👇

□ 1. Introduction to Object-Oriented Programming

- What is Object-Oriented Programming (OOP)?
- Difference between **Procedure-Oriented** and **Object-Oriented** Programming
- Advantages of OOP
- Key OOP concepts:
  - **Objects**
  - **Classes**
  - **Encapsulation**
  - **Abstraction**
  - **Inheritance**
  - **Polymorphism**
  - **Message Passing**

□ 2. Classes and Objects

- Defining a class
- Creating objects
- Accessing class members
- Member functions (inside & outside class definition)
- Private, Public, Protected members
- Scope Resolution Operator `::`
- Inline functions
- `this` pointer
- Static data members and static member functions

⚙ 3. Constructors and Destructors

- Default constructor
- Parameterized constructor
- Copy constructor
- Constructor overloading
- Constructor with default arguments
- Destructor and its use
- Dynamic constructor and memory allocation

□ 4. Inheritance (Reusability)

- Concept of inheritance
- Base and Derived classes
- Types of inheritance:
  - Single inheritance
  - Multiple inheritance
  - Multilevel inheritance
  - Hierarchical inheritance
  - Hybrid inheritance
- Constructor and destructor in inheritance
- Protected access specifier
- Virtual base classes (to handle ambiguity)

🔁 5. Polymorphism

(a) Compile-Time Polymorphism

- Function overloading
- Operator overloading
- Rules for overloading
- Friend functions and friend classes

(b) Runtime Polymorphism

- Virtual functions
- Pure virtual functions and Abstract classes
- Function overriding
- Base class pointer and Derived class object
- Virtual destructors

🔒 6. Encapsulation and Abstraction

- Data hiding
- Getters and setters
- Abstract classes
- Interfaces using abstract base classes
- Real-world examples

🎁 7. Dynamic Memory Management

- `new` and `delete` operators
- Dynamic arrays and objects

- Shallow copy vs Deep copy
- Copy constructors in dynamic allocation

♾️ 8. Operator Overloading

- Overloading unary and binary operators
- Overloading using member and friend functions
- Rules and limitations of operator overloading
- Overloading `<<` and `>>` for I/O operations

🧬 9. Templates

- Function templates
- Class templates
- Template specialization
- Applications of templates (e.g., generic stack or list)

🚨 10. Exception Handling

- Introduction to exceptions
- `try`, `catch`, and `throw` keywords
- Multiple catch blocks
- Catching all exceptions
- Re-throwing exceptions
- Standard exception classes

💾 11. File Handling in C++

- Streams: ifstream, ofstream, fstream
- Opening and closing files
- Reading and writing text/binary files
- File modes and flags
- Random access in files (seekg, seekp, tellg, tellp)

🧠 12. Advanced OOP Concepts

- Pointers to objects

- `this` pointer usage
- Object slicing
- Nested classes
- Composition and Aggregation
- Namespaces
- Inline functions
- Static vs Dynamic binding

□ 13. Standard Template Library (STL) – Optional but Important

- Overview of STL
- Containers (vector, list, map, set, etc.)
- Iterators
- Algorithms (`sort`, `find`, etc.)
- Functors (Function objects)

□ 14. Mini Project Ideas

- Bank Management System
- Library Management System
- Student Record System
- Employee Payroll System
- Inventory Management System

Would you like me to make this syllabus into a **printable PDF or DOC file** for your C++ OOP preparation (for college or self-study)?

You said:

□ 2. Classes and Objects Defining a class Creating objects Accessing class members Member functions (inside & outside class definition) Private, Public, Protected members Scope Resolution Operator :: Inline functions this pointer Static data members and static member functions 50 problem statement in one line

ChatGPT said:

Here are **50 one-line C++ problem statements** based on the topic **"Classes and Objects"** 👇

□ C++ Classes and Objects – 50 Problem Statements

1. Define a class `Student` with roll number and name, and display the details.
2. Create a class `Book` to store title, author, and price, and print them.
3. Write a program to define a class `Rectangle` and calculate its area and perimeter.
4. Create a class `Circle` to compute the area using a radius given by the user.
5. Define a class `Employee` with data members `id`, `name`, and `salary`, and display details.
6. Write a program using a class `Car` with model, brand, and mileage attributes.
7. Create a class `BankAccount` to deposit and withdraw money from a balance.
8. Define a class `Complex` to add and subtract two complex numbers.
9. Make a class `Time` with hour and minute, and add two time objects.
10. Create a class `Distance` with feet and inches, and add two distances.
11. Write a program with a class `Calculator` having add, subtract, multiply, and divide methods.
12. Implement a class `Student` with marks of three subjects and calculate the average.
13. Define a class `Triangle` to check if sides form a valid triangle.
14. Create a class `Temperature` to convert Celsius to Fahrenheit.
15. Write a program to show how private and public access specifiers work in a class.
16. Create a class `Box` with length, breadth, and height to find the volume.
17. Define a class `Laptop` and count how many laptop objects are created using a static variable.
18. Create a class `Employee` and use a static function to display company name.
19. Write a class `Person` with name and age, and use inline function to display details.
20. Create a class `Student` with private roll number and public setter and getter functions.
21. Demonstrate use of the `this` pointer to distinguish between local and class variables.
22. Create a class `Account` with deposit and withdraw functions using `this` pointer.
23. Define a class `Rectangle` with inline functions for area and perimeter.
24. Create a class `Counter` that increments a count each time an object is created.
25. Implement a class `Product` to find total cost using price and quantity.
26. Create a class `Car` with engine capacity and fuel type, and display details using inline function.
27. Write a class `ElectricityBill` to calculate total bill based on units consumed.
28. Define a class `Bank` with static data member `rateOfInterest` shared by all accounts.
29. Create a class `Circle` using a static member function to display value of π.
30. Write a class `Person` to show the difference between accessing private and public members.
31. Define a class `Room` and calculate the cost of painting its walls.
32. Write a class `Student` that reads marks of 5 subjects and displays total and average.
33. Create a class `Currency` to add and subtract rupees and paise.
34. Implement a class `Employee` with private salary and a public function to increase it by 10%.
35. Define a class `Rectangle` to compare two rectangles by area.
36. Create a class `BankAccount` and show how to access private balance using a public method.
37. Define a class `Book` with constructor and use inline function to display book details.
38. Write a class `Movie` to store title, director, and rating, and display top-rated movie.
39. Create a class `Date` with day, month, year, and display it in dd/mm/yyyy format.
40. Define a class `Point` to calculate distance between two points using member functions.
41. Write a program with class `Student` that displays total number of students created.

42. Create a class `Temperature` and convert Kelvin to Celsius using inline function.
43. Define a class `Complex` and demonstrate the use of scope resolution operator `::` for defining functions.
44. Write a program with class `Counter` having static member variable to count created objects.
45. Create a class `Car` with a static function that displays total number of cars created.
46. Define a class `Rectangle` and overload its constructor to initialize dimensions.
47. Write a program with class `Student` using `this` pointer to return object reference.
48. Create a class `Circle` with radius as private and function to calculate area as inline.
49. Define a class `Employee` to show the difference between public, private, and protected members.
50. Create a class `Shape` with static data member `objectCount` to track total objects made.

Would you like me to give **input–output examples** and **solutions (code)** for each of these 50 problems next?
I can provide them in a **step-by-step or grouped by concept (e.g., static, this pointer, inline, etc.)** format.

You said:

⚙ 3. Constructors and Destructors Default constructor Parameterized constructor Copy constructor Constructor overloading Constructor with default arguments Destructor and its use Dynamic constructor and memory allocation

ChatGPT said:

Here are **50 one-line C++ problem statements** based on the topic
⚙ **"Constructors and Destructors"** 👇

⚙ C++ Constructors and Destructors – 50 Problem Statements

1. Write a program to demonstrate a **default constructor** that initializes student details.
2. Create a class `Box` that uses a **parameterized constructor** to initialize length, breadth, and height.
3. Define a class `Rectangle` using a **default constructor** to assign default dimensions.
4. Write a class `Circle` with a **parameterized constructor** to initialize radius and calculate area.
5. Implement a **copy constructor** in a class `Student` to copy one object's data into another.
6. Create a class `Car` demonstrating **constructor overloading** for different initialization options.
7. Define a class `Book` with **constructor having default arguments** for price and author.
8. Write a program to demonstrate a **destructor** releasing allocated memory.

9. Create a class `Person` to show how **constructor and destructor** work automatically in object lifetime.
10. Implement a **copy constructor** to clone data from one employee object to another.
11. Write a class `Laptop` that dynamically allocates memory for brand name using a **dynamic constructor**.
12. Define a class `BankAccount` to initialize account details using **constructor overloading**.
13. Write a program to show how **default constructor** and **destructor** execute order in stack objects.
14. Create a class `Complex` with **parameterized constructor** to initialize real and imaginary parts.
15. Implement **constructor overloading** for class `Student` to handle name only or name with roll number.
16. Create a **copy constructor** in class `Book` to duplicate an existing book object.
17. Define a class `Matrix` with **parameterized constructor** to initialize matrix dimensions.
18. Write a class `Car` with **constructor having default arguments** for color and model.
19. Create a class `Point` to initialize coordinates using **parameterized constructor**.
20. Demonstrate **destructor** execution order when multiple objects are created.
21. Write a class `DynamicArray` that allocates memory in a **constructor** and deallocates in **destructor**.
22. Define a class `Employee` using **constructor overloading** to set default or custom salary.
23. Create a class `Student` to show how **copy constructor** prevents shallow copy issues.
24. Write a class `Shape` with **constructor and destructor** printing messages when called.
25. Define a **dynamic constructor** that allocates an integer array and calculates its sum.
26. Implement a class `Book` using **constructor overloading** for hardcopy and eBook editions.
27. Write a program to show **constructor with default argument** initializing employee designation.
28. Create a class `BankAccount` with **copy constructor** copying balance and account type.
29. Define a class `Person` using **constructor overloading** to initialize with or without age.
30. Write a program to illustrate **destructor** freeing memory dynamically allocated for string data.
31. Create a class `Car` where **parameterized constructor** initializes engine capacity and mileage.
32. Write a class `Rectangle` showing **default**, **parameterized**, and **copy constructor** together.
33. Define a class `Time` to add two objects using **copy constructor**.
34. Write a class `Book` demonstrating **constructor overloading** with different numbers of arguments.
35. Create a **dynamic constructor** allocating a float array and calculating average.
36. Implement a **destructor** to release memory used by dynamically allocated character array.
37. Write a class `Student` to show **constructor with default arguments** for marks.
38. Create a class `Laptop` using **constructor overloading** to set brand, RAM, and price.

39. Define a class `Complex` and demonstrate **copy constructor** with deep copy of dynamic data.
40. Write a program to show **constructor overloading** in class `Rectangle` for square or rectangle creation.
41. Create a **dynamic constructor** that initializes an array with user input values.
42. Define a **destructor** to confirm object destruction message after function scope ends.
43. Write a class `Employee` with **parameterized constructor** and **destructor** to show lifetime.
44. Implement a class `BankAccount` that uses **copy constructor** to duplicate an account.
45. Create a **constructor with default arguments** in class `Car` for model name and price.
46. Define a class `Book` and **overload constructor** to initialize either title or title and price.
47. Write a program showing **default constructor**, **parameterized constructor**, and **destructor** sequence.
48. Create a class `Vector` with **dynamic constructor** allocating elements and calculating magnitude.
49. Define a class `Employee` using **constructor overloading** for permanent and temporary employees.
50. Write a program that shows how **destructor** is called when object goes out of scope.

Would you like me to give **input–output examples and solutions (C++ code)** for these 50 constructor–destructor problems next?
I can organize them in sections like **default constructor**, **copy constructor**, **overloading**, etc., for easier study.

You said:

 4. Inheritance (Reusability) Concept of inheritance Base and Derived classes Types of inheritance: Single inheritance Multiple inheritance Multilevel inheritance Hierarchical inheritance Hybrid inheritance Constructor and destructor in inheritance Protected access specifier Virtual base classes (to handle ambiguity)

ChatGPT said:

Here are **50 one-line C++ problem statements** based on the topic
 “Inheritance (Reusability)” 👇

C++ Inheritance – 50 Problem Statements

🌱 Concept of Inheritance / Base and Derived Classes

1. Create a base class `Person` and derived class `Student` to display name and roll number.
2. Define a base class `Vehicle` and derived class `Car` showing inheritance of properties.
3. Write a program where class `Employee` inherits from base class `Person`.
4. Create a base class `Animal` with a derived class `Dog` demonstrating reusability.

5. Implement a base class `Shape` and derived class `Circle` with area calculation.
6. Create a base class `Account` and derived class `SavingsAccount` with interest calculation.
7. Write a program showing how derived class can access base class members.
8. Define a base class `Employee` and derived class `Manager` with additional allowance data.
9. Create a base class `Student` with derived class `GraduateStudent` using constructor initialization.
10. Write a program to show **public**, **protected**, and **private** inheritance differences.

□ Single Inheritance

11. Create a base class `Shape` and derived class `Rectangle` to compute area.
12. Define a base class `Employee` and derived class `Programmer` showing single inheritance.
13. Write a program where class `Animal` is base and `Dog` is derived to display behavior.
14. Implement a single inheritance example where `Book` inherits from `Publication`.
15. Create a base class `Account` and derived class `CurrentAccount` with withdrawal functionality.

□ Multilevel Inheritance

16. Create a class `Person → Employee → Manager` showing multilevel inheritance.
17. Define `Animal → Mammal → Dog` hierarchy showing constructor calls.
18. Write a program `Student → Graduate → ResearchScholar` with degree details.
19. Create a multilevel inheritance chain `Shape → Polygon → Rectangle` to calculate area.
20. Implement `Device → Computer → Laptop` to demonstrate reusability of features.

🌳 Hierarchical Inheritance

21. Create a base class `Shape` and derived classes `Circle` and `Square` for area calculation.
22. Define a base class `Employee` and derived classes `Engineer` and `Manager`.
23. Write a program `Animal` as base class, and `Dog`, `Cat`, `Horse` as derived classes.
24. Create a base class `Student` with derived classes `ArtsStudent` and `ScienceStudent`.
25. Implement a base class `Account` with derived classes `Savings` and `Current`.

⚯ Multiple Inheritance

26. Create a class `Result` derived from `Student` and `Sports` using multiple inheritance.

27. Define classes `Teacher` and `Researcher`, and create a derived class `Professor`.
28. Write a program combining two base classes `Person` and `Worker` into one derived class.
29. Implement multiple inheritance using `MusicStudent` derived from `Student` and `Musician`.
30. Create a class `SmartPhone` derived from `Phone` and `Camera`.

□ Hybrid Inheritance

31. Write a program combining multiple and multilevel inheritance in class hierarchy.
32. Create a hybrid inheritance model `Person → Student` and `Person → Employee → PartTimeStudent`.
33. Implement hybrid inheritance using `Device → Computer` and `Device → Phone → SmartPhone`.
34. Define a hybrid structure combining `Shape → Rectangle` and `Shape → Circle → Cylinder`.
35. Create a hybrid inheritance example involving academic and sports performance of a student.

⚙ Constructor and Destructor in Inheritance

36. Write a program showing order of constructor and destructor calls in single inheritance.
37. Create a program to show constructor–destructor order in multilevel inheritance.
38. Define base and derived classes with parameterized constructors showing call sequence.
39. Write a program demonstrating base constructor calling through initialization list.
40. Create a class hierarchy showing when destructors are called automatically.

🔒 Protected Access Specifier

41. Define a base class `Account` with protected data and a derived class `Savings` accessing it.
42. Create a base class `Student` with protected marks and show derived class accessing them.
43. Write a program to illustrate how protected members are inherited but not publicly accessible.
44. Define `Employee` base class with protected salary accessed by derived class `Manager`.
45. Create `Shape` base class with protected area variable accessed by derived classes.

□ Virtual Base Classes (Ambiguity Handling)

46. Create a diamond inheritance problem and solve it using **virtual base classes**.
47. Write a program `Person → Student` and `Person → Employee → WorkingStudent` using virtual inheritance.
48. Define classes `A`, `B`, and `C` where `C` inherits from both `A` and `B` virtually.
49. Implement a **virtual base class** example to avoid duplicate constructor calls.
50. Write a program showing how **virtual inheritance** resolves ambiguity in multiple paths.

Would you like me to give the **input–output examples and full code solutions** for these 50 inheritance problems next — grouped by type (Single, Multiple, Multilevel, etc.)?

You said:

🔁 5. Polymorphism (a) Compile-Time Polymorphism Function overloading Operator overloading Rules for overloading Friend functions and friend classes (b) Runtime Polymorphism Virtual functions Pure virtual functions and Abstract classes Function overriding Base class pointer and Derived class object Virtual destructors

ChatGPT said:

Here are **50 one-line C++ problem statements** based on the topic
🔁 **Polymorphism** — including both **Compile-Time** and **Runtime** polymorphism 👇

🔁 C++ Polymorphism – 50 Problem Statements

⚙️ (A) Compile-Time Polymorphism

◆ Function Overloading (1–15)

1. Write a program to overload a function `add()` for integers and floats.
2. Create an overloaded function `area()` to find the area of circle, rectangle, and triangle.
3. Write a class `Calculator` with overloaded `sum()` methods for two and three arguments.
4. Implement function overloading for `max()` to find maximum of two and three numbers.
5. Create a class `Display` with overloaded `show()` for integer, double, and string.
6. Write overloaded functions `volume()` for cube, cylinder, and sphere.
7. Implement function overloading for `swap()` to exchange integer and float values.
8. Create overloaded functions `print()` for different data types.
9. Write a class `Student` with overloaded `setData()` for roll number and name.
10. Demonstrate overloading of `area()` for different parameter types (int, float).
11. Write overloaded functions `power()` to compute exponentiation for int and double.

12. Create a program to show function overloading with default arguments.
13. Define overloaded `multiply()` functions to handle both integers and matrices.
14. Write overloaded `compare()` functions for integers, floats, and strings.
15. Implement overloading in a class `Time` for setting time in hours only or hours and minutes.

◆ Operator Overloading (16–30)

16. Overload the `+` operator to add two complex numbers.
17. Overload the `-` operator to subtract two time objects.
18. Write a class to overload the `==` operator to compare two strings.
19. Overload the `++` operator (prefix) to increment an integer data member.
20. Overload the `--` operator (postfix) to decrement an object's value.
21. Overload the `*` operator to multiply two matrices.
22. Overload the `/` operator to divide one fraction object by another.
23. Overload the `<` and `>` operators to compare two employee salaries.
24. Overload the `[]` operator to access elements of an array class.
25. Overload the `=` operator to copy object values manually.
26. Overload the `()` operator to treat object as a function call.
27. Overload the `+=` operator to concatenate two strings.
28. Overload the `<<` and `>>` operators for input and output of class `Student`.
29. Overload the `!` operator to invert a boolean-like object.
30. Write a class to overload both `+` and `*` for a `Vector` class.

◆ Rules for Overloading (31–33)

31. Write a program to show that overloaded functions must differ by parameter type or number.
32. Demonstrate ambiguity error when overloading is done with default arguments incorrectly.
33. Create an example showing that function return type alone cannot distinguish overloaded functions.

◆ Friend Functions and Friend Classes (34–38)

34. Write a friend function to add two objects of different classes.
35. Create two classes `Box` and `Cube` and a friend function to compare their volumes.
36. Write a program where a friend class `Inspector` accesses private data of class `Company`.
37. Define a class `Distance` and a friend function to add two distances.
38. Create a friend function to display private members of class `Account`.

□ (B) Runtime Polymorphism

◆ Virtual Functions (39–43)

39. Write a program with base class `Shape` and derived classes `Circle` and `Rectangle` using virtual `area()` function.
40. Create a base class `Employee` with virtual function `calculateSalary()` overridden by derived classes.
41. Define a virtual function `show()` in class `Base` and override it in class `Derived`.
42. Demonstrate runtime polymorphism using base class pointer calling derived class function.
43. Write a program with base class pointer pointing to derived class object invoking virtual function.

◆ Pure Virtual Functions and Abstract Classes (44–46)

44. Create an abstract base class `Shape` with pure virtual function `area()` implemented in derived classes.
45. Define an abstract class `Animal` with pure virtual `sound()` function and derived classes `Dog` and `Cat`.
46. Implement abstract base class `Account` with derived classes `Savings` and `Current` defining `interest()`.

◆ Function Overriding (47–48)

47. Write a program showing function overriding using base class `Vehicle` and derived class `Car`.
48. Create a base class `Person` with `display()` overridden in derived class `Student`.

◆ Base Class Pointer and Derived Class Object (49)

49. Demonstrate how a base class pointer can call overridden function of derived class using virtual keyword.

◆ Virtual Destructors (50)

50. Write a program showing why destructors should be virtual when deleting derived objects via base class pointer.

Would you like me to give **input/output examples and C++ code solutions** for these 50 polymorphism problems next — grouped as **Compile-Time (Overloading)** and **Runtime (Virtual Functions)**?

You said:

🔒 6. Encapsulation and Abstraction Data hiding Getters and setters Abstract classes Interfaces using abstract base classes Real-world examples

ChatGPT said:

Here are **50 one-line C++ problem statements** based on the topic
🔒 **Encapsulation and Abstraction** 👇

🔒 C++ Encapsulation and Abstraction – 50 Problem Statements

□ 1. Data Hiding (Encapsulation Basics)

1. Create a class `BankAccount` with private balance and public deposit/withdraw methods.
2. Write a class `Student` with private marks and a method to safely update them.
3. Define a class `Employee` with private salary and a method to modify it securely.
4. Implement a class `Car` where speed is private and can be accessed only through functions.
5. Create a class `Rectangle` with private length and width and methods to calculate area.
6. Write a class `Temperature` that hides Celsius and Fahrenheit values and provides conversion methods.
7. Create a class `Person` that hides name and age, with public methods for access.
8. Define a class `Account` that hides account number and balance details.
9. Implement a class `Product` with private cost and discount rate to calculate final price.
10. Write a program demonstrating encapsulation with private data and controlled access.

⚙ 2. Getters and Setters

11. Create a class `Student` with getters and setters for roll number and name.
12. Define a class `Book` with private title and price and methods `setTitle()` and `getPrice()`.
13. Implement a class `Employee` with getter and setter for employee ID and salary.
14. Create a class `Car` with getters and setters for model name and mileage.
15. Write a program for class `BankAccount` with getter for balance and setter for deposit.
16. Define a class `Laptop` with getters and setters for brand and RAM size.

17. Implement a class `Circle` using setter for radius and getter for area.
18. Create a class `Person` with private age and a setter that checks valid age before assignment.
19. Write a class `Rectangle` with methods to set dimensions and get area/perimeter.
20. Define a class `Movie` with setters for title and rating and getter for rating.

□ 3. Abstract Classes (Abstraction Concepts)

21. Create an abstract class `Shape` with pure virtual function `area()` and derive `Circle` and `Rectangle`.
22. Define an abstract class `Employee` with pure virtual function `calculatePay()`.
23. Write a program with abstract class `Animal` and derived classes `Dog` and `Cat` implementing `sound()`.
24. Implement an abstract class `Account` with function `interest()` overridden by derived classes.
25. Create abstract class `Vehicle` with virtual function `startEngine()` implemented in `Car` and `Bike`.
26. Define an abstract class `Appliance` with `powerConsumption()` implemented by `Fan` and `AC`.
27. Write a program to show you cannot create an object of an abstract class directly.
28. Implement abstract class `Shape` with functions `draw()` and `area()` in derived classes.
29. Create an abstract class `Payment` and derived classes `CreditCard` and `UPI` overriding `pay()`.
30. Define an abstract class `Sensor` and derived class `TemperatureSensor` overriding `readValue()`.

⚮ 4. Interfaces using Abstract Base Classes

31. Create an interface `Printable` with pure virtual function `print()` implemented by `Book`.
32. Define an interface `Playable` with method `play()` implemented by `Music` and `Video`.
33. Write a program defining interface `Drawable` implemented by `Circle` and `Rectangle`.
34. Implement interface `Rechargeable` with method `charge()` in `Phone` and `Laptop`.
35. Create an interface `Database` with functions `connect()` and `disconnect()` implemented by `MySQL` and `MongoDB`.
36. Define an interface `Flyable` with pure virtual `fly()` method implemented by `Bird` and `Airplane`.
37. Write an interface `Storable` and implement it in class `File` and `CloudStorage`.
38. Implement multiple interfaces `Readable` and `Writable` in class `Document`.
39. Create interface `Drivable` with `drive()` method implemented by `Car` and `Bike`.
40. Define abstract class `Logger` with pure virtual method `log()` implemented by `ConsoleLogger` and `FileLogger`.

🌍 5. Real-World Examples (Encapsulation + Abstraction Together)

41. Create a class `ATM` that encapsulates user PIN and balance, exposing only withdraw/deposit.
42. Define a class `Hospital` with private patient data and public methods for admission and discharge.
43. Implement a class `SmartPhone` hiding internal hardware info, exposing only user functions.
44. Write a class `ECommerce` encapsulating customer details and abstract method `paymentProcess()`.
45. Create a class `Library` managing private books list with methods to borrow or return books.
46. Define a class `OnlineExam` encapsulating private questions and showing results using abstraction.
47. Write a class `Restaurant` hiding menu details and showing only order interface.
48. Create a class `BankSystem` hiding all account details and providing public transaction interface.
49. Implement a class `AirlineSystem` hiding passenger data and providing ticket booking interface.
50. Write a class `SmartHome` that hides internal devices and exposes control through a single interface.

Would you like me to give **input–output examples and full C++ code** for these 50 encapsulation and abstraction problems next —
or organize them into **sections (Encapsulation, Getters/Setters, Abstract Classes, Interfaces, Real-world Examples)** for step-by-step coding practice?

You said:

🛍️ 7. Dynamic Memory Management new and delete operators Dynamic arrays and objects Shallow copy vs Deep copy Copy constructors in dynamic allocation

ChatGPT said:

Here are **50 one-line C++ problem statements** based on
🛍️ **Dynamic Memory Management** — `new` / `delete`, dynamic arrays/objects, shallow & deep copy, and copy constructors:

◆ A. `new` and `delete` Operators (10 problems)

1. Create a dynamic integer variable using `new` and print its value.
2. Dynamically allocate memory for a float and initialize it.
3. Create a dynamic array of integers of size `n` entered by the user.
4. Allocate an array dynamically and find its sum.
5. Demonstrate deleting dynamically allocated memory to avoid memory leaks.

6. Use `new` to allocate an array of strings and input names from the user.
7. Create a dynamic 2D array using pointers and print it in matrix form.
8. Write a program to allocate and deallocate memory for an object of a class.
9. Dynamically allocate memory for an array of objects and display their data.
10. Demonstrate what happens if you forget to use `delete` (memory leak example).

◆ B. Dynamic Arrays and Objects (10 problems)

11. Create a dynamic array that stores student marks and calculate the average.
12. Take a dynamic array of `n` integers and find the maximum element.
13. Write a class `Student` and create objects dynamically using `new`.
14. Create an array of dynamic objects and input employee details.
15. Write a function to allocate memory dynamically for a character array (string).
16. Implement a program that creates a dynamic array and doubles its size when full.
17. Use `new` and `delete` inside a class to manage memory for string members.
18. Create a class that dynamically allocates memory for an integer array in its constructor.
19. Write a program to dynamically allocate memory for `m x n` matrix and fill it.
20. Create a class that manages a dynamic list of book titles using `new` and `delete`.

◆ C. Shallow Copy vs Deep Copy (10 problems)

21. Demonstrate shallow copy using a class with pointer members.
22. Show how modifying one object affects another in a shallow copy.
23. Implement deep copy using a user-defined copy constructor.
24. Write a program to explain why default copy constructor causes shallow copy.
25. Create a class `String` and implement deep copy for its character array.
26. Demonstrate shallow copy in dynamic array allocation.
27. Compare output of shallow and deep copy in same program.
28. Implement a copy constructor that performs deep copy for a dynamic integer array.
29. Write a class that shows both shallow and deep copy behaviors based on flag.
30. Use deep copy to safely duplicate dynamic objects without shared memory.

◆ D. Copy Constructors in Dynamic Allocation (10 problems)

31. Write a class with a copy constructor to correctly handle dynamic allocation.
32. Show how a custom copy constructor prevents memory corruption.
33. Demonstrate the need of copy constructor in classes using dynamic arrays.
34. Implement a `Vector` class with a copy constructor for dynamic memory.
35. Create a class `Employee` with dynamic name allocation and copy constructor.
36. Write a copy constructor to correctly copy strings between two objects.
37. Implement copy constructor to perform deep copy for a matrix.
38. Write a copy constructor to copy one object’s dynamically allocated array to another.

39. Demonstrate how copy constructor avoids double deletion errors.
40. Implement a copy constructor that allocates fresh memory for deep copy.

◆ E. Real-world & Mixed Problems (10 problems)

41. Create a `BankAccount` class with dynamically allocated account holder names.
42. Implement a `Student` class storing dynamic arrays of subject marks.
43. Write a `Library` class managing dynamically allocated list of books.
44. Design a `Database` class that allocates records dynamically using `new`.
45. Create a `Matrix` class that dynamically allocates a 2D array and overloads copy.
46. Demonstrate allocation and deletion of objects dynamically in inheritance.
47. Implement a simple dynamic array resizer (like `std::vector`).
48. Create a `DynamicString` class that performs deep copy in copy constructor.
49. Write a class that dynamically allocates memory for employee records and cleans up in destructor.
50. Demonstrate full memory lifecycle — allocation (constructor), copy (copy constructor), and deletion (destructor).

Would you like me to make a **C++ practice sheet (with input/output format)** for these 50 questions next?
(Useful if you want to code them one by one for OOP mastery.)

You said:

🔗 8. Operator Overloading Overloading unary and binary operators Overloading using member and friend functions Rules and limitations of operator overloading Overloading << and >> for I/O operations

ChatGPT said:

Here are **50 one-line C++ problem statements** based on
🔗 **Operator Overloading** — Unary, Binary, Member/Friend functions, Rules, and I/O operator (<<, >>) overloading:

□ A. Unary Operator Overloading (10 problems)

1. Overload the unary `-` operator to negate a complex number.
2. Overload the unary `++` operator to increment the value of an object's data member.
3. Overload the unary `--` operator to decrement a private variable of a class.
4. Overload the unary `!` operator to check if an object's value is zero.
5. Overload the unary `~` operator to invert all bits of an integer class.
6. Overload the unary `++` operator (prefix form) using a member function.
7. Overload the unary `++` operator (postfix form) using a member function.

8. Overload the unary `-` operator using a friend function.
9. Overload the unary `+` operator to return the absolute value of an object.
10. Overload the `--` operator for an object that represents a counter.

➕ B. Binary Operator Overloading (10 problems)

11. Overload the `+` operator to add two complex numbers.
12. Overload the `-` operator to subtract two vectors.
13. Overload the `*` operator to multiply two matrices.
14. Overload the `/` operator to divide two fraction objects.
15. Overload the `%` operator to find remainder between two integers using a class.
16. Overload the `==` operator to compare two strings for equality.
17. Overload the `!=` operator to compare two objects for inequality.
18. Overload the `>` and `<` operators for comparing object sizes.
19. Overload the `+` operator to concatenate two strings using a class.
20. Overload the `*` operator to perform scalar multiplication of a vector.

⚙ C. Member Function vs Friend Function Overloading (10 problems)

21. Overload the `+` operator using a **member function** for a `Distance` class.
22. Overload the `-` operator using a **member function** for a `Point` class.
23. Overload the `*` operator using a **friend function** for a `Matrix` class.
24. Overload the `+` operator using a **friend function** for a `Time` class.
25. Overload the `/` operator using a **friend function** for a `Complex` class.
26. Overload the `<` operator using a **member function** to compare objects.
27. Overload the `>` operator using a **friend function** for object comparison.
28. Overload the `==` operator using a **member function** to check equality.
29. Overload the `!=` operator using a **friend function** to check inequality.
30. Overload the `+` operator both as **member** and **friend**; compare both results.

⚠ D. Rules and Limitations of Operator Overloading (10 problems)

31. Try to overload the `.` operator and explain why it's not allowed.
32. Overload the `[]` operator to access array elements of a class.
33. Overload the `()` operator to call an object like a function.
34. Overload the `->` operator to access class members using an object pointer.
35. Overload the `=` operator to copy data between objects.
36. Overload the `+=` operator to add and assign in a single operation.
37. Demonstrate why operator overloading must involve at least one user-defined type.
38. Show that operator precedence and associativity can't be changed by overloading.
39. Demonstrate ambiguity when overloading operators with both friend and member functions.

40. Show an example where operator overloading leads to logical errors (bad design).

💬 E. Overloading << and >> for I/O Operations (10 problems)

41. Overload << and >> for a `Complex` class to input/output complex numbers.
42. Overload << and >> for a `Point` class to read and print coordinates.
43. Overload << and >> for a `Student` class to display name and marks.
44. Overload << and >> for a `Book` class to handle book details.
45. Overload << for a `Time` class to print time in `hh:mm:ss` format.
46. Overload >> to input details of an `Employee` class.
47. Overload both << and >> for a `Fraction` class to simplify input/output.
48. Overload << to print a vector object as `[x, y, z]`.
49. Overload >> to take input for a `Matrix` object from the user.
50. Overload << and >> together for a `Car` class to input and display car info.

Would you like me to make a **C++ problem sheet with input/output examples and class skeletons** for these 50 operator overloading problems next?
(It'll help you implement and test each problem easily.)

You said:

 9. Templates Function templates Class templates Template specialization Applications of templates (e.g., generic stack or list)

ChatGPT said:

Here are **50 one-line C++ problem statements** based on
 Templates — Function Templates, Class Templates, Specialization, and Generic Applications:

◆ A. Function Templates (15 problems)

1. Write a function template `swap()` to swap two variables of any type.
2. Create a function template `max()` to find the maximum of two values.
3. Implement a function template `min()` to find the minimum of three numbers.
4. Write a template function `sum()` to add two numbers of generic type.
5. Create a template function `multiply()` to multiply two numbers of any type.
6. Write a function template to print an array of any type.
7. Implement a function template `reverse()` to reverse an array of generic type.
8. Write a template function to find the average of elements in an array.
9. Implement a function template `compare()` to check equality of two values.
10. Write a function template to find the largest element in an array.
11. Create a template function `display()` that prints elements of a vector of any type.

12. Write a template function to find sum of elements in a vector.
13. Implement a function template `absolute()` to return absolute value of any numeric type.
14. Write a template function `countGreaterThan()` to count elements greater than a given value in an array.
15. Implement a function template `isSorted()` to check if an array of any type is sorted.

◆ B. Class Templates (15 problems)

16. Create a class template `Box` with generic length, width, and height.
17. Implement a class template `Pair` to store two values of any type.
18. Write a class template `Array` to handle generic array operations.
19. Create a class template `Stack` to push, pop, and display elements.
20. Implement a class template `Queue` for generic enqueue and dequeue.
21. Create a class template `Point` with x and y coordinates of any type.
22. Write a class template `Calculator` for addition and multiplication of generic types.
23. Implement a class template `Matrix` to store and display 2D arrays.
24. Create a class template `LinkedList` for generic insertion and deletion.
25. Implement a class template `Node` for storing data and pointer to next node.
26. Write a class template `PairSum` to store two values and calculate their sum.
27. Create a class template `StackArray` that uses a dynamic array for storage.
28. Implement a class template `CircularQueue` for generic types.
29. Write a class template `MinMax` to find minimum and maximum from an array.
30. Create a class template `Vector` with push_back, pop_back, and display functions.

◆ C. Template Specialization (10 problems)

31. Write a class template `Printer` for general type and specialize for `char*` strings.
32. Create a function template `compare()` and specialize it for `const char*`.
33. Implement a class template `Sum` for generic numeric types and specialize for `bool`.
34. Write a class template `Display` and specialize it for `string` type.
35. Create a class template `MaxValue` for numeric types and specialize for `char`.
36. Implement a function template `add()` and specialize for arrays of integers.
37. Write a class template `Calculator` and specialize for floating point numbers to use rounding.
38. Create a class template `Printer` and specialize for `bool` to print TRUE/FALSE.
39. Implement a template `Swap` and specialize for `string` arrays.
40. Write a class template `Compare` for numeric types and specialize for `string` type to compare lexicographically.

◆ D. Applications of Templates (Generic Data Structures / Real-world Use) (10 problems)

41. Implement a generic `Stack` class using templates.
42. Write a generic `Queue` class template and test enqueue/dequeue operations.
43. Create a template-based `LinkedList` to store elements of any type.
44. Implement a generic `Pair` class to hold coordinates and calculate distance.
45. Create a template class `ArrayOperations` to find sum, min, and max of elements.
46. Write a generic `Matrix` class for addition and multiplication of matrices.
47. Implement a template `SmartArray` with dynamic allocation and resizing.
48. Create a generic `PriorityQueue` template using arrays or vectors.
49. Write a template `GenericCalculator` for addition, subtraction, multiplication, and division.
50. Implement a generic `Graph` class template with adjacency list representation.

If you want, I can next create a **full C++ practice sheet for Templates** with **code skeletons, input/output examples, and hints** for all 50 problems — so you can directly practice coding them.

Do you want me to do that?

You said:

10. Exception Handling Introduction to exceptions try, catch, and throw keywords Multiple catch blocks Catching all exceptions Re-throwing exceptions Standard exception classes

ChatGPT said:

Here are **50 one-line C++ problem statements** based on
✹ **Exception Handling** — covering `try`, `catch`, `throw`, multiple catch blocks, re-throwing, and standard exceptions:

◆ A. Introduction to Exceptions (5 problems)

1. Write a program to divide two numbers and handle division by zero using exceptions.
2. Throw an exception if a negative number is entered for square root calculation.
3. Implement a program that throws an exception when a user enters invalid input for age.
4. Write a program that checks for array index out-of-bounds and throws an exception.
5. Throw a custom exception if a student's marks exceed 100.

◆ B. `try`, `catch`, and `throw` Keywords (10 problems)

6. Use `try`, `catch`, and `throw` to handle division by zero.
7. Write a program to throw an exception when a string is empty.
8. Implement a program to throw an exception if a character input is not a digit.

9. Write a program that throws an exception if user enters invalid menu choice.
10. Implement a function that throws exception when withdrawing more than account balance.
11. Throw an exception when a negative number is passed to a factorial function.
12. Write a program to throw an exception when a file fails to open.
13. Implement a program to throw an exception for invalid student roll number (negative value).
14. Throw an exception if a user enters a value outside a given range.
15. Write a program that throws an exception when dividing by zero inside a class method.

◆ C. Multiple Catch Blocks (10 problems)

16. Catch `int` and `float` exceptions separately in a division program.
17. Write a program to handle multiple exceptions: division by zero and negative square root.
18. Implement multiple catch blocks for `int` and `char` exceptions.
19. Catch different exceptions for file operations: file not found and permission denied.
20. Handle exceptions for both invalid array index and negative value input.
21. Write a program with multiple catch blocks to handle division by zero, overflow, and underflow.
22. Catch exceptions for invalid type conversion (e.g., string to int).
23. Write a program with multiple catches for negative numbers, zero, and numbers above 100.
24. Implement multiple catch blocks for arithmetic, logical, and runtime errors.
25. Handle exceptions for invalid input format and invalid menu selection using multiple catch blocks.

◆ D. Catching All Exceptions (5 problems)

26. Write a program using `catch(...)` to handle any exception.
27. Implement `catch(...)` after multiple specific catch blocks.
28. Demonstrate catching all exceptions in a program reading integers from the user.
29. Use `catch(...)` to handle unknown exceptions thrown inside a function.
30. Create a program with multiple exception types and catch-all handler.

◆ E. Re-throwing Exceptions (5 problems)

31. Write a program that catches an exception, prints a message, and re-throws it.
32. Implement re-throwing exception from a function to main.
33. Demonstrate catching a division by zero exception, re-throwing, and handling in main.
34. Write a program to catch invalid input in a function, re-throw, and catch in main.

35. Implement nested try-catch blocks with re-throwing exceptions.

◆ F. Standard Exception Classes (15 problems)

36. Handle `std::overflow_error` when an arithmetic operation exceeds limits.
37. Use `std::underflow_error` for numeric underflow scenario.
38. Throw `std::invalid_argument` when an invalid argument is passed.
39. Handle `std::out_of_range` for accessing invalid array index.
40. Use `std::length_error` for invalid string length operation.
41. Throw `std::domain_error` for square root of negative numbers.
42. Handle `std::range_error` for calculation exceeding a valid range.
43. Use `std::logic_error` for logical mistakes in program flow.
44. Demonstrate catching `std::runtime_error` in a file operation program.
45. Handle `std::bad_alloc` when dynamic memory allocation fails.
46. Throw `std::bad_cast` when type casting fails in dynamic_cast.
47. Use `std::bad_typeid` when typeid operator is used incorrectly.
48. Catch `std::ios_base::failure` during file I/O errors.
49. Demonstrate multiple standard exceptions in a single program (e.g., invalid_argument + out_of_range).
50. Implement a program using both user-defined and standard exceptions together.

If you want, I can create a **full C++ Exception Handling practice sheet** next — with **code skeletons, input/output examples, and hints** for all 50 problems.

Do you want me to do that?

You said:

💾 11. File Handling in C++ Streams: ifstream, ofstream, fstream Opening and closing files Reading and writing text/binary files File modes and flags Random access in files (seekg, seekp, tellg, tellp)

ChatGPT said:

Here are **50 one-line C++ problem statements** based on
💾 **File Handling in C++** — covering file streams, opening/closing, reading/writing, file modes, and random access:

◆ A. Basic File Operations (10 problems)

1. Write a program to create a text file `data.txt`.
2. Open a file in write mode and write a simple message.
3. Open a file in read mode and display its content.

4. Create a file using `fstream` and write multiple lines.
5. Open a file and append additional data to it.
6. Check if a file exists before opening it.
7. Close a file after writing and reopening in read mode.
8. Open a file in binary mode and write some integers.
9. Read a binary file and display the integers stored.
10. Demonstrate automatic closing of a file using RAII (`fstream` object goes out of scope).

◆ B. Reading and Writing Text Files (10 problems)

11. Write a program to read a file line by line using `getline()`.
12. Write a program to count the number of words in a text file.
13. Read a file character by character and display it.
14. Copy contents of one text file to another.
15. Write a program to convert all text in a file to uppercase while writing to another file.
16. Append user input lines to an existing file.
17. Count the number of lines in a text file.
18. Replace a word in a text file with another word.
19. Read a text file into a string vector and display it.
20. Write a program to merge contents of two text files into a new file.

◆ C. Reading and Writing Binary Files (10 problems)

21. Create a binary file to store integers using `ofstream`.
22. Read integers from a binary file using `ifstream`.
23. Store a structure `Student` in a binary file and read it back.
24. Write an array of floats to a binary file.
25. Read the array of floats back from the binary file.
26. Store multiple `Employee` objects in a binary file.
27. Read all `Employee` objects from the binary file.
28. Append a new record to an existing binary file.
29. Update a particular record in a binary file.
30. Delete a record from a binary file using temporary file approach.

◆ D. File Modes and Flags (10 problems)

31. Open a file in `ios::in` mode and read content.
32. Open a file in `ios::out` mode and write content.
33. Demonstrate opening a file in `ios::app` mode and adding data.
34. Use `ios::ate` to move to the end of file and write data.
35. Open a file in `ios::binary` mode and write numbers.
36. Combine `ios::in | ios::out` to read and write in the same file.

37. Use `ios::trunc` to clear the content of a file while opening.
38. Use `ios::nocreate` equivalent logic to open a file only if it exists.
39. Demonstrate `ios::ate` with `ios::in` to read from the end of file.
40. Show differences between `ios::app` and `ios::ate` while writing.

◆ E. Random Access in Files (seekg, seekp, tellg, tellp) (10 problems)

41. Use `tellg()` to find current read position in a text file.
42. Use `tellp()` to find current write position in a text file.
43. Use `seekg()` to move to the beginning of a file and read.
44. Use `seekp()` to move to a specific position in a file and write.
45. Read the 5th integer from a binary file using `seekg()`.
46. Write at the 3rd position of a binary file using `seekp()`.
47. Swap two records in a binary file using random access.
48. Append data at a specific position without overwriting using `seekp()`.
49. Read a specific line from a text file using random access with `seekg()`.
50. Demonstrate `seekg()` and `seekp()` to move forward and backward in a file.

If you want, I can next create a **full C++ File Handling practice sheet** with **input/output examples, hints, and code skeletons** for all 50 problems — so you can directly practice them.

Do you want me to do that?

You said:

 12. Advanced OOP Concepts Pointers to objects this pointer usage Object slicing Nested classes Composition and Aggregation Namespaces Inline functions Static vs Dynamic binding

ChatGPT said:

Here are **50 one-line C++ problem statements** based on
 Advanced OOP Concepts — covering pointers to objects, `this` pointer, object slicing, nested classes, composition/aggregation, namespaces, inline functions, and static vs dynamic binding:

◆ A. Pointers to Objects (8 problems)

1. Create a class `Student` and access its members using a pointer to object.
2. Write a program to call a member function using an object pointer.
3. Allocate a dynamic object using `new` and access its methods.
4. Create an array of object pointers and call member functions.

5. Pass an object pointer to a function and display its data.
6. Demonstrate accessing private members through a public method using object pointer.
7. Use `this` pointer with object pointer to return the calling object.
8. Create a program that deletes a dynamically allocated object using pointer.

◆ B. `this` Pointer Usage (6 problems)

9. Use `this` pointer to return current object from a member function.
10. Implement method chaining using `this` pointer.
11. Use `this` pointer to compare two objects of the same class.
12. Return object by reference using `this` pointer in a class method.
13. Use `this` pointer to distinguish between data members and function parameters.
14. Demonstrate `this` pointer in a class constructor.

◆ C. Object Slicing (4 problems)

15. Show object slicing when a derived class object is assigned to a base class object.
16. Write a program where slicing causes loss of derived class data.
17. Prevent object slicing using pointers or references.
18. Demonstrate slicing with multiple derived classes.

◆ D. Nested Classes (5 problems)

19. Create a class `Outer` with a nested class `Inner` and access `Inner` members.
20. Implement a nested class that modifies private members of the outer class.
21. Create a class `University` with nested class `Department` to store department info.
22. Demonstrate multiple levels of nested classes.
23. Use nested class to encapsulate helper functionality inside a main class.

◆ E. Composition and Aggregation (7 problems)

24. Implement composition: class `Car` contains `Engine` object.
25. Implement aggregation: class `Library` contains multiple `Book` pointers.
26. Show difference between composition and aggregation in a program.
27. Create a `School` class composed of `Principal` and `Teacher` objects.
28. Demonstrate dynamic aggregation using pointers.
29. Implement composition where destructor of main class deletes contained objects.
30. Create aggregation example with multiple `Student` objects in `Classroom`.

◆ F. Namespaces (5 problems)

31. Define two namespaces with functions having the same name and call them.
32. Use `using namespace` to avoid writing namespace repeatedly.
33. Nest one namespace inside another and call nested function.
34. Demonstrate aliasing a namespace using `namespace alias = original`.
35. Use namespaces to avoid name conflicts in a multi-class program.

◆ G. Inline Functions (5 problems)

36. Write an inline function to calculate the square of a number.
37. Use inline function inside a class to get/set a private member.
38. Demonstrate performance benefit of inline function for a small function.
39. Create an inline function to print array elements.
40. Implement inline function for a simple calculator operation (add, subtract).

◆ H. Static vs Dynamic Binding (10 problems)

41. Demonstrate static binding with a non-virtual function in base and derived class.
42. Show dynamic binding using a virtual function pointer to a derived object.
43. Create a base class pointer pointing to derived object and call virtual method.
44. Demonstrate late binding with multiple derived classes overriding the same virtual function.
45. Show static binding with overloaded functions.
46. Implement dynamic binding with pure virtual function (abstract class).
47. Show difference in output when virtual keyword is removed.
48. Demonstrate static binding inside constructors and destructors.
49. Use virtual function table concept to explain dynamic binding.
50. Implement a polymorphic collection of objects using base class pointer array to show dynamic binding.

I can next create a **complete Advanced OOP practice sheet** with **C++ code skeletons, input/output examples, and hints** for all 50 problems — so you can start coding them directly.

Do you want me to do that?

You said:

 13. Standard Template Library (STL) – Optional but Important Overview of STL Containers (vector, list, map, set, etc.) Iterators Algorithms (sort, find, etc.) Functors (Function objects)

ChatGPT said:

Here are **50 one-line C++ problem statements** based on
 Standard Template Library (STL) — covering containers, iterators, algorithms, and functors:

◆ A. STL Overview / Basics (5 problems)

1. Write a program to demonstrate the use of STL vector.
2. Create a list of integers and display them using an iterator.
3. Implement a map of student ID and name and display all entries.
4. Use set to store unique integers and print them in sorted order.
5. Demonstrate inserting elements into a vector, list, and set.

◆ B. Vectors (10 problems)

6. Create a vector of integers and find the sum of all elements.
7. Sort a vector of integers using `std::sort`.
8. Reverse a vector using `std::reverse`.
9. Find an element in a vector using `std::find`.
10. Insert an element at a specific position using `insert()`.
11. Delete an element at a specific position using `erase()`.
12. Resize a vector dynamically and display its elements.
13. Merge two vectors into a single vector.
14. Remove duplicates from a vector using `sort` and `unique`.
15. Count the frequency of a specific element in a vector.

◆ C. Lists (5 problems)

16. Create a list of integers and display them using an iterator.
17. Insert elements at the beginning and end of a list.
18. Delete an element from a list.
19. Sort a list using `list::sort()`.
20. Merge two lists into one sorted list.

◆ D. Maps (5 problems)

21. Create a map to store student marks and print all entries.
22. Update the value of a key in a map.
23. Delete a key-value pair from a map.
24. Find a specific key in a map and display its value.
25. Iterate over a map and display keys and values.

◆ E. Sets (5 problems)

26. Insert integers into a set and display them.
27. Delete a specific element from a set.
28. Check if an element exists in a set.
29. Use `set::lower_bound` and `set::upper_bound`.
30. Perform union and intersection of two sets using iterators.

◆ F. Iterators (5 problems)

31. Use iterator to traverse a vector and display elements.
32. Use reverse iterator to display a list in reverse order.
33. Iterate over a map and print all key-value pairs.
34. Use iterator to remove elements greater than a given value in a vector.
35. Use constant iterator to display elements of a vector without modifying them.

◆ G. Algorithms (10 problems)

36. Sort a vector of integers using `std::sort`.
37. Find the maximum and minimum element in a vector using `std::max_element` and `std::min_element`.
38. Count occurrences of an element using `std::count`.
39. Reverse a vector using `std::reverse`.
40. Find an element in a vector using `std::find`.
41. Use `std::accumulate` to find sum of elements in a vector.
42. Merge two sorted vectors using `std::merge`.
43. Remove duplicates from a vector using `std::unique`.
44. Use `std::binary_search` to check if an element exists in a sorted vector.
45. Sort a vector of pairs based on second element using custom comparator.

◆ H. Functors / Function Objects (5 problems)

46. Create a functor to square numbers and apply it to a vector using `std::for_each`.
47. Write a functor to compare two numbers and use it in `std::sort`.

48. Implement a functor to check if a number is even.
49. Use a functor to multiply each element of a vector by a constant.
50. Combine two functors: one for addition and one for multiplication on a vector.

If you want, I can next make a **full STL practice sheet** with **code skeletons, input/output examples, and hints** for all 50 problems — so you can start coding and practicing directly.

Do you want me to do that?