The Standard Template Library (STL) in C++ provides a rich set of template classes and functions to handle data structures and algorithms. STL is divided into four major components:

1. **Containers**: Classes that store data.
2. **Algorithms**: Functions for processing sequences of elements.
3. **Iterators**: Objects that allow traversal of elements in a container.
4. **Functions**: Objects that act as function wrappers.

Below is an overview of the most commonly used STL containers and their important functions:

**1. Sequence Containers**

**a. vector**

A dynamic array that can resize itself automatically.

* **Important Functions:**
  + push\_back(element): Adds an element to the end.
  + pop\_back(): Removes the last element.
  + insert(position, element): Inserts element at a specific position.
  + erase(position): Removes element at a specific position.
  + resize(new\_size): Resizes the container to contain new\_size elements.
  + clear(): Removes all elements.
  + size(): Returns the number of elements.
  + capacity(): Returns the size of the allocated storage capacity.
  + empty(): Checks if the container is empty.
  + front(), back(): Access the first and last elements.
  + data(): Returns a pointer to the underlying array.

**b. deque**

A double-ended queue that allows fast insertion and deletion at both ends.

* **Important Functions:**
  + push\_front(element), push\_back(element): Adds elements to the front or back.
  + pop\_front(), pop\_back(): Removes elements from the front or back.
  + insert(position, element): Inserts element at a specific position.
  + erase(position): Removes element at a specific position.
  + size(), clear(), empty(): Similar to vector.
  + front(), back(), at(position): Access elements.

**c. list**

A doubly-linked list allowing fast insertions and deletions.

* **Important Functions:**
  + push\_front(element), push\_back(element): Adds elements to the front or back.
  + pop\_front(), pop\_back(): Removes elements from the front or back.
  + insert(position, element): Inserts element at a specific position.
  + erase(position): Removes element at a specific position.
  + remove(value): Removes all elements with the specified value.
  + sort(): Sorts the list.
  + reverse(): Reverses the order of elements.
  + merge(other\_list): Merges two sorted lists.
  + size(), clear(), empty(): Similar to vector.

**d. forward\_list**

A singly-linked list with fast insertion and deletion at the front.

* **Important Functions:**
  + push\_front(element): Adds an element to the front.
  + pop\_front(): Removes the front element.
  + insert\_after(position, element): Inserts element after a specific position.
  + erase\_after(position): Removes the element after a specific position.
  + remove(value): Removes all elements with the specified value.
  + sort(), reverse(), merge(): Similar to list.
  + size(), clear(), empty(): Similar to vector (but size() is not provided in forward\_list).

**e. array**

A fixed-size array with fast access to elements.

* **Important Functions:**
  + at(position), []: Access elements.
  + front(), back(): Access the first and last elements.
  + data(): Returns a pointer to the underlying array.
  + size(), empty(): Similar to vector.
  + fill(value): Fills the array with the specified value.
  + swap(other\_array): Swaps contents with another array.

**2. Associative Containers**

**a. set**

A collection of unique elements in sorted order.

* **Important Functions:**
  + insert(element): Adds an element.
  + erase(element): Removes an element.
  + find(value): Finds an element.
  + count(value): Returns 1 if the element is present, 0 otherwise.
  + size(), clear(), empty(): Similar to vector.
  + lower\_bound(value), upper\_bound(value): Return iterators for element ranges.

**b. multiset**

Similar to set, but allows duplicate elements.

* **Important Functions:**
  + insert(element): Adds an element.
  + erase(element): Removes an element.
  + find(value): Finds an element.
  + count(value): Returns the count of the element.
  + size(), clear(), empty(): Similar to vector.

**c. map**

A collection of key-value pairs, sorted by keys.

* **Important Functions:**
  + insert(pair): Adds a key-value pair.
  + erase(key): Removes a key-value pair.
  + find(key): Finds an element by key.
  + operator[]: Accesses element by key (creates if not present).
  + count(key): Returns 1 if the key is present, 0 otherwise.
  + size(), clear(), empty(): Similar to vector.

**d. multimap**

Similar to map, but allows multiple keys.

* **Important Functions:**
  + insert(pair): Adds a key-value pair.
  + erase(key): Removes key-value pairs with a specific key.
  + find(key): Finds the first element with the key.
  + count(key): Returns the count of elements with a specific key.
  + size(), clear(), empty(): Similar to vector.

**3. Unordered Containers**

**a. unordered\_set**

A collection of unique elements with fast access (hash table).

* **Important Functions:**
  + insert(element), erase(element), find(element), count(element): Similar to set.
  + size(), clear(), empty(): Similar to vector.

**b. unordered\_multiset**

Similar to unordered\_set but allows duplicate elements.

* **Important Functions:**
  + insert(element), erase(element), find(element), count(element): Similar to multiset.
  + size(), clear(), empty(): Similar to vector.

**c. unordered\_map**

A collection of key-value pairs with fast access (hash table).

* **Important Functions:**
  + insert(pair), erase(key), find(key), operator[], count(key): Similar to map.
  + size(), clear(), empty(): Similar to vector.

**d. unordered\_multimap**

Similar to unordered\_map but allows multiple keys.

* **Important Functions:**
  + insert(pair), erase(key), find(key), count(key): Similar to multimap.
  + size(), clear(), empty(): Similar to vector.

**4. Container Adapters**

**a. stack**

LIFO (Last In, First Out) data structure.

* **Important Functions:**
  + push(element): Adds an element to the top.
  + pop(): Removes the top element.
  + top(): Accesses the top element.
  + size(), empty(): Similar to vector.

**b. queue**

FIFO (First In, First Out) data structure.

* **Important Functions:**
  + push(element): Adds an element to the back.
  + pop(): Removes the front element.
  + front(), back(): Accesses the first and last elements.
  + size(), empty(): Similar to vector.

**c. priority\_queue**

A max-heap by default, providing constant time lookup of the largest element.

* **Important Functions:**
  + push(element): Adds an element.
  + pop(): Removes the largest element.
  + top(): Accesses the largest element.
  + size(), empty(): Similar to vector.

**Summary**

These are the essential STL data structures and their important functions in C++. Each container has unique characteristics suitable for different scenarios, and knowing them helps in writing efficient C++ code.