Assignment cpp

---

1. Display the multiplication table of a number entered by the user.
2. Calculate the sum of all elements in an integer array.
3. Identify the largest number in a given array.
4. Implement linear search to find an element in an array.
5. Implement binary search on a sorted array.
6. Sort an array of integers using the bubble sort algorithm.
7. Sort an array using the selection sort method.
8. Sort an array using the insertion sort technique.
9. Check if a string is a palindrome without using built-in functions.
10. Reverse a string entered by the user.
11. Count the number of vowels and consonants in a given string.
12. Display all prime numbers within a specified range.
13. Find the Greatest Common Divisor (GCD) and Least Common Multiple (LCM) of two numbers.
14. Calculate the sum of digits of a given number.
15. Convert a number (up to 999) into words (e.g., 123 -> "One Two Three").
16. Determine if a given year is a leap year.
17. Convert temperatures between Celsius, Fahrenheit, and Kelvin.

Assignment cpp

---

18. Create a calculator that performs basic operations using switch-case statements.
19. Add two matrices of the same dimensions.
20. Compute the transpose of a given matrix.
21. Calculate the sum of the main and secondary diagonals of a square matrix.
22. Remove all vowels from a given string.
23. Simulate a simple banking system with options to deposit, withdraw, and check balance.

Projects

Library Management System:

- Develop a system to manage books, members, and transactions in a library.

Inventory Management System:

- Create a system to track inventory levels, orders, sales, and deliveries.

Assignment cpp

Student Information System:

- Build a system to manage student records, courses, and grades.

Online Quiz Platform:

- Develop a platform where users can take quizzes and view their scores.

Assignment cpp

Basic OOP Concepts

1. Class and Object Creation

1. **Define a `Person` Class:**
   - Create a `Person` class with attributes like `name`, `age`, and `gender`. Include methods to set and display these attributes.
2. **Rectangle Class:**
   - Design a `Rectangle` class with `length` and `breadth` as private members. Include methods to calculate area and perimeter.
3. **Circle Class:**
   - Implement a `Circle` class with a private member `radius`. Provide methods to compute the area and circumference.
4. **Student Class:**
   - Create a `Student` class with attributes `studentID`, `name`, and `grade`. Include methods to input and display student details.
5. **BankAccount Class:**
   - Develop a `BankAccount` class with `accountNumber`, `accountHolder`, and `balance`.

Assignment cpp

Implement methods to deposit, withdraw, and display account details.

2. Constructors and Destructors

6. **Default and Parameterized Constructors:**
   - Create a `Book` class with attributes `title`, `author`, and `price`. Implement both default and parameterized constructors.
7. **Copy Constructor:**
   - Define a `Point` class with `x` and `y` coordinates. Implement a copy constructor and demonstrate its usage.
8. **Destructor Usage:**
   - Develop a `Resource` class that allocates memory dynamically. Implement a destructor to free the allocated memory.

3. Member Functions and Accessors

9. **Getters and Setters:**
   - Create a `Car` class with private members `brand`, `model`, and `year`. Implement getter and setter methods for each attribute.
10. **Static Members:**

Assignment cpp

- Design a `Counter` class that keeps track of the number of objects created using a static member variable.

---

2. Encapsulation and Access Specifiers

4. Access Control

11. **Private vs. Public Members:**
    - Implement a `Laptop` class with private members `brand`, `model`, and `price`, and public methods to access and modify these members.
12. **Protected Members:**
    - Create a `Device` class with protected members and derive a `Smartphone` class that accesses these members.

5. Encapsulation Practices

13. **Secure Employee Class:**
    - Develop an `Employee` class where sensitive data like `salary` is kept private and can only be modified through specific methods.
14. **Immutable Class:**

Assignment cpp

- Create an `ImmutableString` class where once an object is created, its value cannot be changed.

6. Friend Classes and Functions

15. **Friend Function:**
    - Implement a `Box` class and a friend function that calculates the volume of the box.
16. **Friend Class:**
    - Design two classes, `Engine` and `Car`, where `Engine` is a friend of `Car` and can access its private members.

---

3. Inheritance

7. Single Inheritance

17. **Animal and Dog Classes:**
    - Create a base class `Animal` with a method `makeSound()`. Derive a class `Dog` that overrides `makeSound()`.
18. **Vehicle and Car Classes:**
    - Implement a `Vehicle` class with attributes like `speed` and `capacity`. Derive a `Car` class that adds specific features.

Assignment cpp

---

8. Multiple Inheritance

19. **SmartPhone Inheriting Multiple Classes:**
    - Design classes `Camera` and `Phone`, then create a `SmartPhone` class that inherits from both.
20. **Hybrid Inheritance Example:**
    - Create classes `Person`, `Employee`, and `Manager`, demonstrating multiple inheritance scenarios.

9. Multilevel Inheritance

21. **Class Hierarchy:**
    - Develop a multilevel inheritance structure with classes `Grandparent`, `Parent`, and `Child`.
22. **University Structure:**
    - Implement classes `University`, `Department`, and `Professor` to showcase multilevel inheritance.

10. Hierarchical Inheritance

23. **Shape Inheritance:**
    - Create a base class `Shape` and derive multiple classes like `Circle`, `Square`, and `Triangle` from it.
24. **Device Inheritance:**

Assignment cpp

- Design a `Device` class with subclasses `Laptop`, `Tablet`, and `Smartphone`.

11. Hybrid Inheritance

25. **Complex Inheritance Structure:**
    - Implement a hybrid inheritance model combining multiple and multilevel inheritance with classes like `Base`, `Derived1`, `Derived2`, and `Derived3`.

12. Inheritance Best Practices

26. **Avoiding the Diamond Problem:**
    - Create a diamond inheritance scenario and resolve it using virtual inheritance.
27. **Using `override` Keyword:**
    - Demonstrate the use of the `override` keyword in derived classes to prevent accidental method overriding.

---

4. Polymorphism

13. Compile-Time Polymorphism

28. **Function Overloading:**

Assignment cpp

- Implement a `print` function that is overloaded to handle different data types (e.g., `int`, `double`, `string`).

29. **Operator Overloading:**
    - Overload the `+` operator for a `Complex` number class to add two complex numbers.
30. **Constructor Overloading:**
    - Create a `Rectangle` class with multiple constructors to initialize objects in different ways.

14. Run-Time Polymorphism

31. **Virtual Functions:**
    - Develop a base class `Shape` with a virtual method `draw()`. Derive classes `Circle` and `Square` that override `draw()`.
32. **Abstract Classes:**
    - Implement an abstract class `Employee` with a pure virtual function `calculateSalary()`, and derive `FullTime` and `PartTime` classes.
33. **Dynamic Binding:**
    - Demonstrate dynamic binding using base class pointers to derived class objects.

15. Pure Virtual Functions and Interfaces

Assignment cpp

---

34. **Interface Implementation:**
    - Create an interface `Drawable` with a pure virtual function `draw()`, and implement it in classes `Circle` and `Rectangle`.
35. **Multiple Interfaces:**
    - Design a class that implements multiple interfaces, such as `Printable` and `Scannable`.

16. Polymorphism Best Practices

36. **Using `final` Keyword:**
    - Show how to prevent further inheritance or overriding by using the `final` keyword in classes and methods.
37. **Avoiding Slicing:**
    - Explain and demonstrate object slicing in inheritance and how to prevent it using pointers or references.

---

5. Abstraction and Interfaces

17. Abstract Classes

38. **Abstract Base Class:**
    - Create an abstract class `Appliance` with a pure virtual function `operate()`, and derive classes like

Assignment cpp

`WashingMachine` and `Refrigerator` that implement `operate()`.

18. Interface Classes

39. **Multiple Interfaces Implementation:**
    - Design an interface `Serializable` with a method `serialize()` and `Deserializable` with `deserialize()`. Implement them in a `User` class.

19. Data Hiding and Abstraction

40. **Secure Data Management:**
    - Develop a `SecureData` class that hides sensitive information and provides methods to access and modify it securely.

20. Abstract Factories

41. **Factory Design Pattern:**
    - Implement the Factory Design Pattern to create objects of different classes based on input parameters.

21. Template Abstraction

42. **Generic Classes:**

Assignment cpp

- Create a template class `Container` that can hold objects of any type, demonstrating abstraction through templates.

---

6. Advanced OOP Concepts

22. Operator Overloading

43. **Overloading Comparison Operators:**
    - Overload the `==` and `!=` operators for a `Student` class to compare two student objects based on their IDs.
44. **Stream Insertion and Extraction Operators:**
    - Overload the `<<` and `>>` operators for a `Book` class to enable easy input and output of book details.

23. Exception Handling in OOP

45. **Custom Exceptions:**
    - Create custom exception classes for handling errors in a `BankAccount` class, such as `InsufficientFundsException`.
46. **Exception Safe Classes:**

Assignment cpp

- Design a `FileHandler` class that properly handles exceptions during file operations to prevent resource leaks.

24. Smart Pointers and Resource Management

47. **Using `std::unique_ptr`:**
    - Implement a class that manages dynamic memory using `std::unique_ptr` to ensure proper resource management.
48. **Using `std::shared_ptr`:**
    - Create interconnected objects using `std::shared_ptr` and demonstrate reference counting.

25. Move Semantics and Rvalue References

49. **Move Constructor and Move Assignment:**
    - Develop a `Vector` class that implements move semantics to optimize performance during object transfers.
50. **Optimizing with `std::move`:**
    - Show how to use `std::move` in a class that contains large data members to enhance efficiency.

Assignment cpp

---

7. Design Patterns and Best Practices

26. Singleton Pattern

51. **Implement Singleton:**
    - Create a `Logger` class following the Singleton design pattern to ensure only one instance exists.

27. Factory Method Pattern

52. **Shape Factory:**
    - Implement a Factory Method to create different `Shape` objects (`Circle`, `Square`, etc.) based on input.

28. Observer Pattern

53. **Event System:**
    - Design an event system using the Observer pattern where observers can subscribe to events emitted by a subject.

29. Strategy Pattern

54. **Sorting Strategies:**
    - Implement different sorting algorithms (e.g., QuickSort, MergeSort) using the Strategy pattern to allow interchangeable sorting strategies.

30. Command Pattern

Assignment cpp

---

55. **Undo Functionality:**
    - Develop a text editor that uses the Command pattern to implement undo and redo functionalities.

31. Adapter Pattern

56. **Legacy System Integration:**
    - Create an Adapter that allows a new `MediaPlayer` class to work with legacy `AdvancedMediaPlayer` interfaces.

32. Decorator Pattern

57. **Stream Decorators:**
    - Implement decorators to add functionalities like buffering and encryption to a basic data stream class.

33. Composite Pattern

58. **Graphic Objects:**
    - Design a composite structure for graphic objects where individual shapes and groups of shapes can be treated uniformly.

34. Facade Pattern

59. **Home Theater System:**

Assignment cpp

- Create a Facade for a home theater system that simplifies the interface for controlling multiple components like TV, DVD player, and speakers.

35. MVC Architecture

60. **Simple MVC Application:**
    - Develop a basic Model-View-Controller (MVC) application to separate concerns in a C++ program.

Additional Advanced Topics

36. Templates and Generic Programming

61. **Template Inheritance:**
    - Create a base template class `Storage<T>` and derive classes like `Storage<int>` and `Storage<string>` with specialized functionalities.

37. Multiple Inheritance and Virtual Inheritance

62. **Diamond Problem Resolution:**
    - Implement multiple inheritance with virtual inheritance to resolve the diamond problem in a class hierarchy.

38. RTTI and Type Identification

Assignment cpp

---

63. **Dynamic Casting:**
    - Use `dynamic_cast` to safely convert pointers or references within an inheritance hierarchy.

39. Virtual Destructors

64. **Polymorphic Destruction:**
    - Demonstrate the importance of virtual destructors in a base class when dealing with dynamic polymorphism.

40. Abstract Data Types

65. **Abstract Stack Implementation:**
    - Design an abstract `Stack` class and implement it using different underlying data structures like arrays and linked lists.

41. Composition vs. Inheritance

66. **Car and Engine Classes:**
    - Compare and implement `Car` as a class that inherits from `Engine` versus a class that contains an `Engine` object (composition).

42. Delegation

67. **Task Assignment:**

Assignment cpp

- Implement a `Manager` class that delegates tasks to `Worker` classes using delegation principles.

43. Interfaces vs. Abstract Classes

68. **Shape Interfaces:**
    - Compare the use of pure abstract classes (interfaces) versus concrete abstract classes in designing a `Shape` hierarchy.

44. Encapsulation of Collections

69. **Library Class:**
    - Design a `Library` class that encapsulates a collection of `Book` objects, providing methods to add, remove, and search for books.

45. Design by Contract

70. **Validated Setters:**
    - Implement setter methods in a `User` class that validate input data before setting private members, following the Design by Contract principle.