  
//t2 = t1; // Derived <- Base 불가능  
  
return 0;  
}
```

 Microsoft Visual Studi

```
PlainA  
<<Fancy+++A>>  
FIXED
```

<응용문제>

1. 아래의 코드를 기반으로 Polygon Class를 상속받는 Rectangle Class를 선언하고, 둘레와 넓이를 구하는 프로그램을 작성하시오.

```
class Polygon {
public:
    Polygon() {}
    Polygon(int point, float length) { /* 구현 */ }
    ~Polygon() {}
    virtual void calcPerimeter() { /* 구현 */ }
    virtual void calcArea() { /* 구현 */ }
protected:
    int mPoint; // 꼭지점의 갯수
    double mLength; // 한 변의 길이
};

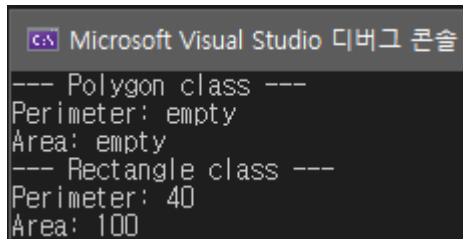
class Rectangle : public Polygon {
public:
    Rectangle() {}
    Rectangle(int point, float length) : /* 구현 */ {}
    ~Rectangle() {}
    void calcPerimeter() override { /* 구현 */ }
    void calcArea() override { /* 구현 */ }
};

int main() {
    Polygon pol;
    Rectangle rec(4, 10);

    cout << "--- Polygon class ---" << endl;
    pol.calcPerimeter();
    pol.calcArea();
    cout << "--- Rectangle class ---" << endl;
    rec.calcPerimeter();
    rec.calcArea();

    return 0;
}
```

1-출력화면:



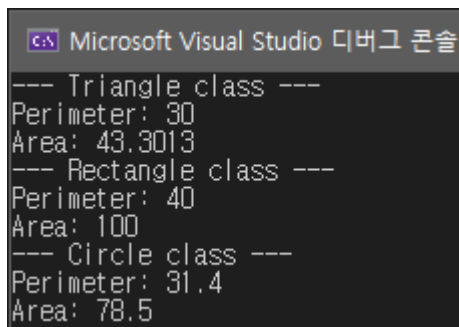
```
Microsoft Visual Studio 디버그 콘솔
--- Polygon class ---
Perimeter: empty
Area: empty
--- Rectangle class ---
Perimeter: 40
Area: 100
```

2. 1번 문제에 이어 Polygon Class를 상속받는 Triangle, Circle Class를 추가로 작성하시오. 단, 모든 도형은 정다각형이라 가정함.

```
int main() {
    Triangle tri(3, 10);
    Rectangle rec(4, 10);
    Circle cir(0, 5);
    cout << "--- Triangle class ---" << endl;
    tri.calcPerimeter();
    tri.calcArea();
    cout << "--- Rectangle class ---" << endl;
    rec.calcPerimeter();
    rec.calcArea();
    cout << "--- Circle class ---" << endl;
    cir.calcPerimeter();
    cir.calcArea();

    return 0;
}
```

2-출력화면:



```
Microsoft Visual Studio 디버그 콘솔
--- Triangle class ---
Perimeter: 30
Area: 43.3013
--- Rectangle class ---
Perimeter: 40
Area: 100
--- Circle class ---
Perimeter: 31.4
Area: 78.5
```

3. KTX 열차가 출발역, 종착역을 포함하여 총 5개의 역에 정차한다. 이때 각 역에서는 입력된 숫자만큼의 사람이 내리거나 탑승한다 이 기차는 아래와 같은 조건을 만족하면서 운행된다고 가정한다.

- 기차는 역 번호 순서대로 운행한다.
- 출발역에서 내린 사람 수와 종착역에서 탄 사람 수는 0이다.
- 각 역에서 현재 기차에 있는 사람보다 더 많은 사람이 내리는 경우는 없다.
- 기차의 정원은 최대 300명이고, 정원을 초과할 수 없다.

5개 역에 대해 기차에서 내린 사람 수와 탄 사람 수가 주어졌을 때, 기차에 사람이 가장 많을 때의 사람 수를 계산하는 프로그램을 작성하시오.

```
class Train {
public:
    Train() {}
    Train(int people) { /* 구현 */ }
    ~Train() {}
    virtual int station(int takeOff, int takeOn) { /* 구현 */ }
protected:
    int mPeople; // 사람 수
};
```

```

class Ktx : public Train {
public:
    Ktx() : /* 구현 */ {}
    Ktx(int people) : /* 구현 */ {}
    ~Ktx() {}
    // 기차에 사람이 타고 내리는 함수
    int station(int takeOff, int takeOn) { /* 구현 */ }
    int getPeople() { /* 구현 */ }
};

int main()
{
    Ktx k;
    /* 구현 */
    return 0;
}

```

3-출력화면:

```

Microsoft Visual Studio 디버그 콘솔
1번역: 0 210
2번역: 40 63
3번역: 50 20
4번역: 27 25
5번역: 201 0
가장 많은 사람이 탑승 했을 때의 사람 수: 233

```

```

Microsoft Visual Studio 디버그 콘솔
1번역: 0 210
2번역: 143 34
3번역: 200 20
정원미달입니다

```

```

Microsoft Visual Studio 디버그 콘솔
1번역: 0 250
2번역: 18 84
정원초과입니다

```

4. 어벤져스 캐릭터 배틀 프로그램을 만들려고 한다. [시작코드]와 출력화면을 바탕으로 아래의 조건에 맞게 구현하시오. 단, Character 클래스는 Avengers 클래스를 상속받음.

- 사용자는 캐릭터를 선택하고 상대방 캐릭터는 랜덤으로 선택된다. (상대방 캐릭터와 사용자 캐릭터 중복 가능)
- 각 캐릭터는 캐릭터 이름, 공격력, 방어력, 체력을 갖는다.
- 공격하는 함수는 상대방에게 공격력을 가하고, 공격받는 함수(방어 함수)는 상대방의 공격력에서 자신의 방어력의 차만큼 자신의 체력을 감소한다.
- 서로 공격을 주고받다가, 어느 한쪽의 체력이 0 이하가 되면 배틀을 종료한다. 공격을 주고받을 때 마다 자신과 상대방의 캐릭터 체력을 출력하고, 선포는 자신의 캐릭터가 먼저 하는 것으로 설정한다.

	IronMan	CaptainAmerica	Thor
공격력	70	60	80
방어력	40	50	30
체력	100	100	100

[시작코드]

```
class Avengers {
public:
    Avengers() {
        name = "";
        attack_point = 0;
        defense_point = 0;
        health = 0;
    }
    ~Avengers() {}
    // 캐릭터 설정 함수
    virtual void set(string _name, int _attack, int _defense, int _health) {}
    // 공격 함수
    virtual int attack() { return 0; }
    // 방어 함수
    virtual void defense(int _attack_point) { }
    // 캐릭터 정보 출력 함수
    virtual void print_info() { }
protected:
    string name;           // 캐릭터 이름
    int attack_point;      // 공격력
    int defense_point;     // 방어력
    int health;            // 체력
};

class Character : public Avengers {
public:
    /* 구현 */
    int get_health() { return health; }
};

int main() {
    Character my_char;
    Character enemy_char;

    /* 구현 */

    cout << endl << "--Battle--" << endl;
    cout << "My Life: " << my_char.get_health() << "Wt"
        << "Enemy Life:" << enemy_char.get_health() << endl;

    while (1) { /* 구현 */ }

    return 0;
}
```

4-출력화면:

```
Microsoft Visual Studio 디버그 콘솔
Choose your character(IronMan, CaptainAmerica, Thor): IronMan
--My Character--
Name: IronMan
Attack_Point: 70
Defense_Point: 40
Health: 100
--Enemy Character--
Name: CaptainAmerica
Attack_Point: 60
Defense_Point: 50
Health: 100

--Battle--
My Life: 100      Enemy Life:100
My Life: 100      Enemy Life:80
My Life: 80       Enemy Life:80
My Life: 80       Enemy Life:60
My Life: 60       Enemy Life:60
My Life: 60       Enemy Life:40
My Life: 40       Enemy Life:40
My Life: 40       Enemy Life:20
My Life: 20       Enemy Life:20
My Life: 20       Enemy Life:0
You Win!
```