**BENJAMIN ORINA OMBUI**

**SCT121-1044/2022**

**DIT: OOP Assignment**

**Part A:**

**Instructions for part A: answer all the Questions in this section.**

1. **Using a well labeled diagram, explain the steps of creating a system using OOP principles. [4 Marks]**

*Steps of Creating a System Using OOP Principles:*

*Identify Objects:*

*Identify the main entities or objects that represent key elements in the system. These could be real-world entities or abstract concepts.*

*Define Classes:*

*For each identified object, create a class. A class is a blueprint that defines the properties (attributes) and behaviors (methods) of the object.*

*Establish Relationships:*

*Identify relationships between classes. Determine how objects interact with each other. Common relationships include association, aggregation, and composition.*

*Encapsulation:*

*Encapsulate the data by defining access modifiers (public, private, protected) for class members (attributes and methods) to control their visibility and access.*

*Inheritance:*

*Utilize inheritance to promote code reuse. Identify commonalities among classes and create a hierarchy where subclasses inherit attributes and behaviors from a superclass.*

*Polymorphism:*

*Implement polymorphism to enable objects of different classes to be treated as objects of a common base class. This allows for flexibility and extensibility in the system.*

*Create Objects:*

*Instantiate objects from the defined classes. Objects represent instances of the classes and will have their own unique state while sharing common behaviors.*

*Implement Methods:*

*Implement the methods in each class. Methods define the behaviors and operations that objects of the class can perform.*

*Testing:*

*Test the system by creating scenarios and verifying that the objects behave as expected. Ensure that interactions between objects are correct.*

*Refinement:*

*Refine the system based on testing results and feedback. This may involve modifying classes, adding new features, or improving existing functionality.*

*Documentation:*

*Document the system by creating class diagrams, sequence diagrams, and other UML diagrams to provide a clear understanding of the structure and behavior of the system.*

*Maintenance:*

*Regularly maintain and update the system as needed. This includes addressing bugs, adding new features, and ensuring the system remains scalable and adaptable*

1. **What is the Object Modeling Techniques (OMT). [1 Marks]**

*-Object Modeling Techniques (OMT) is a method for modeling and designing systems using object-oriented principles.*

1. **Compare object-oriented analysis and design (OOAD) and object analysis and design (OOP). [2 Marks]**

*OOAD is concerned with the high-level design and analysis of a system before the actual coding begins while OOP is focused on the actual implementation of the system, translating the design into executable code.*

1. **Discuss Mian goals of UML. [2 Marks]**

*Standardization: Establish a universal modeling notation for clear communication.*

*Visual Representation: Use graphical diagrams to represent system aspects*

1. **DESCRIBE three advantages of using object oriented to develop an information system. [3Marks]**

*1.Modularity and Reusability:*

*Modularity: Breaks the system into manageable objects, enhancing organization.*

*Reusability: Allows the reuse of objects, reducing development effort and time.*

*2.Encapsulation and Security:*

*Encapsulation: Hides internal details, promoting robust and secure systems.*

*Security: Enables control over data access, enhancing overall system security.*

*3.Inheritance and Extensibility:*

*Inheritance: Supports code reuse, establishing class hierarchies.*

*Extensibility: Facilitates the creation of new classes based on existing ones, adapting to changing requirements efficiently.*

1. **Briefly explain the following terms as used in object-oriented programming. Write a sample java code to illustrate the implementation of each concept. [12 Marks]**
   1. **Constructor**
   2. **object**
   3. **Destructor**
   4. **polymorphism**
   5. **class**
   6. **Inheritance**

*a. Constructor:*

*Explanation: A constructor is a special method in a class that gets called when an object is created. It initializes the object's state and can take parameters.*

*Java Code Example:*

*java*

*Copy code*

*public class Dog {*

*String name;*

*// Constructor*

*public Dog(String dogName) {*

*name = dogName;*

*System.out.println(name + " is born!");*

*}*

*}*

*public class Main {*

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

*// Creating a Dog object invokes the constructor*

*Dog myDog = new Dog("Buddy");*

*}*

*}*

*b. Object:*

*Explanation: An object is an instance of a class. It represents a real-world entity and has state (attributes) and behavior (methods).*

*Java Code Example:*

*java*

*Copy code*

*public class Car {*

*String brand;*

*public void start() {*

*System.out.println("The " + brand + " car is starting.");*

*}*

*}*

*public class Main {*

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

*// Creating an object of the Car class*

*Car myCar = new Car();*

*myCar.brand = "Toyota";*

*myCar.start();*

*}*

*}*

*c. Destructor:*

*Explanation: In Java, there is no explicit destructor like in some other languages. Java uses garbage collection to automatically reclaim memory when an object is no longer needed.*

*d. Polymorphism:*

*Explanation: Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables methods to be called on objects without knowing their specific types.*

*Java Code Example:*

*java*

*Copy code*

*public class Animal {*

*public void makeSound() {*

*System.out.println("Some generic animal sound");*

*}*

*}*

*public class Dog extends Animal {*

*@Override*

*public void makeSound() {*

*System.out.println("Woof! Woof!");*

*}*

*}*

*public class Cat extends Animal {*

*@Override*

*public void makeSound() {*

*System.out.println("Meow!");*

*}*

*}*

*public class Main {*

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

*// Polymorphism: Objects of different types treated as the common base type*

*Animal myDog = new Dog();*

*Animal myCat = new Cat();*

*myDog.makeSound(); // Output: Woof! Woof!*

*myCat.makeSound(); // Output: Meow!*

*}*

*}*

*e. Class:*

*Explanation: A class is a blueprint for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of the class will have.*

*Java Code Example:*

*java*

*Copy code*

*public class Car {*

*String brand;*

*int year;*

*public void start() {*

*System.out.println("The " + year + " " + brand + " car is starting.");*

*}*

*}*

*public class Main {*

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

*// Creating an object of the Car class*

*Car myCar = new Car();*

*myCar.brand = "Toyota";*

*myCar.year = 2022;*

*myCar.start();*

*}*

*}*

*f. Inheritance:*

*Explanation: Inheritance allows a new class (subclass/derived class) to inherit the properties and behaviors of an existing class (superclass/base class). It promotes code reusability.*

*Java Code Example:*

*java*

*Copy code*

*public class Animal {*

*public void eat() {*

*System.out.println("The animal is eating.");*

*}*

*}*

*public class Dog extends Animal {*

*public void bark() {*

*System.out.println("Woof! Woof!");*

*}*

*}*

*public class Main {*

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

*// Inheritance: Dog inherits from Animal*

*Dog myDog = new Dog();*

*myDog.eat(); // Inherited method*

*myDog.bark(); // Own method*

*}*

*}*

**vi. *EXPLAIN* the three types of associations (relationships) between objects in object oriented. [6 Marks]**

*a. Association*

*Explanation: Basic relationship between classes without specifying the nature of the connection.*

*Example: Car and Driver have an association; a car is associated with a driver.*

*b. Aggregation*

*Explanation: "Whole-part" relationship where one class contains another, but the contained class can exist independently.*

*Example: Department has an aggregation with Professor; a department contains professors, but professors can exist independently.*

*c.Composition*

*Explanation: Stronger form of aggregation where the contained class is part of the whole and has no meaningful existence outside the whole.*

*Example: Canvas has a composition with Shape; shapes are part of the canvas, and if the canvas is deleted, the shapes are also deleted.*

**Vii. What do you mean by class diagram? Where it is used and also discuss the steps to draw the class diagram with any one example.**

*A class diagram is a type of diagram in the Unified Modeling Language (UML) that represents the structure and relationships of a system's classes and their components. It provides a visual representation of the classes, their attributes, methods, and associations between them.*

*Purpose and Usage of Class Diagrams:*

*Visualization:*

*Class diagrams help in visualizing the structure of a system by representing classes and their relationships.*

*Design and Analysis:*

*They are used in software design and analysis to plan and document the structure of a system before implementation.*

*Communication:*

*Class diagrams serve as a communication tool among stakeholders, such as developers, architects, and project managers.*

*Documentation:*

*They are used to document the software architecture, helping in understanding and maintaining the system.*

*Steps to Draw a Class Diagram:*

*1. Identify Classes:*

*Identify the main classes that represent the key entities in your system.*

*2. Identify Attributes:*

*For each class, identify its attributes (properties or variables).*

*3. Identify Methods:*

*Identify the methods (functions or operations) associated with each class.*

*4. Identify Relationships:*

*Determine the relationships between classes. Common relationships include association, aggregation, and composition.*

*5. Add Multiplicity:*

*Specify the multiplicity of the relationships to show how many instances of one class are related to another.*

*6. Add Associations:*

*Connect the classes using associations to represent relationships between them.*

*7. Add Inheritance:*

*If there are inheritance relationships, represent them using arrows with a triangle pointing to the superclass.*

*8. Add Aggregation and Composition:*

*Use diamond-shaped arrows to represent aggregation (hollow diamond) and composition (filled diamond).*

*9. Add Access Modifiers:*

*Use symbols like '+' (public), '-' (private), or '#' (protected) to denote access modifiers for attributes and methods.*

*10. Review and Refine:*

*Review the class diagram to ensure it accurately represents the structure and relationships of the system. Refine as needed.*

*EXAMPLE*

*#include <string>*

*#include <vector>*

*class Author {*

*public:*

*std::string name;*

*std::string biography;*

*Author(std::string name, std::string biography) : name(name), biography(biography) {}*

*void writeBook() {*

*// Implement the method here*

*}*

*};*

*class Book {*

*public:*

*std::string title;*

*std::string ISBN;*

*bool availability;*

*Author\* author; // This creates an association with the Author class*

*Book(std::string title, std::string ISBN, bool availability, Author\* author)*

*: title(title), ISBN(ISBN), availability(availability), author(author) {}*

*void checkout() {*

*// Implement the method here*

*}*

*void returnBook() {*

*// Implement the method here*

*}*

*};*

*class Library {*

*public:*

*std::string name;*

*std::string location;*

*std::vector<Book\*> books; // This will hold the association with the Book class*

*Library(std::string name, std::string location) : name(name), location(location) {}*

*void searchBook() {*

*// Implement the method here*

*}*

*void addBook(Book\* book) {*

*books.push\_back(book); // This creates an association with the Book class*

*}*

*};* **[6 Marks]**

1. **Given that you are creating area and perimeter calculator using C++, to computer area and perimeter of various shaped like Circles, Rectangle, Triangle and Square, use well written code to explain and implement the calculator using the following OOP concepts.**
   1. **Inheritance (Single inheritance, Multiple inheritance and Hierarchical inheritance) [10 Marks]**
   2. **Friend functions [5 Marks]**
   3. **Method overloading and method overriding [10 Marks]**
   4. **Late binding and early binding [8 Marks]**
   5. **Abstract class and pure functions [6 Marks]**

*#include <iostream>*

*#include <cmath>*

*// Abstract class representing a Shape*

*class Shape {*

*public:*

*virtual double calculateArea() const = 0;*

*virtual double calculatePerimeter() const = 0;*

*};*

*// Circle class inheriting from Shape*

*class Circle : public Shape {*

*private:*

*double radius;*

*public:*

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

*double calculateArea() const override {*

*return 3.14 \* radius \* radius;*

*}*

*double calculatePerimeter() const override {*

*return 2 \* 3.14 \* radius;*

*}*

*};*

*// Rectangle class inheriting from Shape*

*class Rectangle : public Shape {*

*private:*

*double length;*

*double width;*

*public:*

*Rectangle(double l, double w) : length(l), width(w) {}*

*double calculateArea() const override {*

*return length \* width;*

*}*

*double calculatePerimeter() const override {*

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

*}*

*};*

*// Triangle class inheriting from Shape*

*class Triangle : public Shape {*

*private:*

*double sideA;*

*double sideB;*

*double sideC;*

*public:*

*Triangle(double a, double b, double c) : sideA(a), sideB(b), sideC(c) {}*

*double calculateArea() const override {*

*double s = (sideA + sideB + sideC) / 2.0;*

*return sqrt(s \* (s - sideA) \* (s - sideB) \* (s - sideC));*

*}*

*double calculatePerimeter() const override {*

*return sideA + sideB + sideC;*

*}*

*};*

*// Square class inheriting from Rectangle (Single Inheritance)*

*class Square : public Rectangle {*

*public:*

*Square(double side) : Rectangle(side, side) {}*

*};*

*// Friend function to calculate the diagonal of a Rectangle*

*double calculateDiagonal(const Rectangle& rectangle) {*

*return sqrt(rectangle.length \* rectangle.length + rectangle.width \* rectangle.width);*

*}*

*int main() {*

*// Single Inheritance*

*Square square(4);*

*std::cout << "Square Area: " << square.calculateArea() << ", Perimeter: " << square.calculatePerimeter() << std::endl;*

*// Multiple Inheritance*

*Circle circle(5);*

*std::cout << "Circle Area: " << circle.calculateArea() << ", Perimeter: " << circle.calculatePerimeter() << std::endl;*

*// Hierarchical Inheritance*

*Triangle triangle(3, 4, 5);*

*std::cout << "Triangle Area: " << triangle.calculateArea() << ", Perimeter: " << triangle.calculatePerimeter() << std::endl;*

*// Friend Function*

*Rectangle rectangle(3, 4);*

*std::cout << "Rectangle Diagonal: " << calculateDiagonal(rectangle) << std::endl;*

*return 0;*

*}*

*Explanation:*

*Inheritance:*

*Single Inheritance: Square inherits from Rectangle.*

*Multiple Inheritance: Circle and Triangle inherit from Shape.*

*Hierarchical Inheritance: Rectangle and Triangle inherit from Shape.*

*Friend Function:*

*The calculateDiagonal function is a friend function of the Rectangle class, allowing it to access private members of Rectangle.*

*Method Overloading and Method Overriding:*

*The calculateArea and calculatePerimeter methods are overloaded in different shapes and overridden in derived classes.*

*Late Binding and Early Binding:*

*Late Binding: Achieved through virtual functions in the Shape class. The actual function to be called is determined at runtime.*

*Early Binding: The binding for non-virtual functions is determined at compile-time.*

*Abstract Class and Pure Functions:*

*Shape is an abstract class with pure virtual functions (calculateArea and calculatePerimeter), making it an interface that must be implemented by derived classes.*

1. **Using a program written in C++, differentiate between the following. [6 Marks]**
   1. **Function overloading and operator overloading**
   2. **Pass by value and pass by reference**
   3. **Parameters and arguments**

***Solution:***

a. Function Overloading and Operator Overloading:

Function Overloading:

Function overloading refers to defining multiple functions in the same scope with the same name but different parameters. The compiler determines which function to call based on the number or types of parameters.

*#include <iostream>*

*class OverloadExample {*

*public:*

*// Function Overloading*

*void print(int value) {*

*std::cout << "Printing integer: " << value << std::endl;*

*}*

*void print(double value) {*

*std::cout << "Printing double: " << value << std::endl;*

*}*

*};*

*int main() {*

*OverloadExample obj;*

*obj.print(5);*

*obj.print(3.14);*

*return 0;*

*}*

*Operator Overloading:*

*Operator overloading involves defining custom behaviors for operators when used with user-defined data types (classes/structures).*

*#include <iostream>*

*class Complex {*

*public:*

*double real;*

*double imag;*

*// Operator Overloading for '+'*

*Complex operator+(const Complex& other) const {*

*Complex result;*

*result.real = real + other.real;*

*result.imag = imag + other.imag;*

*return result;*

*}*

*};*

*int main() {*

*Complex a{2.0, 3.0};*

*Complex b{1.0, 2.0};*

*Complex sum = a + b;*

*std::cout << "Sum: " << sum.real << " + " << sum.imag << "i" << std::endl;*

*return 0;*

*}*

*b. Pass by Value and Pass by Reference:*

*Pass by Value:*

*Passing by value involves sending a copy of the variable to the function. Any changes made to the parameter inside the function do not affect the original variable.*

*#include <iostream>*

*void incrementByValue(int x) {*

*x++;*

*}*

*int main() {*

*int num = 5;*

*incrementByValue(num);*

*std::cout << "Original value: " << num << std::endl; // Output: Original value: 5*

*return 0;*

*}*

*Pass by Reference:*

*Passing by reference involves sending the memory address of the variable to the function. Changes made to the parameter inside the function affect the original variable.*

*#include <iostream>*

*void incrementByReference(int &x) {*

*x++;*

*}*

*int main() {*

*int num = 5;*

*incrementByReference(num);*

*std::cout << "Modified value: " << num << std::endl; // Output: Modified value: 6*

*return 0;*

*}*

*c. Parameters and Arguments:*

*Parameters:*

*Parameters are variables declared in a function's signature. They act as placeholders for the values that will be passed to the function when it is called.*

*#include <iostream>*

*void add(int a, int b) {*

*std::cout << "Sum: " << a + b << std::endl;*

*}*

*int main() {*

*add(3, 5); // 3 and 5 are arguments*

*return 0;*

*}*

*Arguments:*

*Arguments are the actual values passed to a function when it is called. They match the parameters' types and order in the function signature.*

*#include <iostream>*

*void multiply(int x, int y) {*

*std::cout << "Product: " << x \* y << std::endl;*

*}*

*int main() {*

*int a = 4;*

*int b = 6;*

*multiply(a, b); // a and b are arguments*

*return 0;*

*}*

***NOTE: To score high marks, you are required to explain each question in detail. Do good research and cite all the sources of your information. DO NOTE CITE WIKIPEDIA.***

**Create a new class called *CalculateG.*Copy and paste the following initial version of the code. Note variables declaration and the types.**

**class** *CalculateG* **{  
int** main**(){**

(*datatype*) gravity =-9.81; // Earth's gravity in m/s^2 (*datatype*) fallingTime = 30;

(*datatype*)initialVelocity = 0.0; (*datatype*) finalVelocity = ;

(*datatype*) initialPosition = 0.0; (*datatype*) finalPosition = ;

// Add the formulas for position and velocity

Cout<<"The object's position after " << fallingTime << " seconds is "

+ finalPosition + << m."<<endl;

// Add output line for velocity (similar to position)

} }

Modify the example program to compute the position and velocity of an object after falling for 30 seconds, outputting the position in meters. The formula in Math notation is:

𝑥(𝑡)=0.5∗𝑎𝑡2 +𝑣𝑖𝑡+𝑥𝑖 𝑣(𝑡)=𝑎𝑡+𝑣𝑖

Solution:

*#include <iostream>*

*class CalculateG {*

*public:*

*int main() {*

*double gravity = -9.81; // Earth's gravity in m/s^2*

*double fallingTime = 30.0;*

*double initialVelocity = 0.0;*

*double initialPosition = 0.0;*

*// Calculate final position and velocity*

*double finalPosition = 0.5 \* gravity \* fallingTime \* fallingTime + initialVelocity \* fallingTime + initialPosition;*

*double finalVelocity = gravity \* fallingTime + initialVelocity;*

*std::cout << "The object's position after " << fallingTime << " seconds is " << finalPosition << " m." << std::endl;*

*std::cout << "The object's velocity after " << fallingTime << " seconds is " << finalVelocity << " m/s." << std::endl;*

*return 0;*

*}*

*};*

*int main() {*

*CalculateG calculateG;*

*return calculateG.main();*

*}*

Run the completed code in Eclipse (Run → Run As → Java Application). 5. Extend *datatype* class with the following code:

**public class** *CalculateG* {

**public double** multi(**......**){ // method for multiplication

}

// add 2 more methods for powering to square and summation (similar to multiplication)

**public void** outline(**......**){  
// method for printing out a result

}  
**int** main() {

// compute the position and velocity of an object with defined methods and print out the

result

} }

Solution:

public class CalculateG {

// Method for multiplication

public double multi(double a, double b) {

return a \* b;

}

// Method for squaring

public double square(double a) {

return a \* a;

}

// Method for summation

public double sum(double a, double b) {

return a + b;

}

// Method for printing out a result

public void outline(String message, double result) {

System.out.println(message + result);

}

public static void main(String[] args) {

CalculateG calculateG = new CalculateG();

double gravity = -9.81; // Earth's gravity in m/s^2

double fallingTime = 30.0;

double initialVelocity = 0.0;

double initialPosition = 0.0;

// Compute the position and velocity of an object with defined methods

double finalPosition = calculateG.sum(calculateG.multi(0.5, calculateG.multi(gravity, calculateG.square(fallingTime))), calculateG.multi(initialVelocity, fallingTime)) + initialPosition;

double finalVelocity = calculateG.sum(calculateG.multi(gravity, fallingTime), initialVelocity);

// Print out the result

calculateG.outline("The object's position after " + fallingTime + " seconds is ", finalPosition);

calculateG.outline("The object's velocity after " + fallingTime + " seconds is ", finalVelocity);

}

}

6. Create methods for multiplication, powering to square, summation and printing out a result in *CalculateG* class.

solution

*public class CalculateG {*

*// Method for multiplication*

*public double multi(double a, double b) {*

*return a \* b;*

*}*

*// Method for squaring*

*public double square(double a) {*

*return a \* a;*

*}*

*// Method for summation*

*public double sum(double a, double b) {*

*return a + b;*

*}*

*// Method for printing out a result*

*public void outline(String message, double result) {*

*System.out.println(message + result);*

*}*

*}*

**Part B:**

**Instructions for part B: Do question 1 and any other one question from this section.**

1. Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:  
1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, write a C++ method to find the sum of all the even- valued terms.

*#include <iostream>*

*int main () {*

*const int limit = 4000000;*

*int term1 = 1;*

*int term2 = 2;*

*int sum = 0;*

*while (term2 <= limit) {*

*if (term2 % 2 == 0) {*

*sum += term2;*

*}*

*// Generate the next Fibonacci term*

*int nextTerm = term1 + term2;*

*term1 = term2;*

*term2 = nextTerm;*

*}*

*std::cout << "Sum of even-valued Fibonacci terms below 4 million: " << sum << std::endl;*

*return 0;*

*}*

**Question Two: [15 marks]**

2. A palindrome number is a number that remain the same when read from behind or front ( a number that is equal to reverse of number) for example, 353 is palindrome because reverse of 353 is 353 (you see the number remains the same). But a number like 591 is not palindrome because reverse of 591 is 195 which is not equal to 591. Write C++ program to check if a number entered by the user is palindrome or not. You should provide the user with a GUI interface to enter the number and display the results on the same interface.

The interface:

**Check if a number is palindrome**

345

Enter the number

Not palindrome

Output 🡪

*#include <iostream>*

*#include <conio.h>*

*bool isPalindrome(int number) {*

*int originalNumber = number;*

*int reversedNumber = 0;*

*while (number > 0) {*

*int digit = number % 10;*

*reversedNumber = reversedNumber \* 10 + digit;*

*number /= 10;*

*}*

*return originalNumber == reversedNumber;*

*}*

*int main() {*

*int userNumber;*

*std::cout << "Enter a number to check if it's a palindrome: ";*

*std::cin >> userNumber;*

*if (isPalindrome(userNumber)) {*

*std::cout << userNumber << " is a palindrome.\n";*

*} else {*

*std::cout << userNumber << " is not a palindrome.\n";*

*}*

*\_getch(); // Wait for a key press before closing the console window*

*return 0;*

*}*

**Question three: [15 marks]**

Write a C++ program that takes 15 values of type integer as inputs from user, store the values in an array.

1. Print the values stored in the array on screen.
2. Ask user to enter a number, check if that number (entered by user) is present in array or not. If it is present print, “the number found at index (index of the number) ” and the text “number not found in this array”
3. Create another array, copy all the elements from the existing array to the new array but in reverse order. Now print the elements of the new array on the screen
4. Get the sum and product of all elements of your array. Print product and the sum each on its own line.

*#include <iostream>*

*#include <algorithm>*

*int main() {*

*int arr[15];*

*int num;*

*int sum = 0;*

*int product = 1;*

*// Input 15 integers*

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

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

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

*sum += arr[i];*

*product \*= arr[i];*

*}*

*// Print the array*

*std::cout << "The array is: ";*

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

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

*}*

*std::cout << std::endl;*

*// Check if a number is in the array*

*std::cout << "Enter a number to find in the array: ";*

*std::cin >> num;*

*auto it = std::find(arr, arr+15, num);*

*if(it != arr+15)*

*std::cout << "The number found at index " << (it - arr) << std::endl;*

*else*

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

*// Reverse the array*

*std::reverse(arr, arr+15);*

*// Print the reversed array*

*std::cout << "The reversed array is: ";*

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

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

*}*

*std::cout << std::endl;*

*// Print the sum and product*

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

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

*return 0;*

*}*