OOP Lab

Name: **Soham Das**

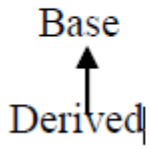
Section: **A1**

Roll No: **002311001004**

Assignment - 3

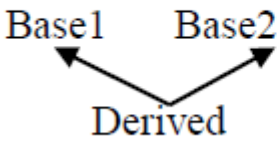
IT-UG2

28. Write empty class declarations for the following class hierarchy.




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

class Base
{public:
    Base()
    {
        cout<<"Base Constructor";
    }
};

class Derived : public Base
{ public:
    Derived()
    {
        cout << "Derived Constructor";
    }
};
int main()
{

    return 0;
}
```

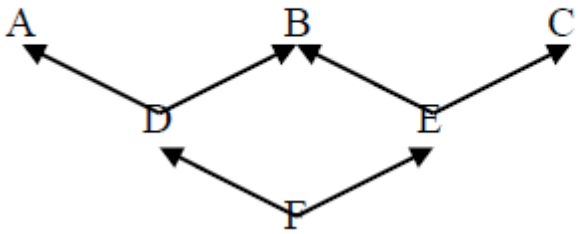
29. Write empty class declarations for the following class hierarchy.




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

class Base1
{
```

```
public:
    Base1()
    {
        cout << "Base1 Constructor";
    }
};
class Base2
{
public:
    Base2()
    {
        cout << "Base2 Constructor";
    }
};

class Derived : public Base1, public Base2
{
public:
    Derived()
    {
        cout << "Derived Constructor";
    }
};
int main()
{

    return 0;
}
```

30. Write empty class declarations for the following class hierarchy.




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

class A
{
public:
    A() { cout << "A constructor"; }
```

```
};

class B
{
public:
    B() { cout << "B constructor"; }
};

class C
{
public:
    C() { cout << "C constructor"; }
};

class D : public A, public B
{
public:
    D() { cout << "D constructor"; }
};

class E : public B, public C
{
public:
    E() { cout << "E constructor"; }
};

class F : public D, public E
{
public:
    F() { cout << "F constructor"; }
};

int main()
{
    return 0;
}
```

31. Write a class Person having data member name, age, height etc. Write proper constructors, methods to get/set them and a method printDetails() that prints all information of a person. Now write another class Student from Person and add data members roll, year of admission etc. Write constructors, methods to get/set them and a override printDetails(). Now create a Person and a Student object and call printDetails() function on them to display their information.

Now Create an array of pointers to Person and store addresses of two Persons and two Students. Call printDetails() on all elements (a loop may be used). Are you getting output which is supposed to come? Make printDetails() function virtual in the base class and check the result.

```
#include <iostream>
#include <string>

using namespace std;

class Person
{
protected:
    string name;
    int age;
    double height;

public:
    Person(string personName, int personAge, double personHeight)
    {
        name = personName;
        age = personAge;
        height = personHeight;
    }

    void printDetails()
    {
        cout << "Name: " << name << ", Age: " << age << ", Height: " <<
height << endl;
    }
};

class Student : public Person
{
private:
    int roll;
    int yearOfAdmission;

public:
    Student(string studentName, int studentAge, double studentHeight,
int studentRoll, int studentYearOfAdmission)
        : Person(studentName, studentAge, studentHeight)
    {
        roll = studentRoll;
        yearOfAdmission = studentYearOfAdmission;
    }
```

```
    void printDetails()
    {
        cout << "Name: " << name << ", Age: " << age << ", Height: " << height
             << ", Roll: " << roll << ", Year of Admission: " << yearOfAdmission << endl;
    }
};

int main()
{
    Person person("John Doe", 45, 5.9);
    Student student("Jane Smith", 20, 5.6, 101, 2022);

    person.printDetails();
    student.printDetails();

    Person *people[4];

    people[0] = new Person("Alice Brown", 50, 5.7);
    people[1] = new Person("Bob White", 55, 5.8);
    people[2] = new Student("Charlie Green", 22, 5.9, 102, 2021);
    people[3] = new Student("David Black", 21, 5.7, 103, 2020);

    for (int i = 0; i < 4; ++i)
    {
        people[i]->printDetails();
    }

    for (int i = 0; i < 4; ++i)
    {
        delete people[i];
    }

    return 0;
}
```

```
#include <iostream>
#include <string>

using namespace std;

class Person
{
```

```
protected:
    string name;
    int age;
    double height;

public:
    Person(string personName, int personAge, double personHeight)
    {
        name = personName;
        age = personAge;
        height = personHeight;
    }

    virtual void printDetails()
    {
        cout << "Name: " << name << ", Age: " << age << ", Height: " <<
height << endl;
    }

    virtual ~Person() {}
};

class Student : public Person
{
private:
    int roll;
    int yearOfAdmission;

public:
    Student(string studentName, int studentAge, double studentHeight,
int studentRoll, int studentYearOfAdmission)
        : Person(studentName, studentAge, studentHeight)
    {
        roll = studentRoll;
        yearOfAdmission = studentYearOfAdmission;
    }

    void printDetails()
    {
        cout << "Name: " << name << ", Age: " << age << ", Height: " <<
height
             << ", Roll: " << roll << ", Year of Admission: " <<
yearOfAdmission << endl;
    }
};
```

```
int main()
{
    Person person("John Doe", 45, 5.9);
    Student student("Jane Smith", 20, 5.6, 101, 2022);

    person.printDetails();
    student.printDetails();

    Person *people[4];

    people[0] = new Person("Alice Brown", 50, 5.7);
    people[1] = new Person("Bob White", 55, 5.8);
    people[2] = new Student("Charlie Green", 22, 5.9, 102, 2021);
    people[3] = new Student("David Black", 21, 5.7, 103, 2020);

    for (int i = 0; i < 4; ++i)
    {
        people[i]->printDetails();
    }

    for (int i = 0; i < 4; ++i)
    {
        delete people[i];
    }

    return 0;
}
```

32. Write a class Employee having data member name, salary etc. Write proper constructors, methods to get/set them and a virtual method printDetails() that prints all information of a person. Now write two classes Manager and Clerk from Employee. Add ‘type’ and ‘allowance’ in the manager and Clerk respectively. Write constructors, methods to get/set them and a override printDetails(). Now create a Manager and a Clerk object and call printDetails() function on them to display their information.

Now Create an array of pointers to Employee and store addresses of two Employee, two Managers and two Clerks. Call printDetails() on all elements (a loop may be used). Also find the total salary drawn by all employees.

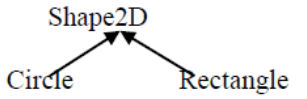
```
#include <iostream>
#include <string>
```

```cpp
using namespace std;

class Employee
{
protected:
    string name;
    double salary;

public:
    Employee(string empName, double empSalary)
    {
        name = empName;
        salary = empSalary;
    }

    virtual void printDetails()
    {
        cout << "Name: " << name << ", Salary: " << salary << endl;
    }

    double getSalary()
    {
        return salary;
    }

    virtual ~Employee() {}
};

class Manager : public Employee
{
private:
    string type;

public:
    Manager(string mgrName, double mgrSalary, string mgrType)
        : Employee(mgrName, mgrSalary)
    {
        type = mgrType;
    }

    void printDetails()
    {
        cout << "Name: " << name << ", Salary: " << salary
             << ", Type: " << type << endl;
    }
};
```

```
class Clerk : public Employee
{
private:
    double allowance;

public:
    Clerk(string clerkName, double clerkSalary, double clerkAllowance)
        : Employee(clerkName, clerkSalary)
    {
        allowance = clerkAllowance;
    }

    void printDetails()
    {
        cout << "Name: " << name << ", Salary: " << salary
             << ", Allowance: " << allowance << endl;
    }
};

int main()
{
    Manager mgr("John Doe", 75000, "Operations");
    Clerk clk("Jane Smith", 35000, 5000);

    mgr.printDetails();
    clk.printDetails();

    Employee *employees[6];

    employees[0] = new Employee("Alice Brown", 50000);
    employees[1] = new Employee("Bob White", 55000);
    employees[2] = new Manager("Charlie Green", 80000, "Sales");
    employees[3] = new Manager("David Black", 82000, "HR");
    employees[4] = new Clerk("Eve Blue", 36000, 4000);
    employees[5] = new Clerk("Frank Red", 37000, 4500);

    double totalSalary = 0;
    for (int i = 0; i < 6; ++i)
    {
        employees[i]->printDetails();
        totalSalary += employees[i]->getSalary();
    }

    cout << "Total Salary drawn by all employees: " << totalSalary <<
endl;
```

```
    for (int i = 0; i < 6; ++i)
    {
        delete employees[i];
    }

    return 0;
}
```

33. Write class definitions for the following class hierarchy



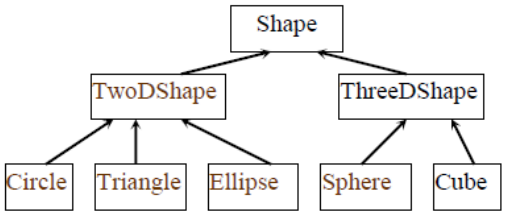
The Shape2D class represents two dimensional shapes that should have pure virtual functions area(), perimeter() etc. Implement these functions in Circle and Rectangle. Also write proper constructor(s) and other functions you think appropriate in the Circle and Rectangle class. Now create an array of 5 Shape2D pointers. Create 3 Circle and 2 Rectangles objects and place their addresses in that array. Use a loop to print area and perimeter of all shapes on this array.

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

using namespace std;

class Shape2D
{
public:
    virtual double area() = 0;
    virtual double perimeter() = 0;
    virtual ~Shape2D() {}
};

class Circle : public Shape2D
{
private:
    double radius;

public:
    Circle(double r)
    {
        radius = r;
    }

    double area()
    {
        return M_PI * radius * radius;
```

```
    }

    double perimeter()
    {
        return 2 * M_PI * radius;
    }
};

class Rectangle : public Shape2D
{
private:
    double length, width;

public:
    Rectangle(double l, double w)
    {
        length = l;
        width = w;
    }

    double area()
    {
        return length * width;
    }

    double perimeter()
    {
        return 2 * (length + width);
    }
};

int main()
{
    Shape2D *shapes[5];

    shapes[0] = new Circle(3.0);
    shapes[1] = new Circle(4.0);
    shapes[2] = new Circle(5.0);
    shapes[3] = new Rectangle(4.0, 6.0);
    shapes[4] = new Rectangle(5.0, 7.0);

    for (int i = 0; i < 5; ++i)
    {
        cout << "Shape " << i + 1 << ": " << endl;
        cout << "Area: " << shapes[i]->area() << endl;
        cout << "Perimeter: " << shapes[i]->perimeter() << endl
```

```
            << endl;
    }

    for (int i = 0; i < 5; ++i)
    {
        delete shapes[i];
    }

    return 0;
}
```

34. ***Implement*** the Shape hierarchy as shown in the figure. Each ***TwoDShape*** should contain function ***getArea*** to calculate the area of two-dimensional shape. Each ***ThreeDShape*** should have member functions ***getArea*** and ***getVolume*** to calculate the surface area and volume of the three-dimensional shape respectively. Create a program that uses Vector of Shape pointers to objects of each concrete class in the hierarchy. Now write a program that processes all the shapes in the Vector such that if the shape is a ***TwoDShape*** it prints name of shape and its area while it prints name of shape, its area and volume if the shape is a ***ThreeDShape***.




```
#include <bits/stdc++.h>
using namespace std;

class Shape
{
public:
    virtual string getName() = 0;
    virtual double getArea() = 0;

    virtual ~Shape() {}
};

class TwoDShape : public Shape
{
public:
    virtual ~TwoDShape() {}
};

class ThreeDShape : public Shape
{
public:
```

```
    virtual double getVolume() = 0;
    virtual ~ThreeDShape() {}
};

class Circle : public TwoDShape
{
private:
    double radius;

public:
    Circle(double r) : radius(r) {}

    string getName() override
    {
        return "Circle";
    }

    double getArea() override
    {
        return M_PI * radius * radius;
    }
};

class Triangle : public TwoDShape
{
private:
    double base, height;

public:
    Triangle(double b, double h) : base(b), height(h) {}

    string getName() override
    {
        return "Triangle";
    }

    double getArea() override
    {
        return 0.5 * base * height;
    }
};

class Ellipse : public TwoDShape
{
private:
    double majorAxis, minorAxis;
```

```
public:
    Ellipse(double a, double b) : majorAxis(a), minorAxis(b) {}

    string getName() override
    {
        return "Ellipse";
    }

    double getArea() override
    {
        return M_PI * majorAxis * minorAxis;
    }
};

class Sphere : public ThreeDShape
{
private:
    double radius;

public:
    Sphere(double r) : radius(r) {}

    string getName() override
    {
        return "Sphere";
    }

    double getArea() override
    {
        return 4 * M_PI * radius * radius;
    }

    double getVolume() override
    {
        return (4.0 / 3) * M_PI * pow(radius, 3);
    }
};

class Cube : public ThreeDShape
{
private:
    double side;

public:
    Cube(double s) : side(s) {}
```

```
    string getName() override
    {
        return "Cube";
    }

    double getArea() override
    {
        return 6 * side * side;
    }

    double getVolume() override
    {
        return pow(side, 3);
    }
};

int main()
{

    vector<Shape *> shapes;

    shapes.push_back(new Circle(3.0));
    shapes.push_back(new Triangle(3.0, 4.0));
    shapes.push_back(new Ellipse(5.0, 2.0));
    shapes.push_back(new Sphere(4.0));
    shapes.push_back(new Cube(2.0));

    for (Shape *shape : shapes)
    {
        cout << "Shape: " << shape->getName() << endl;
        cout << "Area: " << shape->getArea() << endl;

        ThreeDShape *threeDShape = dynamic_cast<ThreeDShape *>(shape);
        if (threeDShape)
        {
            cout << "Volume: " << threeDShape->getVolume() << endl;
        }

        cout << "--------------------------" << endl;
    }

    for (Shape *shape : shapes)
    {
        delete shape;
    }
```

```
    return 0;
}
```

35. Write a program to illustrate the role of virtual destructor.

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

class Base
{
public:
    Base()
    {
        cout << "Base Constructor" << endl;
    }

    virtual ~Base()
    {
        cout << "Base Destructor" << endl;
    }
};

class Derived : public Base
{
private:
    int x;

public:
    Derived()
    {
        cout << "Derived Constructor" << endl;
    }

    ~Derived()
    {
        cout << "Derived Destructor" << endl;
    }
};

int main()
{
    Base *ptr = new Derived();
    delete ptr;
```

```
    return 0;
}
```