C++ Problems

**BASIC PROBLEMS(LAB)**

**01.** Write a C++ program to create a class called Rectangle that has private member variables for length and width. Implement member functions to calculate the rectangle's area and perimeter.

**02.** Write a C++ program to create a class called Person that has private member variables for name, age and country. Implement member functions to set and get the values of these variables.

**03.** Write a C++ program to implement a class called BankAccount that has private member variables for account number and balance. Include member functions to deposit and withdraw money from the account.

**04.** Write a C++ program to implement a class called Date that has private member variables for day, month, and year. Include member functions to set and get these variables, as well as to validate if the date is valid.

**05.** Write a C++ program to implement a class called Student that has private member variables for name, class, roll number, and marks. Include member functions to calculate the grade based on the marks and display the student's information.

**06.** Create a class named Car with private data members color, model, and year. Include public methods to:

• Set the car's color, model, and year.

• Display the car's details.

• Check if the car is considered a vintage car (older than 25 years).

**07.** Create a class Box with private data members length, width, and height. Implement a default constructor to initialize these members to 0. Write a program to create an object of the Box class and display the values of its members.

**08.** Create a class Circle with a private data member radius. Implement a parameterized constructor that initializes radius with the given value. Write a program that calculates and displays the area of the circle.

**09.** Create a class Point with private data members x and y coordinates. Implement:

• A default constructor that initializes both x and y to 0.

• A parameterized constructor that initializes x and y with specific values.

Write a program to create objects using both constructors and display the coordinates of each point.

**10.** Create a class Person with private members name and age. Implement:

• A parameterized constructor to initialize name and age.

• A destructor to display a message when the object is destroyed.

Write a program that creates a Person object and displays the details.

**11.** Create a class Rectangle with private members length and width. Implement:

• A default constructor to initialize both length and width to 1.

• A parameterized constructor to initialize length and width with given values.

• A destructor to display a message when the object is destroyed.

Write a program to create two Rectangle objects, one with the default constructor and one with the parameterized constructor, and display their areas.

**12.** Create a class Rectangle with private members length and width. Implement:

• A parameterized constructor to initialize the dimensions.

• A method area() to calculate and return the area.

• A function compareArea(Rectangle r) that takes a Rectangle object by value and compares its area with the current object's area.

Write a program to create two Rectangle objects and use the compareArea() function to compare their areas.

**13.** Create a class Complex to represent a complex number with real and imaginary parts. Implement:

• A parameterized constructor to initialize the complex number.

• A function addComplex(Complex c1, Complex c2) that takes two Complex objects as arguments and returns their sum as a new Complex object.

Write a program to demonstrate adding two complex numbers using this function.

**14.** Create a class Student with private data members name and grade. Implement a constructor with default arguments where:

• name is initialized to "Unknown".

• grade is initialized to 0.

Write a program that creates multiple Student objects, some with given names and grades and others with default values.

**15**. Write a class **BankAccount** with private members for account holder's name, account number, and balance. Use a **constructor** to initialize these values and a destructor to **display** a message indicating when the object is destroyed. Create a BankAccount **object** and display the account details.

**16**. Write a class **SwapValues** with two private **integer** members. Implement a **friend** **function** that swaps the values of the two private members and displays the **swapped** values.

**17**. Create two classes, **Rectangle** and **Circle**, each with a private member representing their **area**. Write a friend function that **compares** the areas of the two objects and prints which object has a **larger** area.

**18**. Write three classes: **ClassX**, **ClassY**, and **ClassZ**, each containing a private integer member. Implement a single **friend** **function** that has access to the private members of all three classes and finds the maximum value among them.

**19**. Define a class **Student** with private members for **name**, **roll**\_**number**, and **marks**. Implement a **copy** **constructor** to copy the details of one student object to another. Write a destructor to display a message when the object is destroyed. Test this by creating a Student **object** and copying it to another.

**20**. Create a program with an **area** **function** that overloaded to calculate:

• The area of a **circle** (given the radius).

• The area of a **rectangle** (given length and width).

• The area of a **triangle** (given base and height).

**21**. Write an overloaded **findMax** function that:

• Returns the maximum of two **integers**.

• Returns the maximum of two **floating**-**point** **numbers**.

• Returns the maximum of three **integers**.

**22**. Create a **Complex** class with private members for the real and imaginary parts. Overload the +, - and \* **operators** to perform addition. subtraction, and multiplication of complex numbers. Test the overloaded operators by creating two complex number objects and performing **arithmetic** operations on them.

**23**. Create a base class called **Device** with an attribute deviceName and a constructor that sets deviceName. Derive two classes from Device:

• **Phone**, which has additional attributes like phoneModel.

• **Tablet**, which has add.uonal attributes like tabletModel.

Now, create a class called **Phablet** that Inherits from both **Phone** and **Tablet**, resulting in a diamond structure. Usc **virtual** **inheritance** for Device to avoid the diamond problem and ambiguity in accessmg deviceName.

**24.** Define a base class Shape with a pure virtual function area(). Create derived classes Circle and Rectangle that override the area() function.

• Circle should calculate the area based on radius.

• Rectangle should calculate the area based on width and height.

Write a program that creates objects of Circle and Rectangle, calls their area() function, and displays the calculated areas.

**25.** Design a base class Vehicle with a virtual function horn(), which outputs "Generic vehicle horn". Create two derived classes Car and Truck.

• Car should override horn() to output "Car horn: Beep Beep!".

• Truck should override horn() to output "Truck horn: Honk Honk!".

Create objects of Car and Truck and demonstrate function overriding by calling the horn() function on each.

**26.** Write a program that performs division between two numbers entered by the user. If the user tries to divide by zero, throw an exception and catch it to display an error message like "Division by zero is not allowed." Test this with different inputs, including zero as the divisor.

**27.** Create a program that initializes an array of 5 integers. Ask the user to enter an index to access an element in the array. If the user enters an index out of bounds, throw an exception and catch it to display a message like "Array index out of bounds." Test this by entering indices within and outside the array bounds.

**28.** Write a program that prompts the user to enter their age. If the user enters an age less than 0 or greater than 150, throw an exception and catch it to display a message like "Invalid age entered." Test this with valid and invalid age inputs to ensure correct handling.

**29.** Create a BankAccount class with a balance attribute and a withdraw function. In the withdraw function, throw an exception if the withdrawal amount exceeds the balance, with a message like "Insufficient funds." Write a program to create a BankAccount object, deposit some money, and try to withdraw an amount greater than the balance to test the exception handling.

**INLINE FUNCTIONS**

**1.** **Square of a Number**  
Write a C++ program using an inline function to calculate the square of a number entered by the user.

**2.** **Sum of Two Numbers**  
Write a C++ program to define an inline function that calculates the sum of two numbers entered by the user.

**3. Area of a Circle**  
Write a C++ program to calculate the area of a circle using an inline function. The radius of the circle is entered by the user.

**4. Factorial Calculation**  
Write a C++ program that uses an inline function to calculate the factorial of a number entered by the user.

**5. Power of a Number**  
Write a C++ program using an inline function to compute a^b, where a and b are entered by the user.

**6. Temperature Conversion**  
Write a C++ program using an inline function to convert a temperature from Fahrenheit to Celsius.

**7. Maximum of Two Numbers**  
Write a C++ program that uses an inline function to find the maximum of two numbers entered by the user.

**8. Check Even or Odd**  
Write a C++ program to check whether a number is even or odd using an inline function.

**Using <iomanip> (setw, setprecision)**

1. **Display Table**  
   Write a C++ program to display a multiplication table for numbers 1 to 5 using setw() from the iomanip library.
2. **Formatted Decimal Output**  
   Write a C++ program to display the results of a division with fixed-point notation and set the precision to two decimal places.
3. **Aligning Text**  
   Write a C++ program to display a list of items and their prices aligned in two columns using setw().
4. **Scientific Notation**  
   Write a C++ program to display a floating-point number in scientific notation using setprecision() and scientific.
5. **Right Alignment**  
   Write a C++ program to align numbers to the right within a 10-character-wide space using setw().
6. **Currency Formatting**  
   Write a C++ program to format and display a dollar amount to 2 decimal places using fixed and setprecision().
7. **Display Hexadecimal Numbers**  
   Write a C++ program to display numbers in decimal, hexadecimal, and octal formats using manipulators.
8. **Dynamic Formatting**  
   Write a C++ program to accept user input for floating-point numbers and display them with varying precisions (1, 2, and 3 decimal places).

**Function Overloading Problems**

1. **Add Two Numbers**  
   Write a C++ program to overload a function add() to add two integers, two floating-point numbers, and two characters (displaying their ASCII sum).
2. **Find Maximum**  
   Write a C++ program to overload a function max() to find the maximum of two integers, two floating-point numbers, and three integers.
3. **Calculate Volume**  
   Write a C++ program to overload a function volume() to calculate the volume of a cube, a cylinder, and a sphere.
4. **Multiply Two Numbers**  
   Write a C++ program to overload a function multiply() to multiply two integers, two floats, and an integer with a float.
5. **Area Calculation**  
   Write a C++ program to overload a function area() to calculate the area of a circle, a rectangle, and a triangle.
6. **Simple Interest Calculation**  
   Write a C++ program to overload a function interest() to calculate simple interest for different principal amounts, rates, and times.
7. **Reverse a Value**  
   Write a C++ program to overload a function reverse() to reverse an integer, a floating-point number, and a string.
8. **Swapping Values**  
   Write a C++ program to overload a function swap() to swap two integers, two floats, and two strings.

**Constructor-Destructor**

1. **Simple Constructor and Destructor**  
   Write a C++ program to demonstrate constructor and destructor calls in a class Demo.
2. **Parameterized Constructor**  
   Write a C++ program to create a class Point with a parameterized constructor to initialize coordinates (x, y) of a point.
3. **Constructor Overloading**  
   Write a C++ program to overload constructors for a class Shape to initialize with different dimensions (circle, rectangle, and triangle).
4. **Object Counter**  
   Write a C++ program to create a class Counter that uses constructors and destructors to count the number of objects created and destroyed.
5. **Copy Constructor**  
   Write a C++ program to demonstrate the use of a copy constructor in a class Person.
6. **Dynamic Memory Allocation**  
   Write a C++ program to demonstrate dynamic memory allocation in a constructor and proper deallocation in a destructor.
7. **Bank Account Simulation**  
   Write a C++ program to simulate a bank account using constructors (default, parameterized) and destructors.
8. **Nested Constructor Calls**  
   Write a C++ program to demonstrate constructor calls for an object that contains other objects as members.

**Copy Constructor**

1. **Copying Student Details**  
   Write a C++ program to create a class Student with attributes like name and roll. Use a copy constructor to create a duplicate object.
2. **Cloning Bank Account**  
   Write a C++ program to create a class BankAccount with private data like accountNumber and balance. Use a copy constructor to duplicate an account object.
3. **Copying Employee Information**  
   Write a C++ program with a class Employee that stores the name and salary of an employee. Use a copy constructor to copy the data of one employee to another.
4. **Copying a Complex Number**  
   Write a C++ program to create a class Complex to store a complex number. Use a copy constructor to initialize a new complex number object from an existing one.

**Object as Function Argument**

**1. Distance Addition**

Write a C++ program to create a class Distance with two data members: feet and inches. Define a member function to input distance values. Write another function that takes two Distance objects as arguments, adds the distances, and returns the resulting Distance object.

**2. Combine Student Marks**

Write a C++ program to create a class Student with attributes for name and marks. Create a function outside the class that takes two Student objects as arguments and combines their marks into a third Student object. Display the combined marks and the name of the resulting student.

**3. Merge Complex Numbers**

Write a C++ program to define a class Complex with real and imaginary parts. Implement a function that takes two Complex objects as arguments, adds their real and imaginary parts separately, and returns the resultant Complex object. Display the result in the form a+bi.

**Friend Function**

1. **Swapping Private Data Members:**

Write a C++ program with a class Swap that has private data members a and b. Use a friend function to swap the values of a and b.

1. **Finding Maximum of Two Numbers**

Write a C++ program to create two classes, ClassA and ClassB. Use a friend function to find the maximum value between private data members of both classes.

1. **Comparing Areas of Two Shapes:**

Write a C++ program with two classes, Rectangle and Circle. Use a friend function to compare the area of a rectangle and a circle.

1. **Adding Private Data of Two Classes:**

Write a C++ program to create two classes, ClassA and ClassB. Use a friend function to add the private data members of both classes and display the result.

1. **Bank Account Transfer:**

Write a C++ program with a class Account that holds private balance data. Use a friend function to transfer a specific amount from one account to another.

1. **Finding Distance Between Two Points:**

Write a C++ program with a class Point that has private x and y coordinates. Use a friend function to calculate the distance between two points.

1. **Finding Average of Two Students’ Marks:**

Write a C++ program to create a class Student with private marks data. Use a friend function to calculate the average marks of two students.

1. **Accessing Private Data of Another Class:**

Write a C++ program with two classes, ClassA and ClassB. Use a friend function to access private data members of ClassA inside ClassB.

**Friend Class**

1. **Accessing Private Data of Another Class**  
   Write a C++ program with two classes, ClassA and ClassB. Declare ClassB as a friend of ClassA so that it can access and display the private data of ClassA.
2. **Bank Account and Bank Manager**  
   Write a C++ program with a class Account that has private data for account balance and transactions. Create a class Manager as a friend to access and modify the private data of Account.
3. **Complex Number Operations**  
   Write a C++ program with two classes, Complex and Calculator. Make Calculator a friend of Complex to access its private real and imaginary parts for performing addition and multiplication.
4. **Rectangle and Square**  
   Write a C++ program with two classes, Rectangle and Square. Declare Square as a friend of Rectangle to allow the square to access and modify the rectangle's private length and width.
5. **Library and Book**  
   Write a C++ program with a class Book that stores details like title and price. Make a class Library a friend of Book to allow access to these private details for inventory management.
6. **Coordinate and Distance Calculation**  
   Write a C++ program with two classes, Point and Distance. Make Distance a friend of Point to access private coordinates and calculate the distance between two points.
7. **Student and Teacher Collaboration**  
   Write a C++ program with a class Student that stores private exam marks. Create a class Teacher as a friend to calculate and display the average marks of a student.
8. **Access Control Between Classes**  
   Write a C++ program with two classes, ClassA and ClassB. Declare ClassA as a friend of ClassB to grant ClassA permission to modify ClassB's private attributes for a specific operation.

**Inheritance**

**Single** **Inheritance:**

1. **Employee and Salary**  
   Write a C++ program to create a base class Employee with attributes name and id. Derive a class Salary that adds the basic salary and bonus attributes and calculates the total salary.
2. **Shape Area Calculation**  
   Write a C++ program to create a base class Shape with attributes length and width. Derive a class Rectangle that calculates the area of the rectangle.
3. **Book and Author**  
   Write a C++ program with a base class Book that contains details like title and ISBN. Derive a class Author that adds the author name and displays all details.
4. **Vehicle and Car**  
   Write a C++ program with a base class Vehicle that has attributes like model and speed. Derive a class Car to add number of doors and display all the details.

**Multiple Inheritance:**

1. **Person and Employee**  
   Write a C++ program to create two base classes, Person with name and age, and Job with company and salary. Derive a class Employee that combines these attributes and displays the details.
2. **Student and Sports**  
   Write a C++ program to create two base classes, Student with academic marks and Sports with sports scores. Derive a class Result to calculate and display the total score.
3. **Device and Battery**  
   Write a C++ program with two base classes, Device containing attributes like brand and model, and Battery with capacity. Derive a class Smartphone to display all attributes.
4. **Animal and Bird**  
   Write a C++ program with two base classes, Animal containing species and diet, and Bird with wing span. Derive a class Eagle that displays all details.

**Multi-Level Inheritance:**

1. **Person, Employee, and Manager**  
   Write a C++ program where Person is a base class with name and age. Derive Employee to add id and department, and then derive Manager to add team size and display all details.
2. **Shape, Rectangle, and Cuboid**  
   Write a C++ program where Shape is a base class with length and width. Derive Rectangle to calculate the area, and then derive Cuboid to calculate volume.
3. **Animal, Mammal, and Dog**  
   Write a C++ program where Animal is a base class with name. Derive Mammal to add warm-blooded property, and then derive Dog to display breed.
4. **Vehicle, Car, and ElectricCar**  
   Write a C++ program where Vehicle is a base class with speed. Derive Car to add fuel type, and then derive ElectricCar to add battery capacity.

**Hierarchical Inheritance:**

1. **Shape, Rectangle, and Triangle**  
   Write a C++ program where Shape is a base class with length and width. Derive Rectangle to calculate area and Triangle to calculate area based on height.
2. **Person, Teacher, and Student**  
   Write a C++ program where Person is a base class with name and age. Derive Teacher to add subject, and Student to add marks.
3. **Animal, Dog, and Cat**  
   Write a C++ program where Animal is a base class with species. Derive Dog to display barking behavior and Cat to display purring behavior.
4. **Vehicle, Bike, and Truck**  
   Write a C++ program where Vehicle is a base class with speed. Derive Bike to display type of bike, and Truck to display load capacity.

**Hybrid Inheritance Problems:**

1. **Student Academic and Sports Score**  
   Write a C++ program where Person is a base class with name and age. Derive Student from Person for academic details and Sports for sports scores. Use a derived class Result to combine and display all details.
2. **Employee with Salary and Project**  
   Write a C++ program where Person is a base class with name and id. Derive Salary for payment details and Project for work details. Create a derived class Manager to combine and display all information.
3. **Animal with Mammal and Bird**  
   Write a C++ program where Animal is a base class with species. Derive Mammal for mammals’ properties and Bird for birds’ properties. Create a derived class Bat to display details of a mammal that can fly.
4. **Device with Features and Warranty**  
   Write a C++ program where Device is a base class with brand and model. Derive Features for device specifications and Warranty for warranty details. Create a derived class Smartphone to combine and display all information.

**Polymorphism**

**Compile-Time Polymorphism:**

1. **Function Overloading for Area Calculation**  
   Write a C++ program that overloads a function area() to calculate the area of a rectangle, a circle, and a triangle.
2. **Operator Overloading for Complex Numbers**  
   Write a C++ program to overload the + operator for a class Complex to add two complex numbers.
3. **Function Overloading for Conversion**  
   Write a C++ program to overload a function convert() to convert temperature from Celsius to Fahrenheit and vice versa.
4. **Template Function for Maximum Value**  
   Write a C++ program to create a template function maxValue() that works with integers, floating-point numbers, and strings to find the maximum value.

**Run-Time Polymorphism:**

1. **Shape Area Calculation with Virtual Functions**  
   Write a C++ program with a base class Shape having a virtual function area(). Derive classes Rectangle, Circle, and Triangle to calculate and display their respective areas.
2. **Bank Account Types**  
   Write a C++ program with a base class Account that has a virtual function calculateInterest(). Derive classes SavingsAccount and CurrentAccount to calculate interest differently.
3. **Vehicle and Fuel Efficiency**  
   Write a C++ program with a base class Vehicle having a virtual function fuelEfficiency(). Derive classes Car, Bike, and Truck to implement different mileage calculations.
4. **Employee Salary Calculation**  
   Write a C++ program with a base class Employee having a virtual function calculateSalary(). Derive classes PermanentEmployee and ContractEmployee to calculate salaries differently.
5. **Appliance Power Consumption**  
   Write a C++ program with a base class Appliance having a virtual function powerUsage(). Derive classes Fan, AC, and Heater to implement specific power consumption calculations.
6. **Animal Sounds**  
   Write a C++ program with a base class Animal having a virtual function makeSound(). Derive classes Dog, Cat, and Cow to implement specific animal sounds.
7. **Transport Fare Calculation**  
   Write a C++ program with a base class Transport having a virtual function fare(). Derive classes Bus, Train, and Flight to calculate fares differently.
8. **Inventory System for Different Products**  
   Write a C++ program with a base class Product having a virtual function getDetails(). Derive classes Electronic, Grocery, and Clothing to display respective details.

**Abstract Class Problems:**

1. **Shape Area Calculation**

Write a C++ program with an abstract class Shape that has a pure virtual function area(). Derive classes Circle, Rectangle, and Triangle to implement the area() function and calculate the area for each shape.

1. **Employee Salary Calculation**

Write a C++ program with an abstract class Employee that has a pure virtual function calculateSalary(). Derive classes PermanentEmployee and ContractEmployee to implement the calculateSalary() function based on different salary structures.

1. **Animal Sounds**

Write a C++ program with an abstract class Animal that has a pure virtual function makeSound(). Derive classes Dog, Cat, and Cow to implement the makeSound() function for respective animal sounds.

1. **Vehicle Mileage Calculation**

Write a C++ program with an abstract class Vehicle that has a pure virtual function mileage(). Derive classes Car, Bike, and Truck to calculate mileage differently based on their specific attributes.

1. **Appliance Power Consumption**

Write a C++ program with an abstract class Appliance that has a pure virtual function powerUsage(). Derive classes Fan, AC, and Heater to implement the powerUsage() function to calculate power consumption based on their specific requirements.

**STATIC DATA MEMBER OR FUNCTION**

**01. Counting Active Objects:**

Write a C++ program to use a static data member to count the number of active objects in a class, updating the count in the constructor and destructor.

**02. Generating Unique IDs:**

Write a C++ program to use a static data member to generate unique IDs for each object of a class.

**03. Sharing Configuration Data:**

Write a C++ program to use a static data member to store and modify application-wide configuration data shared by all objects of a class.

**04. Using Static Members with Constants:**

Write a C++ program to calculate the area of a circle using a static constant data member in the class.

**05. Limiting Object Creation:**

Write a C++ program to use a static data member to limit the maximum number of objects that can be created from a class.

**06. Tracking Total Value Across Objects:**

Write a C++ program to use a static data member to maintain a cumulative total of a specific variable for all objects of a class.

**07. Managing Singleton Pattern:**

Write a C++ program to implement a Singleton pattern using static data members and static functions.

**08. Global Access to a Counter:**

Write a C++ program to use a static data member to track and display the total number of times a particular function has been called across all instances of a class.

**09. Accessing a Private Static Member:**   
Write a C++ program to access a private static data member using public static member functions without creating any objects.

**10.** **Tracking Maximum Value :**

Write a C++ program to use a static data member to track and store the maximum value entered into objects of a class.

**11.** **Using Static Data for Discounts:**

Write a C++ program to use a static data member to store a discount percentage that is common for all customers of a shopping system.

**12.** **Tracking Login Sessions:**

Write a C++ program to use a static data member to count and display the total number of user login sessions in a system.

**13.** **Using Static Data in a Voting System:**

Write a C++ program to create a simple voting system where a static data member is used to track the total number of votes received.

**14. Storing Shared PI Value for Multiple Shapes:**

Write a C++ program to use a static data member to store the value of PI shared among multiple classes representing different geometric shapes, like Circle and Cylinder.

**15. Counting Specific Types of Objects:**

Write a C++ program to use static data members to count the number of objects created for specific subclasses in an inheritance hierarchy.

**EXCEPTION HANDLING**

**01. GPA Validation:**

Prompt the user to enter their GPA. If the GPA is less than 0 or greater than 4.0, throw an exception with the message "Invalid GPA."

**02. Array Sum:**

Create a program to calculate the sum of an array. Prompt the user to enter the size of the array and ensure it is positive. If the user enters a negative size, throw an exception with the message "Array size cannot be negative."

**03. Password Strength**

Prompt the user to enter a password. If the password length is less than 6 characters, throw an exception and display "Password too short."

**04. Multiplication Table**

Create a program that generates a multiplication table for a given number. If the user enters a negative number, throw an exception with the message "Negative numbers are not allowed."

**05. Voting Eligibility**

Ask the user to enter their age. If the age is below 18, throw an exception with the message "Not eligible to vote."

**06. Prime Number Checker:**

Prompt the user to enter a positive integer. If the user enters a number less than 2, throw an exception with the message "Number must be greater than or equal to 2."

**07. Temperature Range:**

Ask the user to input a temperature. If the temperature is less than -100°C or greater than 150°C, throw an exception with the message "Temperature out of realistic range."

**08. Student Class:**

Create a Student class with attributes for name and grade. Add a method setGrade that throws an exception if the grade is less than 0 or greater than 100, with the message: "Invalid grade entered." Test by creating a Student object and setting valid and invalid grades.

**09. Car Class:**

Write a Car class with attributes fuel and mileage. Add a method drive(distance) that decreases the fuel based on the mileage. If the required fuel exceeds the current fuel, throw an exception with: "Not enough fuel to drive this distance." Test the program with different distances.

**10. Account Class with Transfer Function:**

Create an Account class with attributes balance and accountNumber. Add a method transfer(amount, targetAccount) that throws an exception if the amount is greater than the balance or less than 0, with messages like:

* "Insufficient balance for transfer."
* "Transfer amount must be positive."

Test by creating two accounts and trying valid and invalid transfers.

**TEMPLATE**

**Function Template:**

1. **Find Maximum of Two Values**  
   Write a C++ program to create a function template maxValue() that takes two parameters of the same type and returns the maximum of the two values.
2. **Array Sum**  
   Write a C++ program to create a function template arraySum() that calculates the sum of elements in an array for both integer and floating-point types.
3. **Swap Two Variables**  
   Write a C++ program to create a function template swapValues() to swap two variables of any data type.
4. **Matrix Multiplication**  
   Write a C++ program to create a function template matrixMultiply() to perform matrix multiplication for matrices of integer and floating-point types.
5. **Find Minimum Element in an Array**  
   Write a C++ program to create a function template findMin() that finds the minimum value in an array of any type.

**Class Template:**

1. **Stack Implementation**  
   Write a C++ program to create a class template Stack to perform standard stack operations (push, pop, and display) for different data types.
2. **Queue Implementation**  
   Write a C++ program to create a class template Queue to implement a queue with operations like enqueue, dequeue, and display for integer and string types.
3. **Pair Container**  
   Write a C++ program to create a class template Pair that stores two values of any type and provides methods to display and swap these values.
4. **Dynamic Array**  
   Write a C++ program to create a class template DynamicArray to dynamically allocate memory for an array of any data type and perform operations like insertion, deletion, and display.
5. **Calculator Class**  
   Write a C++ program to create a class template Calculator to perform basic arithmetic operations (addition, subtraction, multiplication, and division) for integer and floating-point numbers.

**PRACTICE( BEFORE EXAM)**

**01**. **Shape Hierarchy (Inheritance, Polymorphism)**

* Create a base class Shape with pure virtual functions getArea() and getPerimeter().
* Derive classes like Circle, Rectangle, Triangle, and Square from Shape.
* Implement the area and perimeter calculation methods in each derived class.
* Use polymorphism to calculate and display the area and perimeter of different shapes dynamically.

**02**. **Vehicle Inheritance (Inheritance, Encapsulation)**

* Create a base class Vehicle with attributes like make, model, and year.
* Derive classes like Car, Motorcycle, and Truck from Vehicle.
* Implement member functions in the derived classes to handle specific functionalities (like loading capacity for trucks, fuel efficiency for cars).
* Demonstrate accessing both base and derived class members.

**03**. **Banking System (Abstraction, Polymorphism)**

* Create an abstract class Account with virtual functions deposit(), withdraw(), and calculateInterest().
* Derive classes SavingsAccount and CurrentAccount from Account.
* Implement the functions differently in each class, considering features like overdraft for current accounts.
* Use a menu-driven program to perform transactions on both types of accounts.

**04**. **Employee Management System (Inheritance, Encapsulation)**

* Create a base class Employee with data members like name, employeeID, and salary.
* Derive two classes Manager and Engineer from Employee.
* Implement methods to calculate bonuses, where the bonus for a Manager is based on team performance and for an Engineer based on project success.
* Use getter and setter methods to access private data members.

**05**. **Library Management System (Inheritance, Polymorphism, Encapsulation)**

* Create a base class LibraryItem with attributes like title, author, and publicationYear.
* Derive classes Book, Magazine, and DVD from LibraryItem.
* Implement virtual functions to calculate the late return fees differently for each type of item.
* Demonstrate dynamic dispatch using pointers to LibraryItem.

**06**. **University Grading System (Abstraction, Encapsulation)**

* Create an abstract class Course with attributes like courseName, credits, and a pure virtual function calculateGrade().
* Derive classes TheoryCourse and PracticalCourse from Course.
* Implement calculateGrade() in both classes based on different grading rules (e.g., exams for theory courses, projects for practical courses).
* Use a vector of pointers to manage different course types and display student grades.

**07**. **Animal Hierarchy (Inheritance, Polymorphism)**

* Create a base class Animal with virtual functions sound() and move().
* Derive classes like Dog, Bird, and Fish from Animal.
* Implement different sound() and move() behaviors for each derived class (e.g., Dog barks, Bird chirps, Fish swims).
* Use polymorphism to handle multiple types of animals in an array of pointers.

**08**. **E-Commerce System (Encapsulation, Abstraction)**

* Create an abstract class Product with virtual functions getPrice(), getDiscount(), and getDetails().
* Derive classes Electronics, Clothing, and Books from Product.
* Implement different pricing and discount mechanisms in each class.
* Simulate an online store where a customer can view the product details and final prices after discounts.

**09**. **School Management System (Inheritance, Polymorphism)**

* Create a base class Person with data members like name, age, and address.
* Derive classes Student, Teacher, and Staff from Person.
* Implement specific functionalities for each derived class (e.g., Student has grades, Teacher has subjects, Staff has work schedules).
* Use polymorphism to print details of various persons involved in the school.

**10**. **Online Payment System (Abstraction, Polymorphism)**

* Create an abstract class Payment with virtual functions processPayment() and getPaymentDetails().
* Derive classes CreditCardPayment, DebitCardPayment, and PayPalPayment from Payment.
* Implement different payment processing methods for each type of payment.
* Demonstrate polymorphism by processing payments of different types dynamically.

**11**. **Online Ticket Booking System (Inheritance, Polymorphism)**

* Create a base class **Ticket** with attributes like **ticketID**, **price**, and virtual functions **displayDetails**() and **calculateTotal**().
* Derive classes **MovieTicket**, **TrainTicket**, and **EventTicket** from Ticket.
* Implement different discount calculations and details for each ticket type.
* Demonstrate polymorphism by creating an array of Ticket pointers to process different ticket types dynamically.

**12**. **Smart Home System (Inheritance, Encapsulation, Polymorphism)**

* Create a base class **SmartDevice** with attributes like **deviceName** and virtual functions **turnOn**(), **turnOff**(), and **displayStatus**().
* Derive classes like **SmartLight**, **SmartThermostat**, and **SmartSpeaker** from **SmartDevice**.
* Implement different behaviors for each device (e.g., dimming for lights, temperature control for thermostats).
* Simulate controlling a smart home with a list of devices and their actions.

**13**. **Vehicle Rental Service (Inheritance, Polymorphism)**

* Create a base class Vehicle with attributes like **registrationNumber**, **rentPerDay**, and a virtual function **calculateRental**(int days).
* Derive classes Car, Bike, and Truck from Vehicle.
* Implement specific rental cost calculations for each vehicle type.
* Use polymorphism to manage rentals dynamically based on the vehicle type.

**14**. **E-Learning Platform (Abstraction, Polymorphism, Encapsulation)**

* Create an abstract class Course with attributes like **courseID**, courseName, and pure virtual functions **enrollStudent**() and **displayCourseDetails**().
* Derive classes **FreeCourse** and **PaidCourse** from Course.
* Implement different enrollment processes and course details display logic for free and paid courses.
* Create an interface Certificate with a function **generateCertificate**() and implement it in the derived classes.

**15**. **Hospital Management System (Abstraction, Polymorphism)**

* Create an abstract class Person with attributes like name, age, and virtual functions **getDetails**() and **displayDetails**().
* Derive classes Doctor, Patient, and Staff from Person.
* Add specific attributes for each derived class, such as specialization for doctors, illness for patients, and department for staff.
* Use a menu-driven program to manage hospital personnel.

**16**. **Pure Virtual Function for Graphics**

Implement an abstract class **Drawable** with a pure virtual function **draw**().

Derive **Circle** and **Square** classes to implement **draw**().

**Task**: Use polymorphism to call the **draw**() function on an array of **Drawable** objects.