Topic Wise Problems List for OOP.

Week1	Lab1	Creating class, Using different access specifiers of class members, methods to manipulate private members and creating objects of class. Defining constructor (parameterized and non-parameterized) of a class to initialize data members of an object and destructor to free memory.

Problems:

1. Write a C++ class with a private data member and a public member function to display the value of that data member.

2. Implement a C++ class with private data members and public member functions to perform addition and subtraction operations on those data members.

3. Create a C++ class with private data members and a public member function that calculates the average of two numbers.

4. Develop a C++ class with private data members and public member functions to find the maximum and minimum values among those data members.

5. Design a C++ class with private data members and public member functions to calculate the factorial of a given number.

6. Implement a C++ class with a private data member and a public member function that checks whether the given number is prime or not.

7. Create a C++ class with private data members and public member functions to calculate the square and cube of a given number.

8. Develop a C++ class with private data members and public member functions to check whether the given string is a palindrome or not.

9. Design a C++ class with private data members and public member functions to reverse the elements of an integer array.

10. Implement a C++ class with private data members and public member functions to calculate the area and perimeter of a rectangle.

Week2	Lab2	Passing and Returning objects to and from function respectively. Object pointer, inconsistency in implicit call to destructor by copy of an object that shares the same memory location. **Lab Performance Evaluation.**

Problems:

1. Write a C++ function that takes two objects of a class as parameters and returns the object with the larger value.

2. Implement a C++ function that receives an object pointer as a parameter and updates the object's data members based on user input.

3. Create a C++ class with a dynamic memory allocation in the constructor. Write a function that takes an object of this class as a parameter, deletes the memory, and sets the pointer to nullptr.

4. Develop a C++ program that demonstrates the issue of inconsistency in implicit calls to the destructor when two objects share the same memory location.

5. Design a C++ function that receives an object as a parameter and creates a new object, copying the values of the original object's data members, and returns the new object.

Week3	Lab3	Overload constructor to initialize objects in different ways. Creating and Using copy constructor. **Lab Performance Evaluation.**

Problems:

1. Write a C++ class that represents a Circle. Overload the constructor to initialize the Circle object with a default radius of 1.0 or a user-defined radius.

2. Implement a C++ class called Rectangle that represents a rectangle. Overload the constructor to initialize the Rectangle object with a default length and width of 1.0 or user-defined values.

3. Create a C++ class called Date that represents a date. Overload the constructor to initialize the Date object with the current date or user-defined day, month, and year values.

4. Develop a C++ program that demonstrates the use of a copy constructor to create a deep copy of an object of a class called Book.

5. Design a C++ class called Time that represents the time of day (hours, minutes, and seconds). Overload the constructor to initialize the Time object with the current time or user-defined hour, minute, and second values.

Week4	Lab4	Friend functions to access private members of a class.

Problems:

1. Create a C++ class called **Rectangle** with private data members **length** and **width**. Define a friend function **calculateArea()** outside the class that calculates and returns the area of the rectangle.

2. Implement a C++ class called **BankAccount** with a private data member **balance**. Define a friend function **withdraw(BankAccount& account, double amount)** outside the class that allows withdrawing the specified amount from the account.

3. Design a C++ class called **Employee** with private data members **name** and **salary**. Define a friend function **displayEmployeeInfo(const Employee& employee)** outside the class that displays the name and salary of an employee.

Week5	Lab5	Implementation of function overloading for different data types. **Lab Performance Evaluation.**

Problems:

1. Create a C++ function called **sum()** that takes two integers as parameters and returns their sum. Overload the **sum()** function to handle floating-point numbers and return their sum.

2. Implement a C++ class called **Calculator** with a member function **multiply()** that takes two integers as parameters and returns their product. Overload the **multiply()** function to handle floating-point numbers and return their product.

3. Design a C++ program that defines a function called **printArray()** that takes an array of integers and its size as parameters and prints the elements of the array. Overload the **printArray()** function to handle an array of characters and print each character.

Week6	Lab6	Default argument in function definitions and prototypes.

Problems:

1. Write a C++ function called **greet**() that takes a string parameter **name** and an integer parameter **count**. The function should print a greeting message **count** number of times. Provide a default value of 1 for the **count** parameter.

2. Implement a C++ class called **Rectangle** with a member function **calculateArea**() that takes two optional parameters: **length** and **width**. If the **length** and **width** parameters are not provided, the function should assume default values of 1.0. The function should calculate and return the area of the rectangle.

3. Design a C++ program that defines a function called **printMessage**() that takes a string parameter **message** and an optional integer parameter **repeatCount**. If the **repeatCount** parameter is not provided, the function should assume a default value of 1. The function should print the **message** **repeatCount** number of times.

Week8	Lab7	Operator overloading – binary operators (arithmetic operators, relational, logical)

Problems:

1. Create a C++ class called **ComplexNumber** to represent complex numbers. Overload the binary + operator to add two complex numbers.

2. Implement a C++ class called **Point** to represent 2D points. Overload the binary == operator to compare two points for equality.

3. Design a C++ class called **Fraction** to represent fractions. Overload the binary / operator to perform division of two fractions.

Week9	Lab8	Unary operator. Assignment operator overloading. Friend operator function. **Lab Performance Evaluation.**

Problems:

1. Create a C++ class called **Counter** to represent a simple counter. Overload the unary ++ operator to increment the counter by 1.

2. Implement a C++ class called **Person** to represent a person's name. Overload the assignment = operator to assign the name of one person object to another person object.

3. Design a C++ class called **Vector** to represent a 2D vector. Overload the friend binary * operator to perform scalar multiplication between a vector and a scalar value.

Week10	Lab9	Introduction to inheritance, access specifiers for deriving a base class, protected members. Multiple inheritance i.e. deriving multiple base classes. Order of constructor and destructor call in inheritance hierarchy.

Problems:

1. Create a base class called **Shape** with a protected member variable **color**. Derive a class called **Rectangle** from **Shape** and implement a member function **getColor**() in **Rectangle** that returns the color of the rectangle.

2. Implement a base class called **Vehicle** with private member variables **model** and **year**. Derive a class called **Car** from **Vehicle** and another class called **Motorcycle** from **Vehicle**. Implement member functions **getModel**() and **getYear**() in both derived classes to access the model and year of the respective vehicles.

3. Design a class hierarchy for an online shopping system. Create a base class called **Product** with protected member variables **name** and **price**. Derive two classes, **Electronics** and **Clothing**, from **Product**. Implement member functions **getProductName**() in both derived classes to access the name of the electronics and clothing products.

Week11	Lab10	Parameter passing from derived class to base class for simple and multiple inheritances. Virtual base class implementation. **Lab Performance Evaluation.**

Problems:

1. Create a base class called **Animal** with a member function **displayInfo**() that displays "I am an animal". Derive a class called **Dog** from **Animal** and override the **displayInfo**() function to display "I am a dog". Pass a parameter from the derived class to the base class constructor to set the animal's name and display it along with the animal type.

2. Implement a base class called **Shape** with a protected member variable **color** and a pure virtual member function **displayArea**(). Derive two classes, **Rectangle** and **Circle**, from **Shape** and implement the **displayArea**() function in each derived class to calculate and display the area of the respective shape. Pass parameters from the derived classes to the base class constructor to set the shape's dimensions.

3. Design a class hierarchy for a video game. Create a base class called **GameObject** with a virtual member function **displayInfo**(). Derive two classes, **Character** and **Enemy**, from **GameObject**. Override the **displayInfo**() function in each derived class to display the specific information about the character and enemy. Use a virtual base class to avoid ambiguity when deriving from both **Character** and **Enemy**.

Week12	Lab11	Pointer to derived classes to access inherited members. Introducing virtual function for run-time polymorphism.

Problems:

1. Create a base class called **Shape** with a virtual function **displayArea**(). Derive two classes, **Rectangle** and **Circle**, from **Shape** and implement the **displayArea**() function in each derived class to calculate and display the area of the respective shape. Use a pointer to the base class to access the **displayArea**() function for different shapes.

2. Design a class hierarchy for a banking system. Create a base class called **Account** with virtual functions **deposit**() and **withdraw**(). Derive two classes, **SavingsAccount** and **CheckingAccount**, from **Account** and implement the **deposit**() and **withdraw**() functions in each derived class to perform the corresponding operations. Use a pointer to the base class to perform deposit and withdrawal operations on different types of accounts.

3. Implement a base class called **Animal** with a virtual function **makeSound**(). Derive two classes, **Cat** and **Dog**, from **Animal** and override the **makeSound**() function in each derived class to produce the respective animal sounds. Use a pointer to the base class to make different animals make their sounds.

Week13	Lab12	Using generic functions and generic classes e.g. templates. Exception handling e.g. dividing by zero in a fraction. **Lab Performance Evaluation.**

Problems:

1. Implement a generic function called **swapValues**() that swaps the values of two variables of any type using templates. Test the function with different types, such as integers, floats, and characters.

2. Create a generic class called **Stack** using templates that can store elements of any type. Implement functions to push an element onto the stack, pop an element from the stack, and display the elements in the stack. Handle the exception of popping from an empty stack.

3. Implement a class called **Fraction** that represents a fraction with a numerator and denominator. Use exception handling to handle the scenario of dividing by zero. Implement functions to set and

display the fraction, as well as a function to calculate and display the decimal representation of the fraction.

Week14	Lab13	One problem analysis and OOP based solution.

Problem: Library Management System

Analysis:
A library management system is a software application that helps in managing the day-to-day operations of a library. The system should be able to handle tasks such as adding new books, updating book information, managing borrowers, handling book borrowing and returning, and generating reports. To solve this problem, we can use the principles of object-oriented programming (OOP) to design and implement a solution.

OOP-based Solution:

1. Class Design:

- Book: Represents a book in the library with attributes like title, author, ISBN, and availability status.
- Member: Represents a library member with attributes like name, ID, and borrowed books.
- Library: Manages the overall library operations like adding books, managing members, and book borrowing/returning.

2. Implementation:

- The Book class encapsulates the attributes and behavior related to a book. It provides methods to get and set the book details.
- The Member class represents a library member. It contains a vector of borrowed books, allowing a member to borrow multiple books.
- The Library class manages the library operations. It has vectors to store books and members. It provides methods to add books, add members, borrow books, return books, and generate reports.

By implementing the above classes and their respective methods, we can create a library management system that supports adding books, managing members, and handling book borrowing and returning. This OOP-based solution helps in organizing the code structure, promoting code reusability, and ensuring modularity and flexibility.