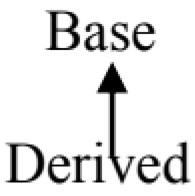
Assignment 3

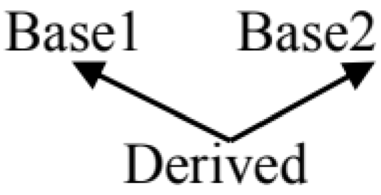
Name: Tanish Majumdar Roll No.: 002311001077

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



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

class Base
{
};

class Derived : public Base
{
};

int main()
{
    Derived d;

    return 0;
}
```

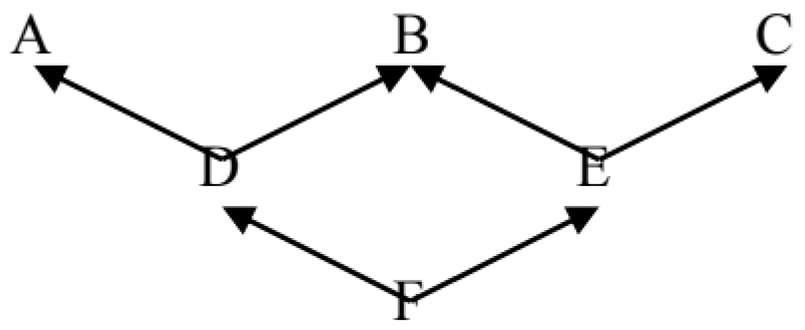
29. Write empty class declarations for the following class hierarchy.



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

```
class Base1
{
};

class Base2
{
};

class Derived : public Base1, public Base2
{
};

int main()
{
    Derived d;

    return 0;
}
```

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



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

class A
{
};

class B
{
};

class C
{
};

class D : public A, public virtual B
```

```
{
};

class E : public C, public virtual B
{
};

class F : public D, public E
{
};

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>
using namespace std;

class Person
{
    string name;
    int age;
    int height;

public:
    Person(string name, int age, int height)
```

```cpp
    {
        this->name = name;
        this->age = age;
        this->height = height;
    }
    void virtual printDetails()
    {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
        cout << "Height: " << height << endl;
    }
};

class Student : public Person
{
    int rollNo;
    int yearOfAdmission;

public:
    Student(string name, int age, int height, int rollNo, int
yearOfAdmission) : Person(name, age, height)
    {
        this->rollNo = rollNo;
        this->yearOfAdmission = yearOfAdmission;
    }
    void printDetails()
    {
        Person::printDetails();
        cout << "Roll No: " << rollNo << endl;
        cout << "Year of Admission: " << yearOfAdmission << endl;
    }
};

int main()
{
    Student s("John Doe", 20, 180, 12345, 2020);
    Person p("Jane Doe", 25, 160);

    s.printDetails();

    Person *people[4];
    people[0] = new Person("Alice", 30, 170);
    people[1] = new Student("Bob", 22, 175, 54321, 2018);
    people[2] = new Person("Charlie", 35, 165);
    people[3] = new Student("David", 24, 180, 67890, 2019);

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

    return 0;
}
```

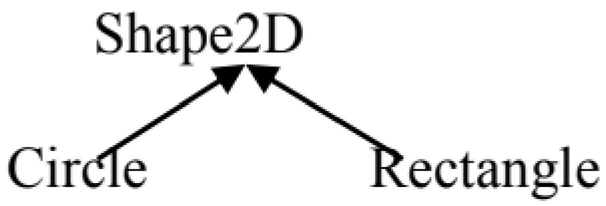
32.Write a class `Employee` having data member name, salary etc.

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>
using namespace std;

class Employee
{
    string name;
    int salary;

public:
    Employee(string name, int salary)
    {
        this->name = name;
        this->salary = salary;
    }
    int getSalary()
    {
        return salary;
    }
    void virtual printDetails()
    {
        cout << "Name: " << name << endl;
        cout << "Salary: " << salary << endl;
    }
};
```

```cpp
class Manager : public Employee
{
    int type;
    int allowance;

public:
    Manager(string name, int salary, int type, int allowance) :
Employee(name, salary)
    {
        this->type = type;
        this->allowance = allowance;
    }
    void printDetails()
    {
        Employee::printDetails();
        cout << "Type: " << type << endl;
        cout << "Allowance: " << allowance << endl;
    }
};

class Clerk : public Employee
{
    int type;
    int allowance;

public:
    Clerk(string name, int salary, int type, int allowance) :
Employee(name, salary)
    {
        this->type = type;
        this->allowance = allowance;
    }
    void printDetails()
    {
        Employee::printDetails();
        cout << "Type: " << type << endl;
        cout << "Allowance: " << allowance << endl;
    }
};

int main()
{
    Employee *e[6];
    e[0] = new Employee("Alice", 30000);
    e[1] = new Manager("Bob", 50000, 1, 10000);
    e[2] = new Clerk("Charlie", 20000, 2, 5000);
    e[3] = new Employee("David", 40000);
    e[4] = new Manager("Eve", 60000, 1, 15000);
    e[5] = new Clerk("Frank", 25000, 2, 6000);
    int totSal = 0;
    for (int i = 0; i < 6; i++)
    {
        e[i]->printDetails();
        cout << endl;
        totSal += e[i]->getSalary();
    }

    cout << "Total Salary: " << totSal << endl;

    return 0;
}
```

33. Write class definitions for the following class hierarchy

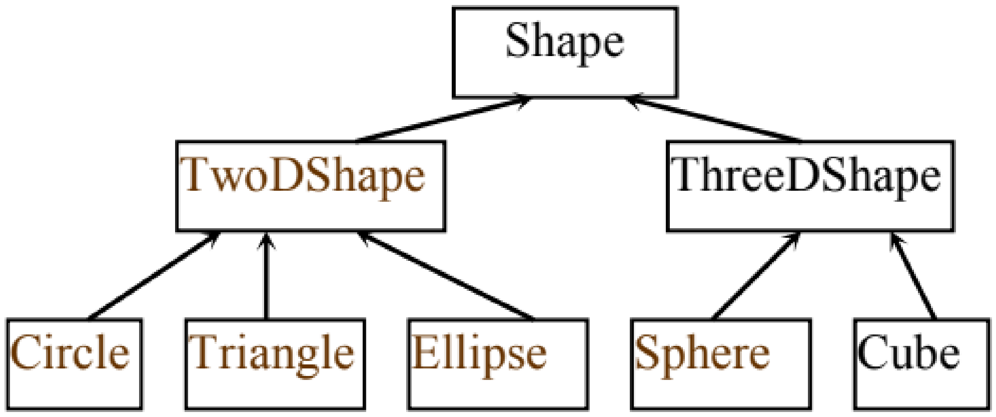


The `Shape2D` class represents two dimensional shapes that should have pure virtual functions `area(), perimeter()` etc. Implement these functions in Circle and Rectangle. Also write proper constructor(s) and other functions you think appropriate in the Circle and Rectangle class. Now create an array 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>
using namespace std;

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

class Circle : public Shape2D
{
    int radius;
```

```
public:
    Circle(int radius)
    {
        this->radius = radius;
    }
    void area()
    {
        cout << "Area of Circle: " << 3.14 * radius * radius << endl;
    }
    void perimeter()
    {
        cout << "Perimeter of Circle: " << 2 * 3.14 * radius << endl;
    }
};

class Rectangle : public Shape2D
{
    int length, breadth;

public:
    Rectangle(int length, int breadth)
    {
        this->length = length;
        this->breadth = breadth;
    }
    void area()
    {
        cout << "Area of Rectangle: " << length * breadth << endl;
    }
    void perimeter()
    {
        cout << "Perimeter of Rectangle: " << 2 * (length + breadth) <<
endl;
    }
};

int main()
{
    Shape2D *s[5];
    s[0] = new Circle(5);
    s[1] = new Rectangle(5, 10);
    s[2] = new Circle(10);
    s[3] = new Rectangle(10, 20);
    s[4] = new Circle(20);

    for (int i = 0; i < 5; i++)
    {
        s[i]->area();
        s[i]->perimeter();
    }

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

class Shape
{
public:
    virtual int is3D() = 0;
};

class Shape2D : public Shape
{
```

```
public:
    virtual void area() = 0;
    int is3D()
    {
        return 0;
    }
};

class Shape3D : public Shape
{
public:
    int is3D()
    {
        return 1;
    }
    virtual void volume() = 0;
    virtual void area() = 0;
};

class Circle : public Shape2D
{
    int radius;

public:
    Circle(int radius)
    {
        this->radius = radius;
    }
    void area()
    {
        cout << "Area of Circle: " << 3.14 * radius * radius << endl;
    }
};

class Triangle : public Shape2D
{
    int a, b, c;

public:
    Triangle(int a, int b, int c)
    {
        this->a = a;
        this->b = b;
        this->c = c;
    }
    void area()
    {
        int s = (a + b + c) / 2;
        cout << "Area of Triangle: " << sqrt(s * (s - a) * (s - b) * (s -
c)) << endl;
```

```
    }
};

class Ellipse : public Shape2D
{
    int major, minor;

public:
    Ellipse(int major, int minor)
    {
        this->major = major;
        this->minor = minor;
    }
    void area()
    {
        cout << "Area of Ellipse: " << 3.14 * major * minor << endl;
    }
};

class Sphere : public Shape3D
{
    int radius;

public:
    Sphere(int radius)
    {
        this->radius = radius;
    }
    void area()
    {
        cout << "Area of Sphere: " << 4 * 3.14 * radius * radius << endl;
    }
    void volume()
    {
        cout << "Volume of Sphere: " << 4 / 3 * 3.14 * radius * radius *
radius << endl;
    }
};

class Cube : public Shape3D
{
    int side;

public:
    Cube(int side)
    {
        this->side = side;
    }
    void area()
    {
        cout << "Area of Cube: " << 6 * side * side << endl;
    }
    void volume()
    {
        cout << "Volume of Cube: " << side * side * side << endl;
    }
};

int main()
{
    vector<Shape *> s;
    s.push_back(new Circle(5));
    s.push_back(new Triangle(3, 4, 5));
    s.push_back(new Ellipse(5, 10));
    s.push_back(new Sphere(5));
    s.push_back(new Cube(5));

    for (int i = 0; i < s.size(); i++)
    {
        if (s[i]->is3D())
        {
            ((Shape3D *)s[i])->area();
            ((Shape3D *)s[i])->volume();
        }
        else
        {
            ((Shape2D *)s[i])->area();
        }
    }

    return 0;
}
```

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

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

class Artist
{
    string name;
    string topSong;

public:
    Artist(string name, string topSong)
    {
        this->name = name;
        this->topSong = topSong;
    }
    void display()
    {
        cout << "Name: " << name << endl;
        cout << "Top Song: " << topSong << endl;
    }
    virtual ~Artist() = 0;
};

Artist::~Artist()
{
    cout << "Artist Destructor" << endl;
}

class LanaDelRey : public Artist
{
    string album;

public:
    LanaDelRey(string name, string topSong, string album) : Artist(name,
topSong)
    {
        this->album = album;
    }
    void display()
    {
        Artist::display();
        cout << "Album: " << album << endl;
    }
};

int main()
{
    LanaDelRey lana("Lana Del Rey", "Summertime Sadness", "Born to Die");
    lana.display();

     Artist *a = new LanaDelRey("Lana Del Rey", "Summertime Sadness",
"Born to Die");
     a->display();
     delete a;

    return 0;
}
```