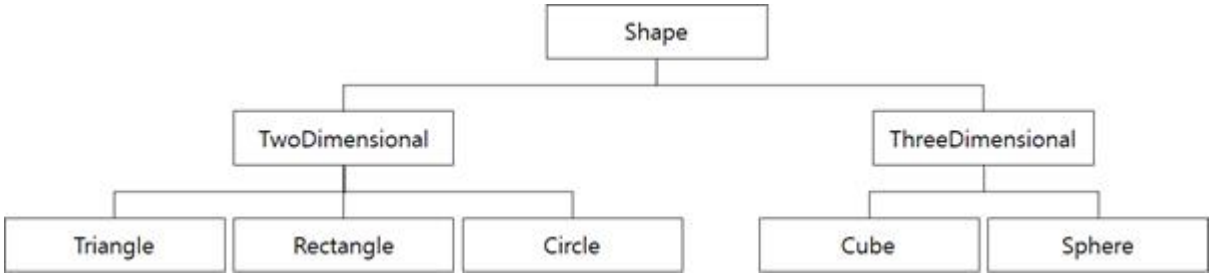
# 실습 11 - 12주차

학과 : 전자공학과 학번 : 2023104322 이름 : 현시온

- 과제는 pdf로 변환하여 제출(과제 문서 첫 줄에 학과/학번/이름 포함)
- 과제는 순서대로 작성하며, 문제와 설명을 모두 포함(형식이 맞지 않으면 감점)
- 프로그램을 작성하는 문제는 소스코드와 실행 결과를 모두 text로 붙여넣기(그림으로 포함하지 말 것)하고 코드 설명 및 결과에 대한 설명을 포함해야 함
- 문의 사항은 이메일(nize@khu.ac.kr) 또는 오픈 카톡방을 이용

1. 아래의 그림과 특징을 가지는 클래스를 선언하고 정의하라. (테스트할 코드도 작성해서 결과를 포함할 것)




A. Shape, TwoDimensional, ThreeDimensional 클래스는 추상 클래스이다.

B. Shape 클래스는 면적을 계산하고 출력하는 순수 가상 함수를 가진다.

C. TwoDimensional 클래스는 둘레를 계산하고 출력하는 순수 가상 함수를 가진다.

D. ThreeDimensional 클래스는 부피를 계산하고 출력하는 순수 가상 함수를 가진다.

E. Shape 클래스의 PI는 static 멤버이다. (원주율로 소수점 5자리까지 지정)

F. 삼각형, 직사각형, 원, 정육면체, 구를 각각 Triangle, Rectangle, Circle, Cube, Sphere 클래스로 정의하고, 각 도형을 표현할 수 있는 변수를 정의하고, 필요한 함수를 정의하라.

코드 및 설명:

```
#include <iostream>
#include <cmath>

class Shape { //클래스 Shape 선언 및 정의
public:
        static double PI;
```

```
        //static 멤버 변수 PI 선언
        virtual void area() = 0;
        //면적 계산 및 출력의 순수 가상 함수 area 선언
};
double Shape::PI = 3.14159;
//static 멤버 변수인 PI를 외부에서 정의
class TwoDimensional : public Shape { //Shape의 파생 클래스 TwoDimensional 선언 및 정의
public:
        virtual void circum() = 0;
        //둘레 계산 및 출력의 순수 가상 함수 circum 선언
};
class Triangle : public TwoDimensional { //TwoDimensional의 파생 클래스 Triangle 선언 및 정의
        double a1, a2, a3; //멤버 변수 선언
public:
        Triangle(double a1, double a2, double a3) : a1(a1), a2(a2), a3(a3) {}; //생성자 선언
및 정의
        void area() override { //함수 area의 오버라이드 선언 및 정의, 삼각형의 넓이를 출력하는
내용
                double s = (a1 + a2 + a3) / 2;
                std::cout << pow(s * (s - a1) * (s - a2) * (s - a3), 0.5) << std::endl;
        }
        void circum() override { //함수 circum의 오버라이드 선언 및 정의, 삼각형의 둘레를
출력하는 내용
                std::cout << a1 + a2 + a3 << std::endl;
        }
};
class Rectangle : public TwoDimensional { //TwoDimensional의 파생 클래스 Rectangle 선언 및 정의
        double a1, a2; //멤버 변수 선언
public:
        Rectangle(double a1, double a2) : a1(a1), a2(a2) {}; //생성자 선언 및 정의
        void area() override { //함수 area의 오버라이드 선언 및 정의, 직사각형의 넓이를
출력하는 내용
                std::cout << a1 * a2 << std::endl;
        }
        void circum() override { //함수 circum의 오버라이드 선언 및 정의, 직사각형의 둘레를
출력하는 내용
                std::cout << 2 * (a1 + a2) << std::endl;
        }
};
class Circle : public TwoDimensional { //TwoDimensional의 파생 클래스 Circle 선언 및 정의
        double a1; //멤버 변수 선언
public:
        Circle(double a1) : a1(a1) {}; //생성자 선언 및 정의
        void area() override { //함수 area의 오버라이드 선언 및 정의, 원의 넓이를 출력하는
내용
                std::cout << a1 * a1 * PI << std::endl;
        }
        void circum() override { //함수 circum의 오버라이드 선언 및 정의, 원의 둘레를 출력하는
내용
                std::cout << 2 * a1 * PI << std::endl;
        }
};

class ThreeDimensional : public Shape { //Shape의 파생 클래스 ThreeDimensional 선언 및 정의
public:
```

```
        virtual void volume() = 0;
        //부피 계산 및 출력의 순수 가상 함수 volume 선언
};
class Cube : public ThreeDimensional { //ThreeDimensional의 파생 클래스 Cube 선언 및 정의
        double a1; //멤버 변수 선언
public:
        Cube(double a1) : a1(a1) {}; //생성자 선언 및 정의
        void area() override { //함수 area의 오버라이드 선언 및 정의, 정육면체의 겉넓이를
출력하는 내용
                std::cout << 6 * a1 * a1 << std::endl;
        }
        void volume() override { //함수 volume의 오버라이드 선언 및 정의, 정육면체의 부피를
출력하는 내용
                std::cout << a1 * a1 * a1 << std::endl;
        }
};
class Sphere : public ThreeDimensional { //ThreeDimensional의 파생 클래스 Sphere 선언 및 정의
        double a1; //멤버 변수 선언
public:
        Sphere(double a1) : a1(a1) {}; //생성자 선언 및 정의
        void area() override { //함수 area의 오버라이드 선언 및 정의, 구의 겉넓이를 출력하는
내용
                std::cout << 4 * PI * a1 << std::endl;
        }
        void volume() override { //함수 volume의 오버라이드 선언 및 정의, 구의 부피를 출력하는
내용
                std::cout << 4. / 3. * PI * a1 * a1 * a1 << std::endl;
        }
};
int main() { //객체 생성 및 메소드 호출을 통해 결과 확인
        Triangle tri(2, 2, 3);
        tri.area();
        tri.circum();
        Rectangle rec(1, 2);
        rec.area();
        rec.circum();
        Circle cir(1);
        cir.area();
        cir.circum();
        Cube cub(1);
        cub.area();
        cub.volume();
        Sphere sph(1);
        sph.area();
        sph.volume();
}
```

2. 아래의 주석과 같이 동작하는 클래스를 선언하고 정의하라.

```
// All data members of Base and Derived classes must be declared
// as private access types
Base *p1 = new Derived(10, 20); // (x, y)
Base *p2 = new Base(5); // (x)
```

```
p1->print(); // prints 10, 20
p1->Base::print(); // prints 10
p2->print(); // prints 5
Derived *p3 = dynamic_cast<Derived *>(p1);
if (p3 != nullptr) p3->print(); // prints 10, 20
const Base b1 = *p2;
b1.print(); // prints 5
Derived d1(1, 3), d2(2, 4);
Derived d3 = (d1 < d2) ? d1 : d2; // operator <: (d1.x+d1.y) < (d2.x+d2.y)
d3.print(); // prints 1, 3
```

코드 및 설명:

```
#include <iostream>
class Base {
	int x;
public:
	Base(int x = 0) : x(x) {};
	virtual void print() const {
		std::cout << x << std::endl;
	}
	int getx() const {
		return x;
	}
};
class Derived : public Base {
	int z;
public:
	Derived(int y, int z) : Base(y), z(z) {};
	void print() const override {
		std::cout << getx() << ", " << z << std::endl;
	}
	int gety() const {
		return getx();
	}
	int getz() const {
		return z;
	}
};
bool operator < (const Derived& a, const Derived& b) {
	return (a.gety() + a.getz()) < (b.gety() + b.getz());
}
//p1의 타입은 클래스 Base의 객체의 포인터인데, 함수 print를 호출할 경우 할당된 클래스 Derived의 메소드 print가 호출되므로 print는 가상 함수여야 한다.
//또한 print가 가상 함수여야 클래스가 polymorphism을 만족하기 때문에 dynamic casting도 가능해진다.
//클래스에 따라 호출되는 함수 print의 내용이 다르기 때문에 override해주어야 한다.
//클래스 Derived의 객체여도 클래스 Base의 함수 print를 호출하면 default 생성자에 의해서가 아닌 입력해준 첫번째 인수를 사용한다. 즉 Derived의 생성자에서 Base의 객체를 호출해야 한다.
//const 객체는 생성 후 수정 불가능한 것과 더불어 클래스의 const 메소드만 호출 가능하다. 즉 함수 print는 const 메소드여야 한다.
```

3. 아래의 주석과 같이 동작하는 클래스를 선언하고 정의하라.

```
Base b1(2), b2(10);
b1.print(); // 2
b2.print(); // 10
for (int i = 0; i < 5; i++) {
   b1.setN(i, (i+1) * 20);
   b2.setN(i, (i+1) * 10);
}
b1.printData();  // 20 40
b2.printData();  // 10 20 30 40 50 0 0 0 0 0
Derived d(5); // Derived는 Base의 파생 클래스
d.print(); // 5
d.printData(); // 0 0 0 0 0
for (int i = 0; i < 10; i++) {
   d.setN(i, (i + 1) * 3);
}
d.printData(); // 3 6 9 12 15
d.insert(99); // Base는 insert 함수를 가지지 않음
d.printData(); // 3 6 9 12 15 99
```

코드 및 설명:

```
#include <iostream>
#include <vector>
class Base { //클래스 Base 선언 및 정의
        int a; //멤버 변수 선언
protected:
        std::vector<int> N; //상속 클래스 Derived도 벡터 N에 접근할 수 있도록 protected로 설정
public:
        Base(int a) : a(a) { //생성자 선언 및 정의, 벡터 N이 크기가 a이고 엘리먼트가 0이도록
초기 설정
                std::vector<int> n(a, 0);
                N = n;
        };
        void print() { //a를 출력하는 함수 print 선언 및 정의
                std::cout << a << std::endl;
        }
        void setN(int index, int elem) {
        //벡터 N의 특정 인덱스의 엘리먼트를 수정하는 함수 setN 선언 및 정의 (입력받는 인덱스
인수가 N의 크기보다 작은 조건 하에 실행)
                if (index < N.size()) {
                        N[index] = elem;
                }
        }
        void printData() { //N의 엘리먼트를 출력하는 함수 printData 선언 및 정의
                for (int i = 0; i < N.size(); i++) {
```

```
                        std::cout << N[i] << ' ';
                }
                std::cout << std::endl;
        }
        friend class Drived; //상속 클래스 Derived도 벡터 N에 접근할 수 있도록 friend class로
설정
};
class Derived : public Base { //Base의 상속 클래스 Derived 선언 및 정의
public:
        Derived(int a) : Base(a) {}; //다른 멤버 변수는 필요없고, Base를 호출하여 생성자 선언
및 정의
        void insert(int elem) { //N에 새로운 엘리먼트를 추가하는 함수 insert를 선언 및 정의
                N.push_back(elem);
        }
};
```