Name	ADITI RAO
UID no.	202220003
Experiment No.	7

AIM:	**Program on Polymorphism: Implement a Program to demonstrate method overriding**

Program 1

PROBLEM STATEMENT:	Consider a class Product with data members barcode and name of the product. Create the appropriate constructor and write getter methods for the individual data members. and write two virtual methods, scanner() and printer(). Derive 2 classes from Product, 1st class is PrepackedFood and 2nd class is FreshFood. the PrepackedFood class should contain the unit price and the FreshFood class should contain a weight and a price per kilo as data members. Override the methods scanner and printer in the derived classes. (These methods will simply output product data on screen or read the data of a product from the keyboard depending upon whether it is Prepacked or FreshFood) In main, create a base class pointer and point it to the appropriate derived class objects to demonstrate runtime polymorphism.
ALGORITHM:	class Product protected: barcode: string nameProduct: string public: Product() barcode = "" nameProduct = "" Product(barcode: string, nameProduct: string) this.barcode = barcode this.nameProduct = nameProduct

```
        getBarcode()
            return barcode

        getNameProduct()
            return nameProduct

        scanner()  // Abstract method
            // To be implemented by derived classes

        printer()  // Abstract method
            // To be implemented by derived classes


class PrepackedFood extends Product
    protected:
        unitPrice: double

    public:
        PrepackedFood(barcode: string, name: string, unitPrice: double)
            super(barcode, name)
            this.unitPrice = unitPrice

        scanner()
            print "Scanning Prepacked Food: " + nameProduct + " (Barcode: "
+ barcode + ")"
            print "Unit Price: " + unitPrice + "Rs."

        printer()
            print "Your Order Item: " + nameProduct + " (Barcode: " +
barcode + ")"
            print "Unit Price: " + unitPrice + "Rs."


class FreshFood extends Product
    protected:
        weight: double
        pricePerKg: double
```

```
    public:
        FreshFood(barcode: string, name: string, weight: double, pricePerKg:
double)
            super(barcode, name)
            this.weight = weight
            this.pricePerKg = pricePerKg

        scanner()
            print "Scanning Fresh Food: " + nameProduct + " (Barcode: " +
barcode + ")"
            print "Weight: " + weight + "kg"
            print "Price per Kg: " + pricePerKg + "Rs."

        printer()
            totalPrice = weight * pricePerKg
            print "Your Order Item: " + nameProduct + " (Barcode: " +
barcode + ")"
            print "Weight: " + weight + "g"
            print "Total Price: " + totalPrice + "Rs."


main()
    productObj: Product pointer

    productObj = new PrepackedFood("123456789", "Shin Ramen", 40)
    productObj->scanner()
    productObj->printer()

    productObj = new FreshFood("987654321", "Mushrooms", 0.25, 230)
    productObj->scanner()
    productObj->printer()
```

PROGRAM:	#include <iostream>
	using namespace std;
	
	class Product
	{
	protected:
	  string barcode;
	  string nameProduct;
	
	public:
	  Product() : barcode(""), nameProduct("") {}
	  Product(string barcode, string nameProduct) : barcode(barcode), nameProduct(nameProduct) {}
	
	  string getBarcode()
	  {
	  return barcode;
	  }
	
	  string getNameProduct()
	  {
	  return nameProduct;
	  }
	
	  virtual void scanner() = 0; /*Pure virtual function. Error when not written: undefined reference to `vtable for Product'
	  collect2.exe: error: ld returned 1 exit status*/
	  virtual void printer() = 0;
	};
	
	class PrepackedFood : public Product
	{
	protected:
	  double unitPrice;
	
	public:
	  // PrepackedFood() : Product("", ""), unitPrice(0) {}
	  PrepackedFood(string barcode, string name, double unitPrice) :

```
Product(barcode, name), unitPrice(unitPrice) {}

    void scanner()
    {
        cout << "Scanning Prepacked Food: " << nameProduct << " (Barcode: " <<
barcode << ")" << endl;
        cout << "Unit Price: " << unitPrice << "Rs." << endl;
    }

    void printer()
    {
        cout << "Your Order Item: " << nameProduct << " (Barcode: " << barcode
<< ")" << endl;
        cout << "Unit Price: " << unitPrice << "Rs." << endl;
    }
};

class FreshFood : public Product
{
protected:
    double weight;
    double pricePerKg;

public:
    // FreshFood() : Product("", ""), weight(0), pricePerKg(0) {}
    FreshFood(string barcode, string name, double weight, double pricePerKg) :
Product(barcode, name), weight(weight), pricePerKg(pricePerKg) {}

    void scanner()
    {
        cout << "Scanning Fresh Food: " << nameProduct << " (Barcode: " <<
barcode << ")" << endl;
        cout << "Weight: " << weight << "kg" << endl;
        cout << "Price per Kg: " << pricePerKg << "Rs." << endl;
    }

    void printer()
    {
```

```
        double totalPrice = weight * pricePerKg;
        cout << "Your Order Item: " << nameProduct << " (Barcode: " << barcode << ")" << endl;
        cout << "Weight: " << weight << "g" << endl;
        cout << "Total Price: " << totalPrice << "Rs." << endl;
    }
};

int main()
{
    Product *productObj;

    productObj = new PrepackedFood("123456789", "Shin Ramen", 40);
    productObj->scanner();
    productObj->printer();

    productObj = new FreshFood("987654321", "Mushrooms", 0.25, 230);
    productObj->scanner();
    productObj->printer();

    return 0;
}
```

RESULT:

```
PS C:\Tesseract\Under Graduate Engineering Degree\SPIT\Seme
PL\Experiment 7\" ; if ($?) { g++ Program1_Product.cpp -o P
Scanning Prepacked Food: Shin Ramen (Barcode: 123456789)
Unit Price: 40Rs.
Your Order Item: Shin Ramen (Barcode: 123456789)
Unit Price: 40Rs.
Scanning Fresh Food: Mushrooms (Barcode: 987654321)
Weight: 0.25kg
Price per Kg: 230Rs.
Your Order Item: Mushrooms (Barcode: 987654321)
Weight: 0.25g
Total Price: 57.5Rs.
```

Program 2

PROBLEM STATEMENT: Create class person with attributes phone_number, name and a method read()for getting and setting the name and phone_number. Include a method show() to

	display the phone_number and name. Derive class student from person with attributes roll_number ,course and method read to override that in base class person. In this read() method give a call to the base class read() and ask for setting roll_number and course. Here also include a method show() which initially calls the base class method show() and then displays the roll_number and course.
ALGORITHM:	class Person protected: phone_number: integer name: string public: method read() display "Enter name: " read name display "Enter phone number: " read phone_number method show() display "Name: " + name display "Phone number: " + phone_number class Student extends Person private: roll_number: integer course: string public: method read() call base class read() method display "Enter roll number: " read roll_number display "Enter course: " read course method show() call base class show() method display "Roll number: " + roll_number display "Course: " + course

	main() s: Student object create s as new Student call s.read() method call s.show() method

PROGRAM:	#include <iostream>
	#include <string>
	using namespace std;
	
	class person
	{
	protected:
	int phone_number;
	string name;
	
	public:
	void read()
	{
	cout << "Enter name: ";
	getline(cin, name);
	cout << "Enter phone number: ";
	cin >> phone_number;
	cin.ignore();
	}
	
	void show()
	{
	cout << "Name: " << name << endl;
	cout << "Phone number: " << phone_number << endl;
	}
	};
	
	class student : public person
	{
	private:
	int roll_number;
	string course;

```
public:
    void read()
    {
        person::read();
        cout << "Enter roll number: ";
        cin >> roll_number;
        cin.ignore(); // Ignore the newline character
        cout << "Enter course: ";
        getline(cin, course);
    }

    void show()
    {
        person::show();
        cout << "Roll number: " << roll_number << endl;
        cout << "Course: " << course << endl;
    }
};

int main()
{
    student s;
    s.read();
    s.show();
    return 0;
}
```

RESULT:

```
PS C:\Tesseract\Under Graduate Engine
PL\Experiment 7\" ; if ($?) { g++ Pro
Enter name: Tom Riddle
Enter phone number: 293235432
Enter roll number: 21
Enter course: Mathematics-1
Name: Tom Riddle
Phone number: 293235432
Roll number: 21
Course: Mathematics-1
PS C:\Tesseract\Under Graduate Engine
```

Program 3	
PROBLEM STATEMENT:	Add a member function to the Rectangle class that computes the area of a Rectangle (length multiplied by width). Add a member function to Block that has the same name, but overrides the computation with a volume calculation (length by width by height). Write a main()function that demonstrates the classes. Save the file as RectangleAndBlock2.cpp

ALGORITHM:	class Rectangle
	  protected:
	  length: double
	  width: double
	
	  public:
	  constructor(length, width)
	  set this.length to length
	  set this.width to width
	
	  method computeArea()
	  return length multiplied by width
	
	class Block inherits Rectangle
	  protected:
	  height: double
	
	public:
	constructor(length, width, height)
	  call base class constructor with length and width
	  set this.height to height
	
	  method computeVolume()
	  return length multiplied by width multiplied by height
	
	main()
	  create rectangle object with length 5.0 and width 3.0
	  display "Rectangle Area: " concatenated with rectangle.computeArea()
	
	  create block object with length 4.0, width 2.0, and height 3.0

	display "Block Volume: " concatenated with block.computeVolume()

PROGRAM:	#include <iostream>
	
	using namespace std;
	
	class Rectangle
	{
	protected:
	  double length;
	  double width;
	
	public:
	  Rectangle(double length, double width)
	  {
	  this->length = length;
	  this->width = width;
	  }
	
	  double computeArea()
	  {
	  return length * width;
	  }
	};
	
	class Block : public Rectangle
	{
	private:
	  double height;
	
	public:
	  Block(double length, double width, double height) : Rectangle(length, width)
	  {
	  this->height = height;
	  }
	
	  double computeVolume()
	  {
	  return length * width * height;

```
    }
};

int main()
{
    Rectangle rectangle(5.0, 3.0);
    cout << "Rectangle Area: " << rectangle.computeArea() << endl;

    Block block(4.0, 2.0, 3.0);
    cout << "Block Volume: " << block.computeVolume() << endl;

    return 0;
}
```

RESULT:

```
PS C:\Tesseract\Under Graduate Engineering Degree\SPIT\Semester II\PSOOPL\Experim
PL\Experiment 7\" ; if ($?) { g++ RectangleAndBlock2.cpp -o RectangleAndBlock2 }
Rectangle Area: 15
Block Volume: 24
```

CONCLUSION:

In the programs demonstrating method overriding, we explored the concept of polymorphism in object-oriented programming. By using inheritance and overriding methods in derived classes, we were able to achieve dynamic polymorphism, where different objects of related classes can exhibit different behaviors based on their specific implementations of overridden methods. This flexibility and extensibility offered by polymorphism enhance code reusability and maintainability, making it a powerful feature in object-oriented programming languages. Understanding and effectively utilizing method overriding and polymorphism can greatly improve the design and functionality of object-oriented systems.