

Here are **30 problem statements** based on **C++ STL vector** (Dynamic Array). These range from beginner to advanced level and will help you master operations like insertion, deletion, sorting, searching, and STL algorithms with `vector`.

✓ Beginner (Basic Vector Operations)

1. Create a vector of integers and print all elements using a for loop.
 2. Read `n` integers from the user and store them in a vector.
 3. Insert an element at the beginning, middle, and end of a vector.
 4. Delete the first and last element from a vector.
 5. Access and print the first, last, and middle elements of a vector.
 6. Check if a vector is empty and print its size and capacity.
 7. Clear all elements of a vector and check if it's empty.
 8. Reverse a vector without using `reverse()` function.
 9. Copy elements of one vector to another using iterators.
 10. Sort a vector in ascending and descending order.
-

⚙ Intermediate (STL Functions and Algorithms)

11. Count the frequency of each element in a vector.
 12. Find the maximum and minimum element in a vector.
 13. Remove all duplicate elements from a vector.
 14. Search for an element using `find()` and return its index.
 15. Rotate the vector `k` times to the left/right.
 16. Merge two vectors and sort the result.
 17. Find the second largest and second smallest element.
 18. Remove all even numbers from a vector.
 19. Calculate the sum and average of all elements in a vector.
 20. Replace all negative numbers with 0 using `transform()`.
-

□ Advanced (STL + Problem Solving)

21. Find all pairs in a vector whose sum equals a given value.
22. Move all zeros to the end while maintaining the order of non-zero elements.
23. Find the longest increasing subsequence using a vector.
24. Implement binary search using `lower_bound` and `upper_bound`.
25. Partition the vector into even and odd numbers.
26. Remove all elements less than a given number `k`.
27. Check if a vector is palindrome or not.
28. Implement sliding window maximum for a given window size `k`.
29. Generate all permutations of a vector using `next_permutation`.
30. Count number of elements strictly greater than `x` using `upper_bound`.

Would you like the **C++ code** or **input/output examples** for any of these problems?

You said:

deque – Double-ended queue

ChatGPT said:

Here are **30 problem statements** based on **C++ STL deque** (Double-Ended Queue) to help you understand and practice its features from basic to advanced levels.

✓ Beginner (Basic deque Operations)

1. Create a deque of integers and print all elements using a for loop.
 2. Insert elements at both front and back of a deque.
 3. Delete elements from the front and back of a deque.
 4. Access the first and last element using `front()` and `back()`.
 5. Check if the deque is empty and print its size.
 6. Clear the deque and confirm it's empty.
 7. Initialize a deque with `n` identical elements.
 8. Traverse a deque using iterators.
 9. Insert an element at a specific position using `insert()`.
 10. Erase an element at a given position using `erase()`.
-

⚙ Intermediate (deque and STL Algorithms)

11. Reverse the deque using `reverse()` or manually.
 12. Sort a deque in ascending and descending order.
 13. Count occurrences of a given number in the deque.
 14. Find the maximum and minimum element in a deque.
 15. Remove all even numbers from the deque.
 16. Copy elements of one deque to another using iterators.
 17. Rotate the deque left or right by `k` positions.
 18. Merge two deques and sort the result.
 19. Replace negative numbers with 0 using `transform()`.
 20. Find if a given element exists in a deque using `find()`.
-

☐ Advanced (Applications of deque)

21. Implement a sliding window maximum using deque.
22. Check whether the sequence in deque is a palindrome.

23. Maintain the first non-repeating character in a stream using deque.
 24. Implement a queue using two deques.
 25. Given a deque, remove elements from either end to reach a target sum.
 26. Use deque to implement a monotonic queue (increasing/decreasing).
 27. Count number of subarrays with min and max within a given range using deque.
 28. Find the maximum sum of all subarrays of size k using sliding window.
 29. Use deque to simulate undo-redo operations.
 30. Solve the "First negative integer in every window of size k " using deque.
-

Would you like **C++ code**, **input/output examples**, or **detailed explanations** for any of these problems?

You said:

list – Doubly linked list

ChatGPT said:

Here are **30 problem statements** based on **C++ STL `list`** (which is a **doubly linked list**)—organized from beginner to advanced level to help you practice `list` effectively.

✔ **Beginner (Basic `list` Operations)**

1. Create a list of integers and print all elements using a loop.
 2. Insert an element at the front using `push_front()`.
 3. Insert an element at the back using `push_back()`.
 4. Delete the first element using `pop_front()`.
 5. Delete the last element using `pop_back()`.
 6. Access the front and back elements using `front()` and `back()`.
 7. Initialize a list with n identical elements and print it.
 8. Traverse a list using iterators.
 9. Clear the list and check if it is empty.
 10. Reverse the list using `reverse()` function.
-

⚙ **Intermediate (`list` with STL Features)**

11. Sort the list in ascending and descending order using `sort()`.
12. Merge two sorted lists using `merge()`.
13. Remove all occurrences of a given element using `remove()`.
14. Remove duplicate elements from a sorted list using `unique()`.
15. Insert an element at a specific position using `insert()` iterator.
16. Erase an element at a specific iterator position using `erase()`.
17. Find the maximum and minimum element in the list.

18. Count the number of times a given value appears in the list.
19. Replace all negative values with 0 using a loop.
20. Copy elements from one list to another using iterators.

❑ Advanced (Problem Solving Using `list`)

21. Reverse every k elements in a list.
22. Check if a list is a palindrome.
23. Find the middle element of the list.
24. Remove all elements less than a given number x .
25. Rotate the list k times to the left or right.
26. Remove all even-positioned elements.
27. Find the intersection of two sorted lists.
28. Partition the list into even and odd numbers.
29. Flatten a list of sorted lists into a single sorted list.
30. Implement LRU (Least Recently Used) Cache logic using `list + unordered_map`.

Would you like the **C++ code**, **input/output examples**, or **step-by-step explanations** for any of these problems?

You said:

forward_list – Singly linked list (C++11)

ChatGPT said:

Here are **30 problem statements** based on **C++ STL forward_list** (Singly Linked List introduced in C++11). These range from beginner to advanced and will help you understand how `forward_list` works compared to `list`.

✔ Beginner (Basic `forward_list` Operations)

1. Create a `forward_list` of integers and print all elements using a loop.
 2. Insert an element at the front using `push_front()`.
 3. Delete the front element using `pop_front()`.
 4. Traverse a `forward_list` using iterators.
 5. Initialize a `forward_list` with n identical elements and print it.
 6. Use `empty()` to check if a `forward_list` is empty.
 7. Use `front()` to access the first element.
 8. Clear a `forward_list` using `clear()`.
 9. Assign values from one `forward_list` to another.
 10. Reverse a `forward_list` using `reverse()`.
-

⚙ Intermediate (STL and `forward_list` Algorithms)

11. Sort the `forward_list` using `sort()`.
 12. Remove all occurrences of a given element using `remove()`.
 13. Remove elements that satisfy a condition using `remove_if()`.
 14. Use `unique()` to remove consecutive duplicates.
 15. Insert elements after a given position using `insert_after()`.
 16. Delete an element after a given iterator using `erase_after()`.
 17. Merge two sorted `forward_lists` using `merge()`.
 18. Count how many times a value appears using a loop.
 19. Replace all negative numbers with 0 using `for` loop and iterators.
 20. Use `before_begin()` and `begin()` to insert or erase at start.
-

□ Advanced (Problem Solving Using `forward_list`)

21. Find the middle element of a `forward_list`.
 22. Check if a `forward_list` is a palindrome.
 23. Remove all elements greater than a given number.
 24. Rotate the `forward_list` to the right by `k` places.
 25. Implement a queue using `forward_list`.
 26. Reverse every `k` nodes of the list.
 27. Detect if a loop exists in a `forward_list`. (*Requires custom node implementation*)
 28. Merge two `forward_lists` into a new sorted list without modifying originals.
 29. Partition list into even and odd values.
 30. Flatten a `forward_list` of lists (nested `forward_list`).
-

Would you like **C++ code**, **input/output examples**, or **detailed explanations** for any of these problem statements?

You said:

set – Unique elements in sorted order

ChatGPT said:

Here are **30 problem statements** based on the C++ STL `set` container (which stores **unique elements in sorted order**). These problems range from beginner to advanced to help you master the behavior and use of `set`.

✓ Beginner (Basic `set` Operations)

1. Create a set of integers and print all elements.
2. Insert multiple elements into a set and observe the sorted order.

3. Try inserting duplicate elements and check how `set` handles them.
 4. Check if an element exists in the set using `find()`.
 5. Erase a specific element from a set.
 6. Clear the set and check if it's empty.
 7. Use `size()` to print the number of elements in the set.
 8. Initialize a set with elements from an array.
 9. Use a loop to print all elements in a set using iterators.
 10. Find the minimum and maximum element in the set.
-

⚙ Intermediate (`set` with STL Algorithms and Applications)

11. Given a list of numbers, print only unique elements using a set.
 12. Find the union of two sets.
 13. Find the intersection of two sets.
 14. Find the difference between two sets.
 15. Count elements less than a given value using `lower_bound()`.
 16. Count elements greater than a given value using `upper_bound()`.
 17. Find the nearest lower and upper element to a given value.
 18. Erase a range of elements from the set.
 19. Create a set of characters from a string to remove duplicates.
 20. Insert custom user-defined types using a comparator.
-

□ Advanced (Problem Solving Using `set`)

21. Given n numbers, print the k -th smallest element using a set.
 22. Check if two arrays have at least one common element using sets.
 23. Find the length of the longest subarray with unique elements.
 24. Remove all even numbers from a set.
 25. Given a set, count how many elements fall within a given range $[L, R]$.
 26. Solve the "Two Sum" problem using a set for fast lookup.
 27. Maintain a dynamic leaderboard using set (score sorted).
 28. Given a stream of numbers, maintain the median using two sets.
 29. Check if a subarray with sum zero exists using set of prefixes.
 30. For each element in an array, find the next greater element on the right using a set.
-

Would you like **C++ code**, **examples**, or **step-by-step explanations** for any of these?

You said:

multiset – Duplicate elements in sorted order

ChatGPT said:

Here are **30 problem statements** based on the **C++ STL `multiset`** container** (which allows **duplicate elements** and keeps them in **sorted order**). These problems progress from beginner to advanced levels and are useful for mastering competitive programming or STL usage.

✓ Beginner (Basic `multiset` Operations)

1. Create a multiset of integers and print all elements.
 2. Insert elements (including duplicates) into a multiset and observe the order.
 3. Count the number of occurrences of a specific element using `count()`.
 4. Remove one occurrence of an element using `erase(iterator)`.
 5. Remove all occurrences of an element using `erase(value)`.
 6. Access the smallest and largest element using `begin()` and `rbegin()`.
 7. Traverse the multiset using iterators and print elements.
 8. Check if the multiset is empty and print its size.
 9. Initialize a multiset using elements from an array or vector.
 10. Clear all elements from the multiset.
-

⚙ Intermediate (STL Functions with `multiset`)

11. Use `equal_range()` to find the range of a particular value.
 12. Use `lower_bound()` and `upper_bound()` for searching ranges.
 13. Sort an array and count duplicates using `multiset`.
 14. Merge two multisets and print the result.
 15. Remove all elements within a given range `[L, R]`.
 16. Print frequency of each element using `multiset`.
 17. Copy a `multiset` into another and compare them.
 18. Create a descending multiset using a custom comparator.
 19. Find the number of unique elements using `set` from a `multiset`.
 20. Find the k-th smallest element from a multiset.
-

□ Advanced (Problem Solving Using `multiset`)

21. Maintain the top-k largest elements in a stream using a multiset.
22. Given an array, for each element find the number of smaller elements on the right.
23. Use `multiset` to simulate a multiset-based priority queue.
24. Given frequencies of characters, store them in a multiset and process top characters.
25. Solve the sliding window maximum/minimum problem using `multiset`.
26. Given a sequence of numbers, remove the median repeatedly using `multiset`.
27. Track the frequency of numbers while deleting and inserting elements.
28. Use `multiset` to simulate order-statistics (partial) by maintaining counts.
29. Print all elements of a multiset in reverse sorted order.
30. Implement a frequency multiset: insert/delete/show max frequency element.

Would you like **C++ code**, **input/output examples**, or **step-by-step explanations** for any specific problem?

You said:

map – Key-value pairs with unique keys (sorted)

ChatGPT said:

Here are **30 problem statements** based on **C++ STL map** – a container that stores **key-value pairs** with **unique keys in sorted order** (based on the key). These are arranged from basic to advanced to help you become proficient with `map`.

✓ Beginner (Basic `map` Operations)

1. Create a map of `int` to `string` and insert key-value pairs.
 2. Print all key-value pairs in a map using a loop.
 3. Access a value by key using `map[key]`.
 4. Check if a key exists in a map using `count()`.
 5. Use `find()` to search for a key and print the value if found.
 6. Erase a key from the map.
 7. Use `size()` and `empty()` functions on the map.
 8. Iterate through a map in sorted order of keys.
 9. Create a map using an initializer list.
 10. Clear all elements from the map.
-

⚙ Intermediate (Useful Applications of `map`)

11. Count the frequency of characters in a string.
 12. Count the frequency of words in a sentence.
 13. Store marks of students (name → marks) and print them sorted by name.
 14. Find the key with the maximum value in a map.
 15. Sort the elements of an array by frequency using a map.
 16. Use a `map<int, vector<int>>` to group elements by a property.
 17. Merge two maps and update values if keys overlap.
 18. Use `lower_bound()` and `upper_bound()` with a map.
 19. Replace values in a map based on a condition (e.g., increase all values > 50 by 10%).
 20. Store and retrieve configuration settings (key → setting value).
-

□ Advanced (Problem Solving Using `map`)

21. Given an array, find the first element that occurs only once using a map.
 22. Implement a leaderboard using `map<int, set<string>>` (score \rightarrow players).
 23. Find the number of subarrays with sum equal to `k` using prefix sum and a map.
 24. Group anagrams using `map<string, vector<string>>`.
 25. Implement LRU Cache using `map` and `list`.
 26. Check if two strings are isomorphic using two maps.
 27. Use a map to implement a basic DNS (domain name \rightarrow IP address).
 28. Given two arrays, find common elements and their frequencies using a map.
 29. Compress coordinates of an array using `map<int, int>`.
 30. Count the number of pairs with a given difference `k` using `map`.
-

Would you like **C++ code**, **examples**, or **explanations** for any of these problems?

You said:

`multimap` – Key-value pairs with duplicate keys (sorted)

ChatGPT said:

Here are **30 problem statements** based on **C++ STL `multimap`**, a container that stores **key-value pairs** where **duplicate keys are allowed**, and the keys are always **sorted**.

✓ Beginner (Basic `multimap` Operations)

1. Create a `multimap<int, string>` and insert key-value pairs (with some duplicate keys).
 2. Print all key-value pairs in a `multimap` using a loop.
 3. Use `insert()` to add multiple values for the same key.
 4. Check the number of entries for a key using `count()`.
 5. Erase all entries with a given key.
 6. Use `clear()` and check if the `multimap` is empty.
 7. Use `begin()`, `end()` to iterate through the `multimap`.
 8. Access all values for a given key using `equal_range()`.
 9. Initialize a `multimap` using a list of pairs.
 10. Find the first occurrence of a key using `find()`.
-

⚙ Intermediate (`multimap` Use Cases and STL Integration)

11. Group students by grade using `multimap<char, string>`.
12. Store file extensions and corresponding filenames (e.g., `.txt` \rightarrow `[file1, file2]`).
13. Sort dictionary entries where multiple definitions exist for a word.
14. Merge two `multimaps` with overlapping keys.
15. Count frequencies using `multimap<int, int>` and process duplicates.

16. Sort items by price, allowing multiple items to have the same price.
 17. Use `lower_bound()` and `upper_bound()` to search key ranges.
 18. Convert a `multimap` to a `map` by keeping only the latest inserted value.
 19. Remove only one occurrence of a key-value pair.
 20. Print all values of a key in descending order using a custom comparator.
-

❑ Advanced (Problem Solving Using `multimap`)

21. Design a system to track login timestamps for users using `multimap<string, int>`.
 22. Create an event scheduler where multiple events can occur at the same time.
 23. Track scores of players across different levels using `multimap<int, string>`.
 24. Group anagrams using `multimap<string, string>`.
 25. Store log messages sorted by timestamp allowing duplicate timestamps.
 26. Design a dictionary with multiple meanings per word (word \rightarrow meanings).
 27. Implement a shopping cart storing product IDs and multiple prices using `multimap`.
 28. Store and sort multiple comments posted at the same time.
 29. Manage student-subject enrollment using `multimap<string, string>`.
 30. Implement an auto-suggestion feature by prefix mapping using `multimap<string, string>`.
-

Would you like **C++ code, examples, or detailed explanations** for any of these?

You said:

`unordered_set` – Unique elements (hashed)

ChatGPT said:

Here are **30 problem statements** based on **C++ STL `unordered_set`** — a container that stores **unique elements in no particular order** using **hashing** for **average constant time** insertions, deletions, and lookups.

✔ Beginner (Basic `unordered_set` Operations)

1. Create an `unordered_set` of integers and print all elements.
2. Insert elements into an `unordered_set` and observe unordered output.
3. Try inserting duplicates and check the behavior.
4. Check if an element exists using `count()` or `find()`.
5. Erase an element from the set using `erase(value)`.
6. Use `size()`, `empty()` to get size and check if set is empty.
7. Clear the entire set using `clear()`.
8. Initialize an `unordered_set` from an array/vector.
9. Traverse all elements using a range-based for loop.
10. Insert a custom hashable type with user-defined hash and equality functions.

🌀 Intermediate (`unordered_set` + Logic Problems)

11. Count the number of unique elements in an array.
12. Find the intersection of two arrays using `unordered_set`.
13. Find the union of two arrays using `unordered_set`.
14. Determine if two arrays are disjoint using `unordered_set`.
15. Remove all duplicates from a given list of elements.
16. Check if a subarray with 0 sum exists using `unordered_set` of prefix sums.
17. Use `unordered_set` to detect duplicates in a stream of input.
18. Store all distinct characters from a string using `unordered_set`.
19. Find all repeating elements in an array using `unordered_set`.
20. Count how many elements are common between two `unordered_sets`.

📦 Advanced (Applications of `unordered_set`)

21. Find the length of the longest subarray with unique elements.
22. Use `unordered_set` to determine if a pair with given sum exists.
23. Use `unordered_set` in a sliding window to track unique characters.
24. Store all substrings of length k of a string using `unordered_set`.
25. Count number of subarrays with equal number of 0s and 1s using prefix sums.
26. Use `unordered_set` to check if a string is a pangram.
27. Use `unordered_set` to remove duplicates from a linked list.
28. Given an array of words, find all anagrams using sorted strings and `unordered_set`.
29. Detect cycle in an undirected graph using DFS and `unordered_set`.
30. Store and quickly lookup blacklisted IP addresses using `unordered_set<string>`.

Would you like **C++ code**, **input/output examples**, or **explanations** for any of the problems?

You said:

`unordered_multiset` – Duplicate elements (hashed)

ChatGPT said:

Here are **30 problem statements** based on C++ STL `unordered_multiset`, a container that allows **duplicate elements** and uses **hashing** to provide **average $O(1)$** time complexity for insertions, deletions, and lookups — **but does not maintain any order**.

✅ Beginner (Basic `unordered_multiset` Operations)

1. Create an `unordered_multiset<int>` and insert some values, including duplicates.
 2. Traverse the `unordered_multiset` and print all elements.
 3. Insert duplicate elements and count how many times an element appears using `count()`.
 4. Use `erase(value)` to remove **all** occurrences of a value.
 5. Use `erase(iterator)` to remove a **single occurrence** of a value.
 6. Check if a value exists using `find()`.
 7. Check the size of the container using `size()` and if it is empty using `empty()`.
 8. Clear all elements using `clear()`.
 9. Create an `unordered_multiset` from an array or vector.
 10. Use a loop to print all elements and note the lack of ordering.
-

⚙ Intermediate (`unordered_multiset` + STL Use Cases)

11. Count frequencies of elements in an array using `unordered_multiset`.
 12. Find the total number of duplicate entries in a container.
 13. Remove only one occurrence of a value from the multiset.
 14. Find the number of elements greater than a given value (requires manual counting).
 15. Merge two `unordered_multisets` and print the result.
 16. Initialize an `unordered_multiset` of strings and remove duplicates while preserving count.
 17. Remove all even numbers from an `unordered_multiset`.
 18. Replace all occurrences of a value with another (remove and re-insert).
 19. Sort elements of an `unordered_multiset` (requires copy to vector).
 20. Given two `unordered_multisets`, find their multiset intersection manually.
-

□ Advanced (Applications in Problem Solving)

21. Given an array, print each element the number of times it appears (frequency map using `unordered_multiset`).
 22. Use `unordered_multiset` to implement a bag of words (word counting in text).
 23. Use `unordered_multiset` to identify all duplicate elements in a list.
 24. Store hashed passwords with collision handling using `unordered_multiset`.
 25. Simulate a deck of cards where duplicates are allowed using `unordered_multiset<string>`.
 26. Group items by price and count how many of each using `unordered_multiset<int>`.
 27. Solve the Two Sum problem by counting occurrences using `unordered_multiset`.
 28. Track login attempts (timestamps) using `unordered_multiset<int>`.
 29. Implement a frequency-based search suggestion system using `unordered_multiset<string>`.
 30. Use `unordered_multiset` in competitive programming for sliding window frequency matching.
-

Would you like **C++ code**, **input/output examples**, or **concept explanations** for any of these problems?

You said:

`unordered_map` – Key-value pairs with unique keys (hashed)

ChatGPT said:

Here are **30 problem statements** based on **C++ STL `unordered_map`**, which stores **key-value pairs** with **unique keys** using **hashing** (unordered, fast average time complexity: $O(1)$ for insert, erase, find).

✔ Beginner (Basic `unordered_map` Operations)

1. Create an `unordered_map<int, string>` and insert key-value pairs.
 2. Access a value using a key (`umap[key]`).
 3. Insert elements using both `[]` and `insert()` method.
 4. Print all key-value pairs using a loop.
 5. Check if a key exists using `count()` or `find()`.
 6. Erase a key-value pair by key using `erase()`.
 7. Check the number of elements using `size()`.
 8. Clear the entire map using `clear()` and check if it's empty.
 9. Traverse the `unordered_map` using a range-based for loop.
 10. Create an `unordered_map` from a vector of pairs.
-

⚙ Intermediate (Use Cases and STL Logic)

11. Count the frequency of characters in a string using `unordered_map<char, int>`.
 12. Count word frequencies in a sentence using `unordered_map<string, int>`.
 13. Given an array, count occurrences of each number using `unordered_map<int, int>`.
 14. Find the most frequent element in an array.
 15. Build a phonebook using `unordered_map<string, string>` and perform lookups.
 16. Use `unordered_map<int, vector<string>>` to group items by ID.
 17. Reverse map: Create `unordered_map<string, int>` from a `map<int, string>`.
 18. Merge two `unordered_maps` (sum values if keys overlap).
 19. Store and access student marks using roll number as key.
 20. Store configuration settings using `unordered_map<string, string>`.
-

❑ Advanced (Problem Solving Using `unordered_map`)

21. Find the first non-repeating character in a string.
22. Implement LRU Cache using `unordered_map` and `list`.

23. Check if two strings are anagrams using `unordered_map<char, int>`.
 24. Given an array, find two elements that sum to a target (Two Sum problem).
 25. Find all subarrays with sum = 0 using prefix sums and `unordered_map`.
 26. Find the length of the longest subarray with equal number of 0s and 1s.
 27. Implement a basic DNS system: domain name → IP address.
 28. Group anagrams using `unordered_map<string, vector<string>>`.
 29. Use `unordered_map<string, unordered_set<string>>` to manage group memberships.
 30. Implement a frequency-based cache replacement policy using `unordered_map`.
-

Would you like **C++ code**, **input/output examples**, or **detailed explanations** for any of these problem statements?

You said:

`unordered_multimap` – Key-value pairs with duplicate keys (hashed)

ChatGPT said:

Here are **30 problem statements** based on **C++ STL `unordered_multimap`**, a container that stores **key-value pairs** with **duplicate keys allowed**, using **hashing** (for average **O(1)** access time). Unlike `map` and `multimap`, it **does not store data in any specific order**.

✓ Beginner (Basic `unordered_multimap` Operations)

1. Create an `unordered_multimap<int, string>` and insert key-value pairs with some duplicate keys.
 2. Print all key-value pairs using a range-based `for` loop.
 3. Insert elements using the `insert()` function (e.g. `insert({key, value})`).
 4. Count how many values are associated with a specific key using `count(key)`.
 5. Use `equal_range(key)` to retrieve all values associated with a given key.
 6. Erase all elements for a specific key using `erase(key)`.
 7. Check if the map is empty using `empty()` and get the size using `size()`.
 8. Initialize an `unordered_multimap` using an initializer list of pairs.
 9. Copy an `unordered_multimap` into another.
 10. Clear all elements in the container using `clear()`.
-

🔗 Intermediate (Practical Use Cases)

11. Store students grouped by their grades using `unordered_multimap<char, string>`.
12. Create a phonebook where multiple people can have the same phone number.
13. Store word translations (e.g., English → [Multiple meanings in Hindi]).

14. Track multiple timestamps for user activity using `unordered_multimap<string, int>`.
 15. Merge two `unordered_multimaps` and print all key-value pairs.
 16. Count all unique keys and values in an `unordered_multimap`.
 17. Create a reverse index: word \rightarrow all documents it appears in.
 18. Group strings by length using `unordered_multimap<int, string>`.
 19. Remove only one occurrence of a specific key-value pair.
 20. Check whether a certain (key, value) pair exists.
-

□ **Advanced (Problem Solving Using `unordered_multimap`)**

21. Implement a simple tagging system: tag \rightarrow list of posts.
 22. Given a list of cities and their temperatures, store and access multiple temps per city.
 23. Create a product database where multiple sellers can sell the same product.
 24. Create a logging system with severity \rightarrow multiple log messages.
 25. Given an array of numbers, store frequency in a way that allows repeated keys.
 26. Track student enrollments where students can be enrolled in multiple courses.
 27. Build an index: character \rightarrow positions where it appears in a string.
 28. Implement a calendar where multiple events can occur on the same day.
 29. Given a set of categories and items, store categories as keys and multiple items as values.
 30. Create a multimap of cities and restaurants, allowing multiple restaurants in the same city.
-

Would you like **C++ code, input/output examples, or detailed explanations** for any specific problems?

You said:

stack – LIFO structure (built on deque or vector)

ChatGPT said:

Here are **30 one-line problem statements** based on the **stack STL container (LIFO structure)** in C++:

□ **30 Stack STL Problem Statements (One Line)**

1. Push integers 1 to 5 onto a stack and print top.
2. Pop the top element of a stack and print remaining size.
3. Check if a stack is empty before popping.
4. Find the top element without popping it.
5. Reverse a string using stack.
6. Evaluate a postfix expression using stack.

7. Convert infix expression to postfix using stack.
8. Check for balanced parentheses using stack.
9. Push and pop characters in a stack and display LIFO order.
10. Sort elements in a stack using another stack.
11. Remove middle element of stack without extra space.
12. Clone a stack using another stack.
13. Implement a min-stack (track min at any time).
14. Remove all elements from stack one by one.
15. Check if stack of strings forms a palindrome.
16. Count elements in a stack without using size().
17. Insert element at bottom of stack.
18. Replace all elements in stack with their squares.
19. Reverse stack using recursion.
20. Check if a sequence of operations is valid stack sequence.
21. Push even numbers and pop until odd appears.
22. Print stack from bottom to top (without modifying).
23. Compare two stacks for equality.
24. Remove adjacent duplicates using stack.
25. Implement queue using two stacks.
26. Next Greater Element using stack.
27. Histogram Maximum Area using stack.
28. Decode a string with nested brackets using stack.
29. Push all vowels into stack from a string.
30. Use stack to simulate browser back button.

If you want **code** or **explanations** for any of these problems, feel free to ask!

You said:

queue – FIFO structur

ChatGPT said:

Here are **30 one-line problem statements** based on the **queue STL container (FIFO structure)** in C++:

□ **30 Queue STL Problem Statements (One Line)**

1. Enqueue integers 1 to 5 and print front element.
2. Dequeue an element and display updated queue.
3. Check if queue is empty before popping.
4. Count the number of elements in the queue.
5. Reverse the queue using a stack.
6. Print elements of a queue without modifying it.
7. Find the front and rear elements of the queue.
8. Implement a circular queue using STL queue.

9. Generate binary numbers from 1 to N using queue.
10. Interleave the first and second halves of queue.
11. Check if a queue of characters is a palindrome.
12. Sort a queue in increasing order.
13. Implement a stack using two queues.
14. Reverse the first K elements of the queue.
15. Sum all elements in a queue.
16. Duplicate each element in a queue.
17. Rotate the queue by K positions.
18. Check if all elements in queue are even.
19. Merge two queues into one.
20. Remove all odd numbers from a queue.
21. Print alternate elements of the queue.
22. Find max element in queue without removing elements.
23. Simulate ticket counter using queue.
24. Find frequency of each element in queue.
25. Implement priority queue using custom comparator.
26. Use queue to simulate call center request system.
27. Print elements in reverse using recursion.
28. Create queue from array elements.
29. Find second element from front.
30. Insert elements and print queue at each step.

Let me know if you'd like **C++ code** or **visual explanation** for any of these problems.

You said:

priority_queue – Max-heap by default

ChatGPT said:

Here are **30 one-line problem statements** based on **priority_queue STL** in C++ (which is a **max-heap by default**):

□ 30 Priority Queue (Max-Heap) STL Problem Statements

1. Insert integers into a max-heap and display the top element.
2. Remove the largest element from a max-heap.
3. Build a max-heap from an array of integers.
4. Display all elements of a priority queue (max-heap) without destroying it.
5. Implement a min-heap using `priority_queue`.
6. Find the K largest elements from a list using a max-heap.
7. Merge two max-heaps into a single heap.
8. Use max-heap to sort an array in descending order.
9. Check if a max-heap is empty before popping.
10. Count the number of elements in a max-heap.

11. Find the second largest element in a max-heap.
12. Use `priority_queue` to simulate a real-time task scheduler.
13. Insert and remove elements dynamically from a max-heap.
14. Maintain a running median using two heaps.
15. Implement a leaderboard using max-heap.
16. Replace the top element of a max-heap with a new value.
17. Store elements with frequency in max-heap and sort by frequency.
18. Print elements in descending order using a max-heap.
19. Track top K frequent elements using a max-heap.
20. Find the largest and second largest from a stream of inputs.
21. Maintain top 3 scores using max-heap.
22. Implement a hospital emergency queue using max-heap.
23. Print all elements greater than a given number in max-heap.
24. Push duplicate elements into a heap and print sorted order.
25. Use custom comparator to reverse max-heap into min-heap.
26. Insert `pair<int, string>` into a max-heap sorted by int.
27. Remove all elements one by one and show max-heap behavior.
28. Replace top element if new value is greater.
29. Track top K elements from a streaming array.
30. Count how many elements are greater than a threshold using a max-heap.

Would you like code examples for any of these problems using C++ `priority_queue`?

You said:

iterator – Basic iterator `const_iterator` – Read-only iterator `reverse_iterator` – Iterates in reverse `back_insert_iterator`, `front_insert_iterator`, `insert_iterator` – Special insert iterators

ChatGPT said:

Here are **30 problem statements** related to **C++ STL iterators**:

☐ **Basic iterator**

1. Traverse a `vector<int>` using a normal iterator and print elements.
2. Modify elements of a `list<int>` using iterator.
3. Print every second element of a `deque<string>` using an iterator.
4. Count how many even numbers are present in a `set<int>` using an iterator.
5. Implement a function that reverses a `vector<int>` using iterators.

const_iterator

6. Print a `vector<float>` using `const_iterator`.
7. Use `const_iterator` to find an element in a `set<int>` and print it.

8. Iterate through a `map<string, int>` using `const_iterator` and print keys only.
 9. Use `const_iterator` to calculate the sum of elements in a `list<int>`.
 10. Show the error that occurs if you try to modify elements using `const_iterator`.
-

reverse_iterator

11. Print a `vector<int>` in reverse using `reverse_iterator`.
 12. Create a palindrome check on a `deque<char>` using forward and reverse iterators.
 13. Traverse a `list<string>` in reverse using `rbegin()` and `rend()`.
 14. Reverse-print all values in a `map<int, string>` using reverse iterators.
 15. Create a function that uses reverse iterator to find last occurrence of an element.
-

Insert Iterators

16. Use `back_insert_iterator` to copy elements from `vector<int>` to another vector.
 17. Use `front_insert_iterator` to insert elements into a `list<int>` in reverse order.
 18. Use `insert_iterator` to insert elements into the middle of a `vector<int>`.
 19. Merge two vectors using `back_inserter`.
 20. Fill a `deque<string>` using `front_inserter` and show reverse behavior.
-

Advanced

21. Implement custom function `printIteratorRange(begin, end)` using templates.
 22. Use `std::find` with iterator on `vector<string>` to locate a value.
 23. Use `advance()` to skip first 3 elements and print remaining elements in `set<int>`.
 24. Use `next()` and `prev()` to access neighbors in `list<int>`.
 25. Use `distance()` to find the number of elements between two iterators.
-

☐ **Iterator Algorithms Integration**

26. Use `std::copy` with `back_inserter` to copy elements from list to vector.
27. Sort a `vector<int>` using iterators with `std::sort`.
28. Use iterator with `std::unique` to remove duplicates from `vector<int>`.
29. Count elements greater than a number using `count_if` with iterator.
30. Write a function that accepts any iterator type and prints values generically.