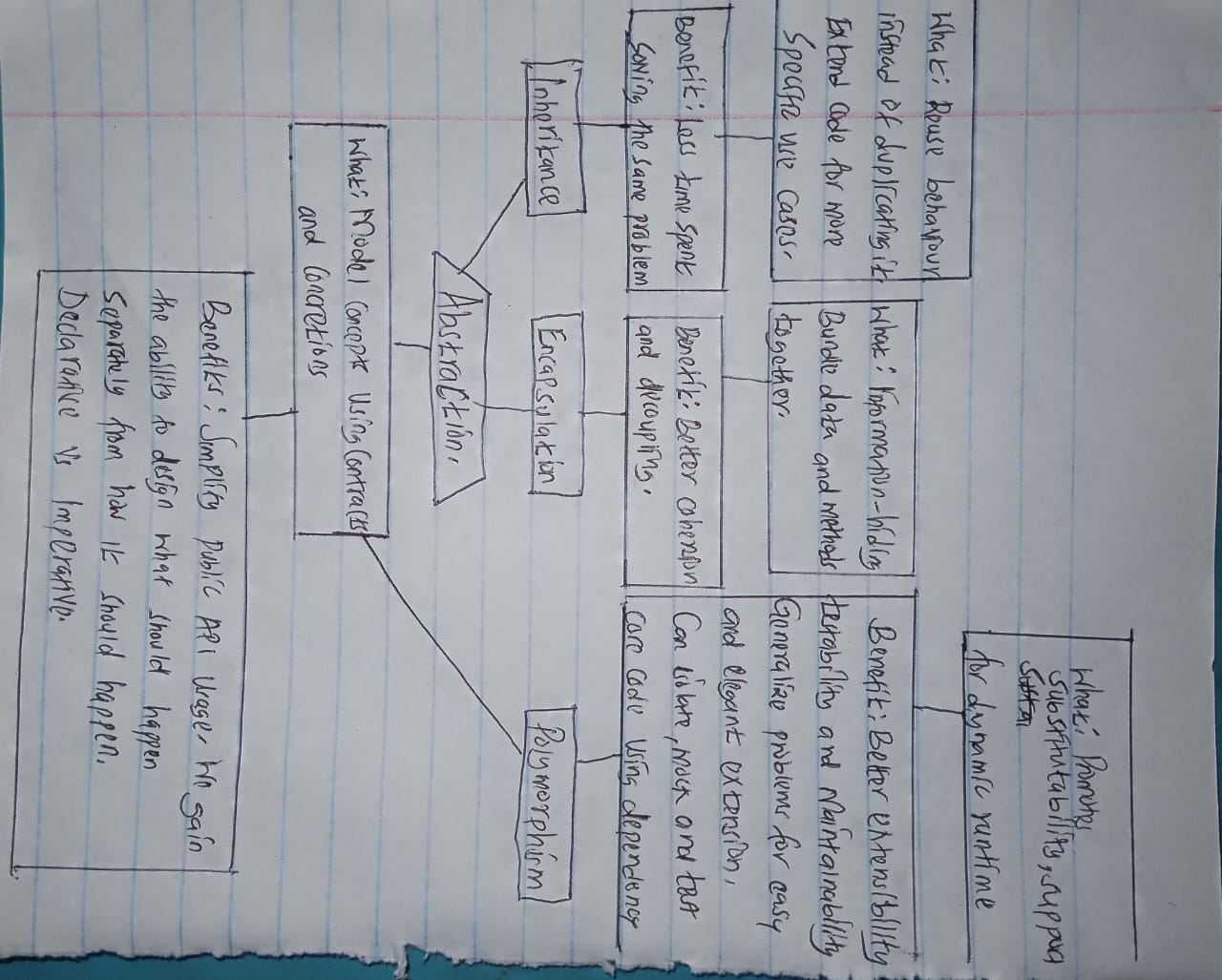
OOP Cat.

1.



2.UML: is an object modelling for software modelling and designing.

3. OOAD: is aprocess for analyzing and designing software applications. It focuses on the objects that make up the application and their interactions with each other.

OAD: is a process for analyzing and designing software applications that use non object oriented programming. It focuses on the data structure and algorithms used to create the application. The process involves breaking down the application into data structures and algorithms and describing their behaviour and relationship.

4. Main goal of UML: To provide a standard way to visualize the design of a system: UML provides a a standard visual language for software design, making it easier for software engineers to communicate their designs to stakeholders.

To enable efficient development processes: UML provides a common language for software engineers to use when discussing and designing a system. This help ensure that everyone involved in the development process is on the same page.

5. 1. Improved code reusability: Object oriented programming allows developers to reuse code, which in turn saves time and resources. By breaking down complex problems into smaller chunks, developers can easily reuse code and create efficient systems that are easier to maintain.

2. Improved security: It provides: Object oriented programming provides an extra layer of security by allowing developers to restrict access to certain data and methods. By encapsulating data and methods, developers can control who can access and modify data, which in turn improves security.

3. Improved scalability: Object oriented programming allows developers to easily extend and modify existing systems. By breaking down complex problems into smaller chunks, developers can quickly and easily add new and functionality to existing systems making them more scalable and easier to maintain.

6. 1. Constructor is a special type of method in which is used to initialize an object. It is called an object is created and has the same name as the class. Constructors are used to initialize the state of an object.

Example:

Public class car{

Private string model;

Public car(string model) {

This.model=model;

}

Public string getmodel(){

Return model;

}

}

Public class main{

Public static void main (string args) {

Car car=new car(“Benz”);

System.out.printin(car.getmodel());

}

}

2. Object oriented programming is a programming paradigm that is based on the concept of objects, which contain data and methods that act on the data. An object is self contained entity that contains both data and code to manipulate the data. In java an object is an instance of a class.

EXAMPLE.

Public class Employee{

Private int id;

Private string name;

Private double salary;

Public Employee(int id,string name, double salary){

this.id=id;

this.name=name;

this.salary=salary;

}

Public string getName(){

Return id;

}

Public string getName(){

Return name;

}

Public double getSalary(){

Return salary;}

Public void setSalary(double salary){

**this.salary=salary;**

**}**

**Public string tostring(){**

**Return “Employee[id=”+”,namke=”=name+”salary+”+salary+”]”;**

**}**

**}**

**3. A destructor is a special type of method that is used in oop to clean up an object before it is destroyed. It is usually called when an object is no longer needed and being removed from the memory. A destructor is called automatically when an object is no longer referenced by any other object.**

**Example.**

**Public class Myclass{**

**Public Myclass(){**

**//constructor code**

**}**

**//destructor method**

**Protected void finalize() throws Throwable{**

**Try{**

**//cleanup code**

**}finally{**

**Super.finalize();**

**}**

**}**

**}**

**4. Polymorphism is a concept in oop that allows objects of different types to be created the same way. Polymorphism allows a single interface to be used to invoke different implementations of a given method.**

**Example:**

**Consider a class Animal, with 2 subclass cat and dog. Both Cat and Dog classes have a method calledmaknoise(). The makeNoise() method will have different implementations for each class. Polymorphism allows us to invoke the makenoise() method on an Animalobject and the correct implementation for the type of object will be used.**

**Example of java code:**

**Public class Animal{**

**Public void makenoise(){**

**System.out.printin(“Generic animal noise”);**

**}**

**}**

**Public class cat extends Animal{**

**Public void makeNoise(){**

**System.out.print(“Meow!”)**

**{**

**}**

**Public class Dog extends Animal{**

**Public void makeNoise(){**

**System.out.prini(“Woof!”);**

**}**

**}**

**Public class Test{**

**Public static void main(String args){**

**Animal a = new Animal();**

**a.makeNoise();**

**Cat c= new cat();**

**c.makeNouisse();**

**Dog d=new Dog();**

**d.makeNoise();**

**}**

**}**

**//output**

**//Generic animal noise**

**//meow**

**//woof!**

**5. Class is a way of encapsulating data and behaviour in object oriented programming. It is a template that defines the properties and methods of an objects. A class is like a blueprint for creating objects. It allows developers to create objects that share the same properties and methods.**

**Sample of java code to illustrate the implementation of a class is as follows:**

**Public class car {**

**//properties of the car class**

**Private string make;**

**Private string model;**

**Private int year;**

**//contructor**

**Public car(string make, string model, int year){**

**This.make=make;**

**This.model=model;**

**}**

**//getter and setter**

**Public string getmake(){**

**Return make;**

**}**

**Public void setmake(string make) {**

**This.make=make;**

**}**

**Public string getmodel(){**

**Return model;**

**}**

**Public void setmodel(string model){**

**This.model;**

**}**

**Public int getyear(){**

**Return year;**

**}**

**Public void setyear(int year){**

**This.year=year;**

**}**

**}**

**6. Inheritance is a feature of object-oriented programming that allows a class to inherit the properties and methods of another class. This allows for code reuse and makes it easier to maintain code.**

**For example, in Java, a class can be declared using the extends keyword to indicate that it inherits from another class.**

**// Parent class**

**public class Vehicle {**

**public void startEngine() {**

**System.out.println("Starting engine of Vehicle");**

**}**

**}**

**// Child class**

**public class Car extends Vehicle {**

**public void startEngine() {**

**System.out.println("Starting engine of Car");**

**}**

**}**

**// main method**

**public static void main(String args[]) {**

**// creating object of child class**

**Car obj = new Car();**

**obj.startEngine();**

**}**

**The output of this code would be:**

**Starting engine of Car**

**1. Association: Association is a relationship between two objects. It is a type of structural relationship that represents how objects are related to one another. This relationship can be unidirectional or bidirectional.**

**2. Aggregation: Aggregation is a special type of association that represents a part-whole or a-has-a relationship. It is a relationship where the child object can exist independently of the parent object.**

**3. Composition: Composition is a special type of aggregation that represents a strong “has-a” relationship. It is a relationship where the child object cannot exist independently of the parent object. The child object is tightly coupled to the parent object and cannot be reused in other contexts.**

**A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. Class diagrams are widely used in software development to describe the types of objects in a system and their relationships.**

**Steps to Draw a Class Diagram:**

**1. Identify the classes: Identify the classes that are necessary to model the system.**

**2. Identify the attributes: Identify the attributes of each class.**

**3. Identify the relationships: Identify the relationships between classes.**

**4. Identify the operations: Identify the operations of each class.**

**5. Draw the class diagram: Draw the class diagram using the classes, attributes, relationships, and operations identified.**

**6. Add multiplicity: Add multiplicity information such as cardinality and ordering.**

**7. Add constraints: Add any constraints such as invariants or pre/post conditions.**

**8. Refine the diagram: Refine the diagram by adding details such as abstract classes, interfaces, and collaborations.**

**Example:**

**Let's consider a simple example of a Library System.**

**1. Identify the classes: The classes in the system are Library, Book, and Member.**

**2. Identify the attributes: The attributes of the Library class are name, address, and phone number. The attributes of the Book class are title, author, and ISBN. The attributes of the Member class are name, address, and phone number.**

**3. Identify the relationships: The relationship between Library and Book is a one-to-many relationship. The relationship between Library and Member is a one-to-many relationship.**

**4. Identify the operations: The operations of the Library class are addBook(), removeBook(), and searchBook(). The operations of the Book class are addAuthor(), removeAuthor(), and searchAuthor(). The operations of the Member class are borrowBook(), returnBook(), and searchBook().**

**5. Draw the class diagram: The class diagram for the Library System is shown below.**

**6. Add multiplicity: The multiplicity between Library and Book is one-to-many. The multiplicity between Library and Member is one-to-many.**

**7. Add constraints: No constraints are necessary for this system.**

**8.Ref**

**A.**

**// Single inheritance example**

**#include <iostream>**

**using namespace std;**

**// Base class**

**class Shape {**

**public:**

**// Area method**

**void getArea() {**

**cout << "Area of the shape is : " << area << endl;**

**}**

**// Perimeter method**

**void getPerimeter() {**

**cout << "Perimeter of the shape is : " << perimeter << endl;**

**}**

**protected:**

**float area;**

**float perimeter;**

**};**

**// Derived class**

**class Rectangle : public Shape {**

**public:**

**Rectangle(float l, float w) {**

**length = l;**

**width = w;**

**area = length \* width;**

**perimeter = 2 \* (length + width);**

**}**

**private:**

**float length;**

**float width;**

**};**

**int main() {**

**// Create an object of Rectangle class**

**Rectangle rect(10, 5);**

**// Call the getArea() and getPerimeter() methods**

**rect.getArea();**

**rect.getPerimeter();**

**return 0;**

**}**

**// Multiple inheritance example**

**#include <iostream>**

**using namespace std;**

**// Base class 1**

**class Shape {**

**public:**

**// Area method**

**void getArea() {**

**cout << "Area of the shape is : " << area << endl;**

**}**

**protected:**

**float area;**

**};**

**// Base class 2**

**class Parameter {**

**public:**

**// Perimeter method**

**void getPerimeter() {**

**cout << "Perimeter of the shape is : " << perimeter << endl;**

**}**

**protected:**

**float perimeter;**

**};**

**// Derived class**

**class Rectangle : public Shape, public Parameter {**

**public:**

**Rectangle(float l, float w) {**

**length = l;**

**width = w;**

**area = length \* width;**

**perimeter = 2 \* (length + width);**

**}**

**private:**

**float length;**

**float width;**

**};**

**int main() {**

**// Create**

**B.**

**#include <iostream>**

**using namespace std;**

**// Abstract Base Class**

**class Shape {**

**public:**

**// Pure virtual function**

**virtual float calculateArea() = 0;**

**virtual float calculatePerimeter() = 0;**

**};**

**// Derived Class**

**class Circle : public Shape {**

**private:**

**float radius;**

**public:**

**Circle(float r) {**

**radius = r;**

**}**

**float calculateArea() {**

**return 3.14 \* radius \* radius;**

**}**

**float calculatePerimeter() {**

**return 2 \* 3.14 \* radius;**

**}**

**};**

**// Derived Class**

**class Rectangle : public Shape {**

**private:**

**float length, width;**

**public:**

**Rectangle(float l, float w) {**

**length = l;**

**width = w;**

**}**

**float calculateArea() {**

**return length \* width;**

**}**

**float calculatePerimeter() {**

**return 2 \* (length + width);**

**}**

**};**

**// Derived Class**

**class Triangle : public Shape {**

**private:**

**float side1, side2, side3;**

**public:**

**Triangle(float s1, float s2, float s3) {**

**side1 = s1;**

**side2 = s2;**

**side3 = s3;**

**}**

**float calculateArea() {**

**float s = (side1 + side2 + side3) / 2;**

**return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));**

**}**

**float calculatePerimeter() {**

**return side1 + side2 + side3;**

**}**

**};**

**// Derived Class**

**class Square : public Shape {**

**private:**

**float side;**

**public:**

**Square(float s) {**

**side = s;**

**}**

**float calculateArea() {**

**return side \* side;**

**}**

**float calculatePerimeter() {**

**return 4 \* side;**

**}**

**};**

**// Friend Function**

**void calculate(Shape &s) {**

**cout << "Area: " << s.cal**

**C.**

**#include <iostream>**

**using namespace std;**

**class Shape**

**{**

**public:**

**virtual void calculateArea() = 0;**

**virtual void calculatePerimeter() = 0;**

**};**

**class Circle : public Shape**

**{**

**private:**

**float radius;**

**public:**

**Circle(float r)**

**{**

**radius = r;**

**}**

**void calculateArea()**

**{**

**cout << "Area of Circle: " << 3.14 \* radius \* radius << endl;**

**}**

**void calculatePerimeter()**

**{**

**cout << "Perimeter of Circle: " << 2 \* 3.14 \* radius << endl;**

**}**

**};**

**class Rectangle : public Shape**

**{**

**private:**

**float length;**

**float breadth;**

**public:**

**Rectangle(float l, float b)**

**{**

**length = l;**

**breadth = b;**

**}**

**void calculateArea()**

**{**

**cout << "Area of Rectangle: " << length \* breadth << endl;**

**}**

**void calculatePerimeter()**

**{**

**cout << "Perimeter of Rectangle: " << 2 \* (length + breadth) << endl;**

**}**

**};**

**class Triangle : public Shape**

**{**

**private:**

**float side1;**

**float side2;**

**float side3;**

**public:**

**Triangle(float s1, float s2, float s3)**

**{**

**side1 = s1;**

**side2 = s2;**

**side3 = s3;**

**}**

**void calculateArea()**

**{**

**float s = (side1 + side2 + side3) / 2;**

**cout << "Area of Triangle: " << sqrt(s \* (s - side1) \* (s - side2) \* (s - side3)) << endl;**

**}**

**void calculatePerimeter()**

**{**

**cout << "Perimeter of Triangle: " << side1 + side2 + side3 << endl;**

**}**

**};**

**class Square : public Shape**

**{**

**private:**

**float side;**

**public:**

**Square(float**

**D.**

**//Early Binding**

**#include <iostream>**

**using namespace std;**

**//Base class**

**class Shape {**

**public:**

**virtual double getArea() = 0;**

**virtual double getPerimeter() = 0;**

**};**

**//Derived classes**

**class Circle : public Shape {**

**private:**

**double radius;**

**public:**

**Circle(double r) {**

**radius = r;**

**}**

**double getArea() {**

**return 3.14 \* radius \* radius;**

**}**

**double getPerimeter() {**

**return 2 \* 3.14 \* radius;**

**}**

**};**

**class Rectangle : public Shape {**

**private:**

**double length;**

**double width;**

**public:**

**Rectangle(double l, double w) {**

**length = l;**

**width = w;**

**}**

**double getArea() {**

**return length \* width;**

**}**

**double getPerimeter() {**

**return 2 \* (length + width);**

**}**

**};**

**class Triangle : public Shape {**

**private:**

**double side1;**

**double side2;**

**double side3;**

**public:**

**Triangle(double s1, double s2, double s3) {**

**side1 = s1;**

**side2 = s2;**

**side3 = s3;**

**}**

**double getArea() {**

**double s = (side1 + side2 + side3) / 2;**

**return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));**

**}**

**double getPerimeter() {**

**return side1 + side2 + side3;**

**}**

**};**

**class Square : public Shape {**

**private:**

**double side;**

**public:**

**Square(double s) {**

**side = s;**

**}**

**double getArea() {**

**return side \* side;**

**}**

**double getPerimeter() {**

**return 4 \* side;**

**}**

**};**

**//Main function**

**int main() {**

**Shape \*shape;**

**double area, perimeter;**

**int option;**

**cout << "Choose a shape:\n";**

**E.**

**#include <iostream>**

**using namespace std;**

**// Abstraction class for area and perimeter calculator**

**class AreaPerimeterCalculator {**

**public:**

**// pure virtual function to calculate area and perimeter of different shapes**

**virtual double calculateArea() = 0;**

**virtual double calculatePerimeter() = 0;**

**};**

**// Class for Circle**

**class Circle : public AreaPerimeterCalculator {**

**private:**

**double radius;**

**public:**

**Circle(double radius) {**

**this->radius = radius;**

**}**

**double calculateArea() {**

**return 3.14 \* radius \* radius;**

**}**

**double calculatePerimeter() {**

**return 2 \* 3.14 \* radius;**

**}**

**};**

**// Class for Rectangle**

**class Rectangle : public AreaPerimeterCalculator {**

**private:**

**double length;**

**double width;**

**public:**

**Rectangle(double length, double width) {**

**this->length = length;**

**this->width = width;**

**}**

**double calculateArea() {**

**return length \* width;**

**}**

**double calculatePerimeter() {**

**return 2 \* (length + width);**

**}**

**};**

**// Class for Triangle**

**class Triangle : public AreaPerimeterCalculator {**

**private:**

**double side1;**

**double side2;**

**double side3;**

**public:**

**Triangle(double side1, double side2, double side3) {**

**this->side1 = side1;**

**this->side2 = side2;**

**this->side3 = side3;**

**}**

**double calculateArea() {**

**double s = (side1 + side2 + side3) / 2;**

**return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));**

**}**

**double calculatePerimeter() {**

**return side1 + side2 + side3;**

**}**

**};**

**// Class for Square**

**class Square : public AreaPerimeterCalculator {**

**private:**

**double side;**

**public:**

**Square(double side) {**

**this->side**

**8. A**

**Function overloading is when a function is declared multiple times with different parameters. This allows the programmer to call the same function with different parameters to achieve different results.**

**Operator overloading is when an operator is given a special meaning for a user-defined data type. This allows the programmer to use the same operator with different data types to achieve different results.**

**8.B**

**Pass by value:**

**Pass by value is a method of passing arguments to a function in which the argument is copied into a new variable within the function. This means that any changes made to the argument within the function will not be reflected in the original variable outside of the function.**

**Pass by reference:**

**Pass by reference is a method of passing arguments to a function in which the argument is not copied into a new variable. Instead, a reference to the original argument is passed into the function. This means that any changes made to the argument within the function will be reflected in the original variable outside of the function.**

**8.C**

**Parameters are the variables that are defined in a function definition. They represent the type and number of values that can be supplied to the function when it is called.**

**Arguments are the actual values that are passed into a function when it is called. They are the values that are provided to the function so that it can do its job.**

**public class CalculateG { public double multi(double a, double b){ // method for multiplication return a\*b; } public double pow(double a, double b){ // method for powering to square return Math.pow(a,b); } public double sum(double a, double b){ // method for summation return a+b; }**

**5 and 6.**

**public void outline(double finalPosition, double finalVelocity){ // method for printing out a result System.out.println("The object's position after 30 seconds is " + finalPosition + " m."); System.out.println("The object's velocity after 30 seconds is " + finalVelocity + " m/s."); } int main() { double gravity =-9.81; // Earth's gravity in m/s^2 double fallingTime = 30; double initialVelocity = 0.0; double finalVelocity = multi(gravity, fallingTime) + initialVelocity; double initialPosition = 0.0; double finalPosition = 0.5 \* multi(gravity, pow(fallingTime,2)) + multi(initialVelocity, fallingTime) + initialPosition; // Add the formulas for position and velocity // Add output line for velocity (similar to position) outline(finalPosition, finalVelocity); } }**

**Question 3.**

**A.**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int arr[15];**

**cout << "Enter 15 integers: " << endl;**

**// Get values from user**

**for (int i = 0; i < 15; i++)**

**{**

**cin >> arr[i];**

**}**

**// Print values stored in the array**

**for (int i = 0; i < 15; i++)**

**{**

**cout << arr[i] << " ";**

**}**

**cout << endl;**

**return 0;**

**}**

**3.B**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int arr[15];**

**int arr2[15];**

**int num, sum = 0, product = 1;**

**// Taking 15 values from user**

**cout << "Enter 15 values: " << endl;**

**for (int i = 0; i < 15; i++)**

**{**

**cin >> arr[i];**

**}**

**// Ask user to enter a number and check for its presence**

**cout << "Enter a number to check: ";**

**cin >> num;**

**int flag = 0;**

**for (int i = 0; i < 15; i++)**

**{**

**if (arr[i] == num)**

**{**

**flag = 1;**

**cout << "Number found at index " << i << endl;**

**break;**

**}**

**}**

**if (flag == 0)**

**cout << "Number not found in this array" << endl;**

**// Copying elements of arr to arr2 in reverse order**

**for (int i = 14; i >= 0; i--)**

**{**

**arr2[14 - i] = arr[i];**

**}**

**// Printing elements of arr2**

**cout << "Elements of the new array are: " << endl;**

**for (int i = 0; i < 15; i++)**

**{**

**cout << arr2[i] << " ";**

**}**

**cout << endl;**

**// Calculating sum and product of all elements of arr**

**for (int i = 0; i < 15; i++)**

**{**

**sum += arr[i];**

**product \*= arr[i];**

**}**

**// Printing sum and product**

**cout << "Sum of elements of array: " << sum << endl;**

**cout << "Product of elements of array: " << product << endl;**

**return 0;**

**3.C**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int arr[15];**

**int newarr[15];**

**cout << "Enter 15 integers: ";**

**for (int i = 0; i < 15; i++)**

**cin >> arr[i];**

**// Copying the elements of arr to newarr in reverse order**

**for (int i = 0; i < 15; i++)**

**newarr[i] = arr[14-i];**

**// Printing the elements of newarr**

**cout << "The elements of the new array in reverse order are: ";**

**for (int i = 0; i < 15; i++)**

**cout << newarr[i] << " ";**

**return 0;**

**}**

**3.D**

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**int array[15];**

**int sum = 0;**

**int product = 1;**

**cout << "Enter 15 values of type integer: " << endl;**

**for (int i = 0; i < 15; i++)**

**{**

**cin >> array[i];**

**sum += array[i];**

**product \*= array[i];**

**}**

**cout << "The product of all elements of the array is: " << product << endl;**

**cout << "The sum of all elements of the array is: " << sum << endl;**

**return 0;**

**}**