

Here are **20 one-line problem statements** based on **C++ STL Stack** 🖱

1. Reverse a string using a stack.
2. Check if parentheses in a string are balanced using a stack.
3. Evaluate a postfix expression using a stack.
4. Convert an infix expression to postfix using a stack.
5. Find the next greater element for each array element using a stack.
6. Implement a stack using two queues.
7. Implement a queue using two stacks.
8. Sort a stack using recursion.
9. Delete the middle element of a stack.
10. Check if a given string is a palindrome using a stack.
11. Find the minimum element in a stack in $O(1)$ time.
12. Reverse a stack using another stack.
13. Find the stock span for each day using a stack.
14. Remove adjacent duplicates from a string using a stack.
15. Check for redundant brackets in an expression using a stack.
16. Evaluate a prefix expression using a stack.
17. Find the largest rectangle area in a histogram using a stack.
18. Implement a special stack that supports `getMin()` in $O(1)$ time.
19. Find the celebrity in a party using a stack.
20. Design a stack that supports push, pop, top, and retrieving max element in $O(1)$ time.

Would you like me to add **constraints and sample input/output** for each (like Codeforces-style)?

You said:

queue

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL Queue** 🖱

1. Implement a queue using two stacks.
2. Implement a stack using two queues.
3. Reverse a queue using a stack.
4. Reverse the first **k** elements of a queue.
5. Generate binary numbers from 1 to **n** using a queue.
6. Find the first negative integer in every window of size **k** using a queue.
7. Implement a circular queue.
8. Implement a deque (double-ended queue) using STL.
9. Interleave the first half and second half of a queue.
10. Check whether all elements of one queue appear in the same order in another queue.
11. Find the maximum of all subarrays of size **k** using a deque.
12. Print the first non-repeating character in a stream using a queue.
13. Design a queue that supports `getMin()` in $O(1)$ time.
14. Check if a queue is a palindrome.
15. Count the number of elements in a queue without using `size()`.
16. Rotate a queue by **k** positions.

17. Implement a priority queue using STL and perform push/pop operations.
18. Find the last element in the queue after a series of enqueue and dequeue operations.
19. Simulate the ticket counter queue and find the time taken for the kth person.
20. Merge two sorted queues into one sorted queue.

Would you like me to include **constraints and test cases (like Codeforces style)** for these queue problems too?

You said:

map

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL Map** 

1. Count the frequency of each element in an array using a map.
2. Find the first non-repeating character in a string using a map.
3. Check if two strings are anagrams using a map.
4. Store and print student names with their marks using a map.
5. Find the most frequent element in an array using a map.
6. Print elements of a map in sorted order by key.
7. Count the number of distinct elements in an array using a map.
8. Given a string, print characters and their frequencies in lexicographical order.
9. Find the intersection of two arrays using a map.
10. Check if two arrays are equal (same elements with same frequency) using a map.
11. Store and retrieve employee details (ID → Name) using a map.
12. Find all pairs with a given sum in an array using a map.
13. Find the index of the first repeating element in an array using a map.
14. Group words by their frequency using a map.
15. Sort an array by frequency using a map.
16. Store and print country-capital pairs using a map.
17. Check if a subarray with sum zero exists using a map.
18. Given a string, find the character with the highest frequency using a map.
19. Merge two maps and print the combined result.
20. Count word frequencies in a sentence using a map.

Would you like me to add **constraints and sample input/output (Codeforces-style)** for these `map` problems next?

You said:

set

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL Set** 

1. Find all unique elements in an array using a set.
2. Count the number of distinct elements in an array using a set.
3. Check if all elements in an array are unique using a set.

4. Find the union of two arrays using a set.
5. Find the intersection of two arrays using a set.
6. Find elements that are present in one array but not in another using a set.
7. Remove duplicate characters from a string using a set.
8. Print all unique characters of a string in sorted order using a set.
9. Given two sets, find their symmetric difference.
10. Check whether two arrays have any common elements using a set.
11. Store and print unique words from a sentence using a set.
12. Find the smallest and largest elements in a set.
13. Implement a simple attendance system using a set of student IDs.
14. Count the number of unique pairs with a given sum using a set.
15. Check if a given element exists in a set.
16. Erase all even numbers from a set and print the remaining elements.
17. Find the missing number in a sequence using a set.
18. Print elements of a set in ascending and descending order.
19. Compare two sets and check if they are equal.
20. Find all subsets of a given set using STL containers.

Would you like me to add **constraints and sample input/output (Codeforces-style)** for these `set` problems too?

You said:

`<list>`

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL `<list>`** 

1. Insert elements at the front and back of a list and display them.
2. Delete a specific element from a list.
3. Reverse a list using STL functions.
4. Remove all duplicate elements from a list.
5. Merge two sorted lists into one sorted list.
6. Sort a list of integers in ascending order.
7. Sort a list of strings lexicographically.
8. Find the middle element of a list.
9. Remove all even numbers from a list.
10. Find the maximum and minimum elements in a list.
11. Count the occurrences of a given value in a list.
12. Check if a list is a palindrome.
13. Rotate a list by **k** positions.
14. Split a list into two halves.
15. Insert an element after every **k-th** position in a list.
16. Concatenate two lists using STL functions.
17. Remove consecutive duplicate elements from a list.
18. Replace all occurrences of a given value in a list with another value.
19. Implement a menu-driven program to perform list operations (push, pop, sort, reverse).
20. Find the sum and average of all elements in a list.

Would you like me to add **constraints and sample input/output (like Codeforces-style)** for these `<list>` problems too?

You said:

forward list

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL `<forward_list>`** 

1. Insert elements at the front of a forward list and display them.
2. Delete a specific element from a forward list.
3. Reverse a forward list using STL functions.
4. Merge two sorted forward lists into one sorted list.
5. Sort a forward list in ascending order.
6. Remove duplicate elements from a forward list.
7. Remove all even numbers from a forward list.
8. Find the maximum and minimum elements in a forward list.
9. Insert an element after a given position in a forward list.
10. Erase elements after a given position using `erase_after()`.
11. Count the number of elements in a forward list.
12. Check if a given value exists in a forward list.
13. Replace all occurrences of a given value with another value.
14. Split a forward list into two halves.
15. Concatenate two forward lists using `splice_after()`.
16. Remove elements greater than a given value.
17. Implement a menu-driven program to perform forward list operations (insert, delete, reverse, merge).
18. Find the sum and average of all elements in a forward list.
19. Create a forward list of strings and sort them alphabetically.
20. Remove consecutive duplicate elements using `unique()`.

Would you like me to add **constraints and sample input/output (Codeforces-style)** for these `<forward_list>` problems too?

You said:

priority quuq

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL `priority_queue`** 

1. Implement a max-heap using `priority_queue` and print elements in descending order.
2. Implement a min-heap using `priority_queue` and print elements in ascending order.
3. Find the k largest elements in an array using a priority queue.
4. Find the k smallest elements in an array using a priority queue.
5. Sort an array using a priority queue.

6. Merge k sorted arrays using a priority queue.
7. Implement a running median finder using a priority queue.
8. Find the top k frequent elements in an array using a priority queue.
9. Implement Dijkstra's shortest path algorithm using a priority queue.
10. Find the minimum cost to connect ropes (Huffman coding type) using a priority queue.
11. Simulate a task scheduler where tasks with higher priority run first.
12. Find the kth largest element in a stream of numbers using a priority queue.
13. Merge k sorted linked lists using a priority queue.
14. Sort a nearly sorted (k-sorted) array using a priority queue.
15. Process a series of events with priorities and print the order of execution.
16. Find the k closest points to the origin using a priority queue.
17. Implement a custom comparator for a priority queue of pairs.
18. Find the kth smallest/largest element in a matrix sorted row-wise and column-wise.
19. Implement a median priority queue supporting both insert and getMedian operations.
20. Simulate a patient queue in a hospital where sicker patients are treated first using a priority queue.

If you want, I can also make **Codeforces-style constraints with sample input/output** for these `priority_queue` problems. Do you want me to do that?

You said:

<unordered_map>

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL <unordered_map>** 📝

1. Count the frequency of each element in an array using `unordered_map`.
2. Find the first non-repeating element in an array using `unordered_map`.
3. Check if two strings are anagrams using `unordered_map`.
4. Store student names with their marks and retrieve them using `unordered_map`.
5. Find the most frequent element in an array using `unordered_map`.
6. Count the number of distinct elements in an array using `unordered_map`.
7. Find the intersection of two arrays using `unordered_map`.
8. Check if two arrays are equal (same elements with same frequency) using `unordered_map`.
9. Count the number of subarrays with sum zero using `unordered_map`.
10. Find all pairs in an array with a given sum using `unordered_map`.
11. Store and retrieve employee details (ID → Name) using `unordered_map`.
12. Count word frequency in a sentence using `unordered_map`.
13. Find the index of the first repeating element in an array using `unordered_map`.
14. Group strings by their frequency using `unordered_map`.
15. Check if a sequence contains duplicates using `unordered_map`.
16. Count elements appearing more than k times in an array using `unordered_map`.
17. Map characters to their ASCII values and print using `unordered_map`.
18. Merge two `unordered_maps` and print the combined result.
19. Find the longest subarray with all unique elements using `unordered_map`.
20. Implement a frequency-based sorting of elements using `unordered_map`.

I can also provide **Codeforces-style constraints with sample input/output** for these `<unordered_map>` problems if you want. Do you want me to do that?

You said:

`<unordered_set>`

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL `<unordered_set>`** 

1. Count the number of distinct elements in an array using `unordered_set`.
2. Check if an array contains duplicates using `unordered_set`.
3. Find the intersection of two arrays using `unordered_set`.
4. Find the union of two arrays using `unordered_set`.
5. Print all unique elements of an array using `unordered_set`.
6. Remove duplicate characters from a string using `unordered_set`.
7. Check if a string has all unique characters using `unordered_set`.
8. Count the number of unique words in a sentence using `unordered_set`.
9. Find elements present in one array but not in another using `unordered_set`.
10. Check if two arrays have any common elements using `unordered_set`.
11. Find the first repeating element in an array using `unordered_set`.
12. Remove all duplicates from an array in $O(n)$ time using `unordered_set`.
13. Store a set of unique IDs and check existence in $O(1)$ using `unordered_set`.
14. Find the length of the longest consecutive sequence in an array using `unordered_set`.
15. Check if a subset exists within an array using `unordered_set`.
16. Print all elements that appear only once in an array using `unordered_set`.
17. Implement a simple login system to check unique usernames using `unordered_set`.
18. Count the number of unique pairs with a given difference using `unordered_set`.
19. Store unique elements from a stream of numbers using `unordered_set`.
20. Merge two lists and keep only unique elements using `unordered_set`.

I can also create **Codeforces-style constraints with input/output examples** for these `<unordered_set>` problems if you want.

You said:

`<vector>`

ChatGPT said:

Here are **20 one-line problem statements** based on **C++ STL `<vector>`** 

1. Find the sum of all elements in a vector.
2. Reverse a vector using STL functions.
3. Find the maximum and minimum elements in a vector.
4. Sort a vector in ascending and descending order.
5. Remove all duplicate elements from a vector.
6. Rotate a vector by **k** positions.
7. Count the frequency of each element in a vector.

8. Find the k largest elements in a vector.
9. Find the k smallest elements in a vector.
10. Merge two sorted vectors into one sorted vector.
11. Find the subarray with the maximum sum in a vector.
12. Check if a vector is a palindrome.
13. Find the first repeating element in a vector.
14. Find the first non-repeating element in a vector.
15. Implement a dynamic array using `vector` and perform push/pop operations.
16. Find all pairs with a given sum in a vector.
17. Count the number of even and odd elements in a vector.
18. Print elements at even and odd positions in a vector.
19. Replace all occurrences of a given value in a vector with another value.
20. Split a vector into two equal halves.

I can also prepare **Codeforces-style constraints with input/output examples** for these `<vector>` problems if you want.

You said:

`<Algorithm>` 100 problem in methods all

ChatGPT said:

Here's a **comprehensive list of 100 one-line problem statements** using **C++ STL `<algorithm>` methods** (covering `sort`, `reverse`, `max_element`, `min_element`, `accumulate`, `count`, `find`, `binary_search`, `lower_bound`, `upper_bound`, `rotate`, `replace`, `unique`, `remove`, `partition`, `next_permutation`, `prev_permutation`, etc.):

Sorting & Searching (1–20)

1. Sort an array in ascending order using `sort()`.
2. Sort an array in descending order using `sort()` with a comparator.
3. Find the maximum element in an array using `max_element()`.
4. Find the minimum element in an array using `min_element()`.
5. Find the sum of elements in an array using `accumulate()`.
6. Count occurrences of a value in a vector using `count()`.
7. Check if a value exists in a vector using `find()`.
8. Check if a vector is sorted using `is_sorted()`.
9. Perform binary search for a value in a sorted array using `binary_search()`.
10. Find the lower bound of a value in a sorted array using `lower_bound()`.
11. Find the upper bound of a value in a sorted array using `upper_bound()`.
12. Find the distance between first and last occurrence of an element using `equal_range()`.
13. Reverse a vector using `reverse()`.
14. Rotate a vector by **k** positions using `rotate()`.
15. Generate all permutations of a vector using `next_permutation()`.
16. Generate the previous permutation of a vector using `prev_permutation()`.
17. Merge two sorted arrays using `merge()`.

18. Find intersection of two sorted arrays using `set_intersection()`.
 19. Find union of two sorted arrays using `set_union()`.
 20. Find the difference of two sorted arrays using `set_difference()`.
-

Modification & Transformation (21–40)

21. Replace all occurrences of a value in a vector using `replace()`.
 22. Replace all elements satisfying a condition using `replace_if()`.
 23. Remove all occurrences of a value using `remove()`.
 24. Remove consecutive duplicates using `unique()`.
 25. Transform elements using `transform()` (e.g., multiply by 2).
 26. Fill a vector with a specific value using `fill()`.
 27. Fill a part of a vector using `fill_n()`.
 28. Swap two elements using `swap()`.
 29. Swap two ranges of elements using `swap_ranges()`.
 30. Copy elements from one vector to another using `copy()`.
 31. Copy elements satisfying a condition using `copy_if()`.
 32. Move elements from one vector to another using `move()`.
 33. Generate a sequence of numbers using `iota()`.
 34. Shuffle a vector randomly using `random_shuffle()` (or `shuffle()`).
 35. Partition elements by a condition using `partition()`.
 36. Stable partition elements by a condition using `stable_partition()`.
 37. Count elements satisfying a condition using `count_if()`.
 38. Remove elements satisfying a condition using `remove_if()`.
 39. Check if all elements satisfy a condition using `all_of()`.
 40. Check if any element satisfies a condition using `any_of()`.
-

Set & Heap Operations (41–60)

41. Make a max-heap using `make_heap()`.
42. Make a min-heap using `make_heap()` with a comparator.
43. Push an element into a heap using `push_heap()`.
44. Pop the largest element from a heap using `pop_heap()`.
45. Sort a heap using `sort_heap()`.
46. Merge two sets using `set_union()`.
47. Find intersection of two sets using `set_intersection()`.
48. Find symmetric difference using `set_symmetric_difference()`.
49. Find difference of two sets using `set_difference()`.
50. Remove duplicates from a sorted vector using `unique()`.
51. Replace all negative numbers with zero using `replace_if()`.
52. Check if a vector contains a value using `find()`.
53. Count elements greater than a number using `count_if()`.
54. Sort in descending order using `greater<int>()`.
55. Find the first element greater than x using `upper_bound()`.
56. Find the first element not less than x using `lower_bound()`.

- 57. Find adjacent duplicates using `adjacent_find()`.
 - 58. Compare two ranges for equality using `equal()`.
 - 59. Swap two ranges using `swap_ranges()`.
 - 60. Rotate a vector to the left by `k` positions using `rotate()`.
-

Math & Numeric Operations (61–80)

- 61. Calculate sum using `accumulate()`.
 - 62. Calculate product of elements using `accumulate()`.
 - 63. Count odd numbers using `count_if()`.
 - 64. Count even numbers using `count_if()`.
 - 65. Check if all numbers are positive using `all_of()`.
 - 66. Check if any number is negative using `any_of()`.
 - 67. Check if no number is zero using `none_of()`.
 - 68. Generate factorials of numbers using `transform()`.
 - 69. Square all elements using `transform()`.
 - 70. Convert characters to uppercase using `transform()`.
 - 71. Convert characters to lowercase using `transform()`.
 - 72. Merge two sorted vectors using `merge()`.
 - 73. Find the sum of squares using `accumulate() + lambda`.
 - 74. Find maximum absolute value using `max_element()` with `lambda`.
 - 75. Find minimum absolute value using `min_element()` with `lambda`.
 - 76. Find the sum of even numbers using `accumulate() + lambda`.
 - 77. Find the sum of odd numbers using `accumulate() + lambda`.
 - 78. Compute prefix sum using `partial_sum()`.
 - 79. Compute inner product of two vectors using `inner_product()`.
 - 80. Count prime numbers using `count_if() + is_prime()`.
-

String & Character Operations (81–100)

- 81. Sort characters of a string using `sort()`.
- 82. Reverse a string using `reverse()`.
- 83. Replace all spaces in a string using `replace()`.
- 84. Remove consecutive duplicate characters using `unique()`.
- 85. Count vowels in a string using `count_if()`.
- 86. Convert string to uppercase using `transform()`.
- 87. Convert string to lowercase using `transform()`.
- 88. Check if a string is palindrome using `equal() + rbegin()`.
- 89. Find the first uppercase character using `find_if()`.
- 90. Find the last lowercase character using reverse iterators.
- 91. Replace all digits in a string using `replace_if()`.
- 92. Count all numeric characters using `count_if()`.
- 93. Remove all punctuation characters using `remove_if()`.
- 94. Sort words in a sentence alphabetically using `sort()`.
- 95. Merge two sorted lists of strings using `merge()`.

96. Find longest string in a vector using `max_element()`.
97. Find shortest string in a vector using `min_element()`.
98. Check if all strings start with a capital letter using `all_of()`.
99. Count strings of a specific length using `count_if()`.
100. Rotate words in a sentence using `rotate()`.