C++ Vector Functions

In C++, the vector header file provides various functions that can be used to perform different operations on a vector.

Function	Description
`size()`	returns the number of elements present in the vector
`clear()`	removes all the elements of the vector
`front()`	returns the first element of the vector
`back()`	returns the last element of the vector
`empty()`	returns **1** (true) if the vector is empty
`capacity()`	check the overall size of a vector

C++ Deque Methods

In C++, the `deque` class provides various methods to perform different operations on a deque.

Methods	Description
`push_back()`	inserts element at the back
`push_front()`	inserts element at the front
`pop_back()`	removes element from the back
`pop_front()`	removes element from the front
`front()`	returns the element at the front
`back()`	returns the element at the back
`at()`	sets/returns the element at specified index
`size()`	returns the number of elements
`empty()`	returns `true` if the deque is empty

List Functions

Method	Description
insert()	It inserts the new element before the position pointed by the iterator.
push_back()	It adds a new element at the end of the vector.
push_front()	It adds a new element to the front.
pop_back()	It deletes the last element.
pop_front()	It deletes the first element.
empty()	It checks whether the list is empty or not.
size()	It finds the number of elements present in the list.
max_size()	It finds the maximum size of the list.
front()	It returns the first element of the list.
back()	It returns the last element of the list.
swap()	It swaps two list when the type of both the list are same.
reverse()	It reverses the elements of the list.
sort()	It sorts the elements of the list in an increasing order.
merge()	It merges the two sorted list.
splice()	It inserts a new list into the invoking list.
unique()	It removes all the duplicate elements from the list.
resize()	It changes the size of the list container.
assign()	It assigns a new element to the list container.
emplace()	It inserts a new element at a specified position.
emplace_back()	It inserts a new element at the end of the vector.
emplace_front()	It inserts a new element at the beginning of the list.

Functions on forward_list

- `push_front(element)`: It adds a new element to the front of the list.
  Parameters: the element to be inserted
  Return value: void

- `pop_front()`: It deletes the first element of the list.
  Parameters: None
  Return value: void

- `insert_after(iterator,{values})`: It inserts the values after the given position.
  Parameters: iterator pointing to the position after which the value needs to be inserted, values that need to be inserted Return value: iterator, pointing to the last inserted element

- `erase_after(iterator)`: It deletes the values after the given position.
  Parameters: iterator, pointing to the position after which the value needs to be deleted
  Return value: void

- `remove(element)`: It removes all the occurrences of the given element.
  Parameters: element that needs to be deleted
  Return value: void

- `front()`: It returns the first element of the list.
  Parameters: None
  Return value: void

- `begin()`: It returns an iterator pointing to the first element of the list.

Parameters: None
Return value: iterator, pointing to the first element

• end(): It returns an iterator pointing to the last element of the list.
Parameters: None
Return value: iterator, pointing to the last element

• max_size(): It returns the maximum number of elements that can be stored by the forward_list.
Parameters: None
Return value: integer, the maximum size

• empty(): It tells us whether the list is empty or not
Parameters: None
Return value: boolean, true if the list is empty otherwise false.

• reverse(): It reverses the list.
Parameters: None
Return value: void

set:

•A Set stores the elements in sorted order.
•Set stores unique elements.
•Elements can only be inserted and deleted but cannot be modified within the set.
•Sets are implemented using a binary search tree.
•Sets are traversed using iterators.

Methods on set

There is a wide range of operations that can be performed on sets in C++. Let us look at some of the vital methods of sets.

1. **begin()**: This method returns an iterator that points to the first element in the set.
2. **end()**: This function returns an iterator that points to the theoretical position next to the last element of the set.
3. **empty()**: This set method is used for checking whether the set is empty or not.

4.**size()**: This function gives the number of elements present in a set .
5.**max_size()**: This method returns upper bound of elements in a set, i.e. the maximum number that a set can hold.
6.**rbegin()**: In contrary to the begin() method, this method returns a reverse iterator pointing to the last element of a set.
7.**rend()**: In contrary to the begin() method, this method returns a reverse iterator pointing to the logical position before the last element of a set.
8.**erase (iterator_position)**: This method when applied on a set, removes the element at the position pointed by the pointer given in the parameter.
9.**erase (const_n)**: This function directly deletes the value 'n' passed in the parameter from a set .
10.**insert (const_n)**: This function inserts a new element 'n' into the set .
11.**find(n)**: This method searches for the element 'n' in the set and returns an iterator pointing to the position of the found element. If the element is not found, it returns an iterator pointing at the end.
12.**count(const_n)**: This set method checks for the occurrence of the passed value 'n' and returns 0 or 1 if the element is found or not found respectively.
13.**clear()**: It removes all the elements present in a set.

Difference between set, multiset, and unordered_set

Parameters	**set**	**multiset**	**unordered_set**
Storage Order	Stores elements in a sorted manner	Stores elements in a sorted manner – in increasing or decreasing order	Stores elements in an unsorted order
Type of elements	Holds unique elements only	It can hold duplicate values	Only unique values are allowed
Modification	Insertion and deletion of elements are possible, but modification of elements is not allowed	Deletion of elements can be done with the help of iterators	Insertion and deletion of elements are possible, but modification of elements is not allowed
Implementation	Implemented internally via Binary Search Trees	Internal implementation through Binary Search Trees	Implemented internally using Hash Tables
Erasing element	More than one elements can be erased by giving the starting and ending iterator	We can erase that element for which the iterator position is given	We can erase that element for which the iterator position is given
Syntax	*set<datatype> setName*	*multiset<datatype> multiSetName;*	*unordered_set<datatype> UnorderedSetName;*

Multiset :

•It also stores the elements in a sorted manner, either ascending or decreasing.
•Multiset allows storage of the same values many times; or, in other words, duplicate values can be stored in the multiset.
•We can delete the elements using iterators.

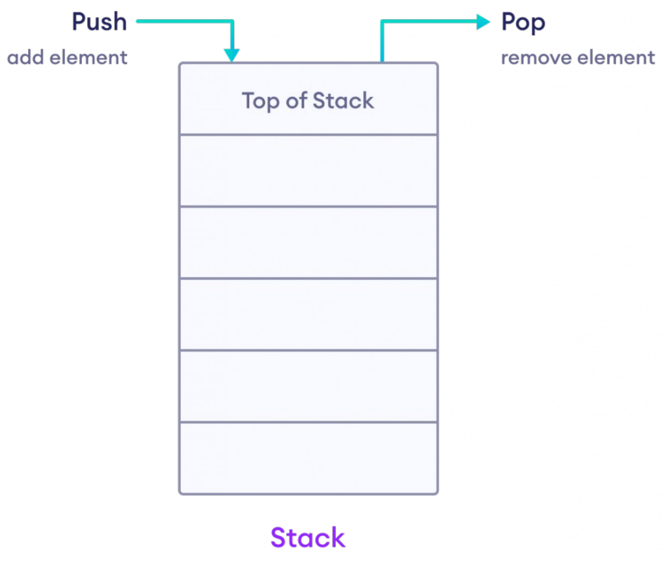
Unordered_Set :

•Unordered_set can stores elements in any order, with no specified order for storing elements. It generally depends upon the machine that you are using.
•Stores unique elements, no duplicates allowed.
•unordered_set uses the hash table for storing the element.

Stack

The STL `stack` provides the functionality of a stack data structure in C++.

The stack data structure follows the **LIFO (Last In First Out)** principle. That is, the element added last will be removed first.



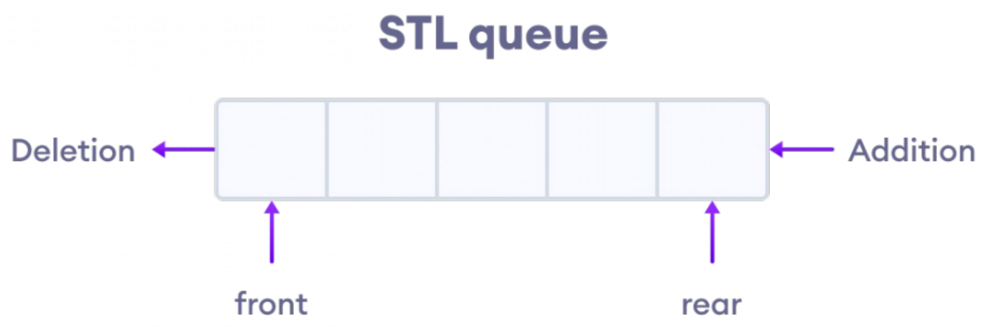
Stack Methods

Operation	Description
`push()`	adds an element into the stack
`pop()`	removes an element from the stack
`top()`	returns the element at the top of the stack
`size()`	returns the number of elements in the stack
`empty()`	returns `true` if the stack is empty

Queue

The STL `queue` provides the functionality of a queue data structure.

The queue data structure follows the **FIFO (First In First Out)** principle where elements that are added first will be removed first.



Queue Methods

Methods	Description

`push()`	inserts an element at the back of the queue
`pop()`	removes an element from the front of the queue
`front()`	returns the first element of the queue
`back()`	returns the last element of the queue
`size()`	returns the number of elements in the queue
`empty()`	returns `true` if the queue is empty

Unordered Map

The STL `unordered_map` is an unordered associative container that provides the functionality of an unordered map or dictionary data structure.

In contrast to a regular map, the order of keys in an unordered map is undefined.

Also, the unordered map is implemented as a hash table data structure whereas the regular map is a binary search tree data structure.

Unordered Map Methods

Methods	Description
`insert()`	insert one or more key-value pairs
`count()`	returns **1** if key exists and **0** if not
`find()`	returns the iterator to the element with the specified key
`at()`	returns the element at the specified key
`size()`	returns the number of elements
`empty()`	returns true if the unordered map is empty
`erase()`	removes elements with specified key
`clear()`	removes all elements