

POLYMORPHISM

1. Write a program to illustrate virtual function by creating a class shape with functions to find the area of the shapes and display the names of the shapes and other essential components of the class. Create derived classes circle, rectangle, and trapezoid each having overriding functions area() and display().

PROGRAM

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

class Shape {
public:
    virtual void area() = 0;
    virtual void display() = 0;

    virtual ~Shape() { }
};

class Rectangle: public Shape {
private:
    int l, b, a;
public:
    void area() {
        cout<<"Enter length and breadth of rectangle: ";
        cin>>l>>b;
        a = l*b;
    }

    void display() {
        cout<<"Area of rectangle = "<<a<<endl;
    }
};

class Triangle: public Shape {
private:
    float b, h, a;
public:
    void area() {
        cout<<"Enter base and height of triangle: ";
        cin>>b>>h;
        a = 0.5f * b * h;
    }
}
```

```

        void display() {
            cout<<"Area of triangle = "<<a<<endl;
        }
};

class Trapeziod: public Shape {
private:
    int b1, b2, h;
    float a;
public:
    void area() {
        cout<<"Enter base1, base2, and height of a trapeziod: ";
        cin>>b1>>b2>>h;
        a = 0.5f * (b1 + b2) * h;
    }

    void display() {
        cout<<"Area of trapeziod = "<<a<<endl;
    }
};

int main() {
    Rectangle r;
    Triangle t;
    Trapeziod tp;

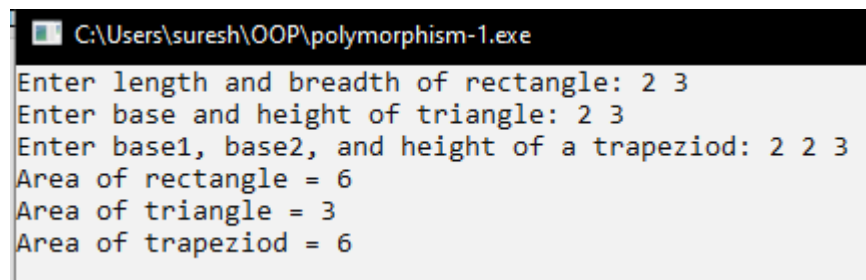
    r.area();
    t.area();
    tp.area();

    r.display();
    t.display();
    tp.display();

    return 0;
}

```

OUTPUT



```

C:\Users\suresh\OOP\polymorphism-1.exe
Enter length and breadth of rectangle: 2 3
Enter base and height of triangle: 2 3
Enter base1, base2, and height of a trapeziod: 2 2 3
Area of rectangle = 6
Area of triangle = 3
Area of trapeziod = 6

```

- 2. Write a program with an abstract class Student and create derive classes Engineering, Medicine and Science from base class Student. Create the objects of the derived classes and process them and access them using an array of pointers of type base class Student.**

PROGRAM

```
#include<iostream>

using namespace std;

class Student {
    protected:
        int roll;
        string name;
    public:
        virtual void read_data() = 0;
        virtual void display() = 0;
};

class Engineering: public Student {
    protected:
        string faculty = "Engineering";
    public:
        void read_data() {
            cout<<"Enter roll number and name of a student: ";
            cin>>roll>>name;
        }

        void display() {
            cout<<"Roll number: "<<roll<<endl;
            cout<<"Name: "<<name<<endl;
            cout<<"Faculty: "<<faculty;
        }
}
```

```
};
```

```
class Medicine: public Student {  
    protected:  
        string faculty = "Medicine";  
    public:  
        void read_data() {  
            cout<<"Enter roll number and name of a student: ";  
            cin>>roll>>name;  
        }  
  
        void display() {  
            cout<<"Roll number: "<<roll<<endl;  
            cout<<"Name: "<<name<<endl;  
            cout<<"Faculty: "<<faculty;  
        }  
};
```

```
class Science: public Student {  
    protected:  
        string faculty = "Science";  
    public:  
        void read_data() {  
            cout<<"Enter roll number and name of a student: ";  
            cin>>roll>>name;  
        }  
  
        void display() {  
            cout<<"Roll number: "<<roll<<endl;  
            cout<<"Name: "<<name<<endl;
```

```

        cout<<"Faculty: "<<faculty;

    }

};

int main() {

    Student *sptr[3];
    Engineering e;
    Science s;
    Medicine m;

    sptr[0] = &e;
    sptr[1] = &s;
    sptr[2] = &m;

    cout<<"Enter information of engineering student: \n";
    sptr[0]->read_data();

    cout<<"Enter information of science student: \n";
    sptr[1]->read_data();

    cout<<"Enter information of medicine student: \n";
    sptr[2]->read_data();

    cout<<"\nInformation of engineering student: \n";
    sptr[0]->display();

    cout<<"\nInformation of science student: \n";
    sptr[1]->display();

```

```

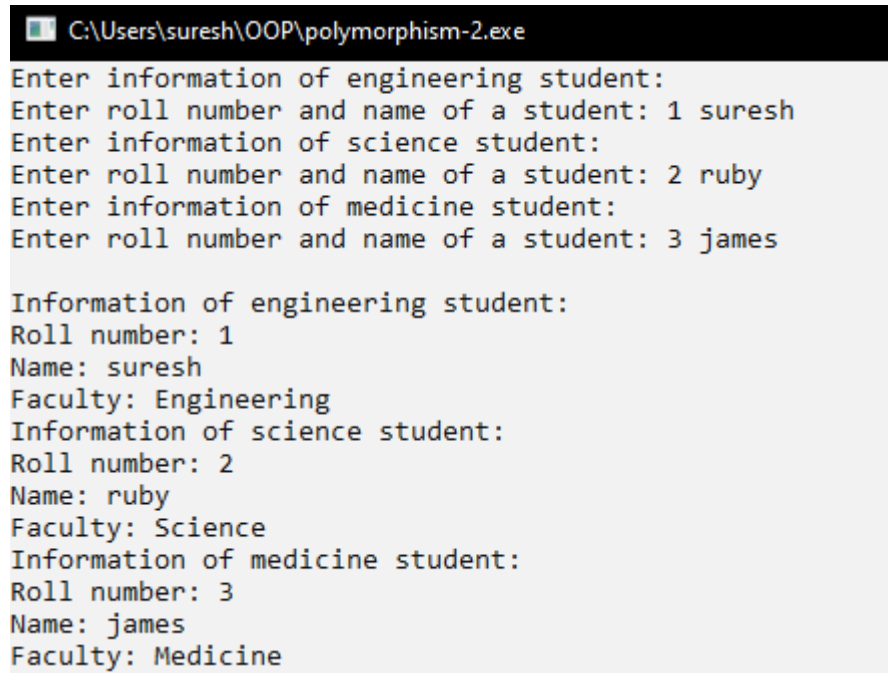
        cout<<"\nInformation of medicine student: \n";

        sptr[2]->display();

        return 0;
}

```

OUTPUT



```

C:\Users\suresh\OOP\polymorphism-2.exe
Enter information of engineering student:
Enter roll number and name of a student: 1 suresh
Enter information of science student:
Enter roll number and name of a student: 2 ruby
Enter information of medicine student:
Enter roll number and name of a student: 3 james

Information of engineering student:
Roll number: 1
Name: suresh
Faculty: Engineering
Information of science student:
Roll number: 2
Name: ruby
Faculty: Science
Information of medicine student:
Roll number: 3
Name: james
Faculty: Medicine

```

3. Demonstrate Deletion of Child Object using Base Class Pointer without using a Virtual Destructor.

PROGRAM

```

#include <iostream>

using namespace std;

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

```

```

~Base() {
    cout << "Base class destructor" << endl;
}
};

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

    ~Derived() {
        cout << "Derived class destructor" << endl;
    }
};

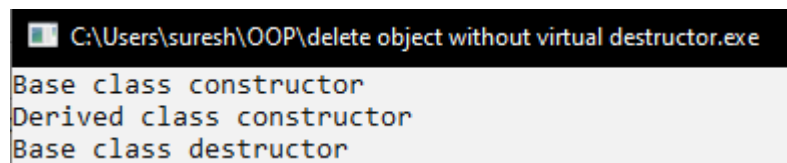
int main() {
    Base* ptr = new Derived();

    delete ptr;

    return 0;
}

```

OUTPUT



```

C:\Users\suresh\OOP\delete object without virtual destructor.exe
Base class constructor
Derived class constructor
Base class destructor

```

4. Demonstrate Deletion of Child Object using Base Class Pointer using a Virtual Destructor.

PROGRAM

```
#include <iostream>

using namespace std;

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

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

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

    ~Derived() {
        cout << "Derived class destructor" << endl;
    }
};
```

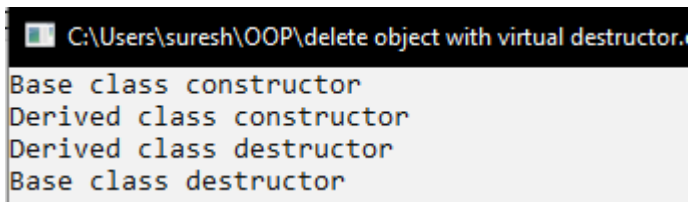


```
int main() {
    Base* ptr = new Derived();

    delete ptr;

    return 0;
}
```

OUTPUT



```
C:\Users\suresh\OOP\delete object with virtual destructor...
Base class constructor
Derived class constructor
Derived class destructor
Base class destructor
```

5. Create a polymorphic class **Vehicle** and create other derived classes **Bus**, **Car**, and **Bike** from **Vehicle**. Illustrate RTTI by the use of `dynamic_cast` and `typeid` operators in this program.

PROGRAM

```
#include <iostream>
#include <typeinfo>

class Vehicle {
public:
    virtual ~Vehicle() {}
    virtual void start() const { std::cout << "Vehicle starting..." << std::endl; }
    virtual void stop() const { std::cout << "Vehicle stopping..." << std::endl; }
};
```

```
class Bus : public Vehicle {
public:
    void start() const override { std::cout << "Bus starting..." << std::endl; }
    void stop() const override { std::cout << "Bus stopping..." << std::endl; }
};
```

```
class Car : public Vehicle {
public:
    void start() const override { std::cout << "Car starting..." << std::endl; }
    void stop() const override { std::cout << "Car stopping..." << std::endl; }
};
```

```
class Bike : public Vehicle {
public:
    void start() const override { std::cout << "Bike starting..." << std::endl; }
    void stop() const override { std::cout << "Bike stopping..." << std::endl; }
};
```

```
void demonstrateRTTI(Vehicle* v) {
    std::cout << "Type of Vehicle: " << typeid(*v).name() << std::endl;
```

```
    if (Bus* b = dynamic_cast<Bus*>(v)) {
        std::cout << "The vehicle is a Bus." << std::endl;
        b->start();
        b->stop();
    } else if (Car* c = dynamic_cast<Car*>(v)) {
        std::cout << "The vehicle is a Car." << std::endl;
        c->start();
        c->stop();
    } else if (Bike* b = dynamic_cast<Bike*>(v)) {
```

```

        std::cout << "The vehicle is a Bike." << std::endl;
        b->start();
        b->stop();
    } else {
        std::cout << "Unknown vehicle type." << std::endl;
    }
}

int main() {
    Bus myBus;
    Car myCar;
    Bike myBike;

    Vehicle* v1 = &myBus;
    Vehicle* v2 = &myCar;
    Vehicle* v3 = &myBike;

    std::cout << "Demonstrating RTTI for Bus:" << std::endl;
    demonstrateRTTI(v1);


    std::cout << "\nDemonstrating RTTI for Car:" << std::endl;
    demonstrateRTTI(v2);

    std::cout << "\nDemonstrating RTTI for Bike:" << std::endl;
    demonstrateRTTI(v3);

    return 0;
}

```

OUTPUT

 C:\Users\suresh\OOP\polymorphism-3.exe

Demonstrating RTTI for Bus:

Type of Vehicle: 3Bus

The vehicle is a Bus.

Bus starting...

Bus stopping...

Demonstrating RTTI for Car:

Type of Vehicle: 3Car

The vehicle is a Car.

Car starting...

Car stopping...

Demonstrating RTTI for Bike:

Type of Vehicle: 4Bike

The vehicle is a Bike.

Bike starting...

Bike stopping...