1. Vector

- **Description**: A dynamic array that can resize itself automatically when elements are added or removed.
- **Key Operations**: `push_back`, `pop_back`, `size`, `at`, `begin`, `end`
- **Example**:

```cpp
#include <vector>
std::vector<int> v;
v.push_back(10); // Add an element
v.push_back(20);
std::cout << v[0]; // Access first element
```

2. List

- **Description**: A doubly linked list that allows fast insertions and deletions at both ends.
- **Key Operations**: `push_front`, `push_back`, `pop_front`, `pop_back`, `size`, `begin`, `end`
- **Example**:

```cpp
#include <list>
std::list<int> l;
l.push_back(10);  // Add element at the end
l.push_front(5);  // Add element at the front
```

3. Deque

- **Description**: A double-ended queue that allows insertion and deletion of elements from both ends efficiently.
- **Key Operations**: `push_front`, `push_back`, `pop_front`, `pop_back`, `at`, `begin`, `end`
- **Example**:

```cpp
#include <deque>
std::deque<int> dq;
dq.push_back(10);
dq.push_front(5);  // Insert at the front
```

4. Set

- **Description**: A collection of unique elements, usually implemented as a balanced tree (ordered).
- **Key Operations**: `insert`, `erase`, `find`, `count`, `size`, `begin`, `end`
- **Example**:

```
#include <set>
std::set<int> s;
s.insert(10);
s.insert(20);
```

5. Map

- **Description**: A collection of key-value pairs, where keys are unique, and elements are ordered by the key.
- **Key Operations**: `insert`, `erase`, `find`, `at`, `begin`, `end`
- **Example**:

```
#include <map>
std::map<int, std::string> m;
m[1] = "one";  // Insert key-value pair
m[2] = "two";
```

6. Multiset

- **Description**: A collection of elements that can have duplicate values, ordered by value.
- **Key Operations**: `insert`, `erase`, `count`, `find`, `begin`, `end`
- **Example**:

```
#include <set>
std::multiset<int> ms;
ms.insert(10);
ms.insert(10);  // Duplicates allowed
```

7. Multimap

- **Description**: A collection of key-value pairs, where keys can be duplicated.
- **Key Operations**: `insert`, `erase`, `find`, `count`, `begin`, `end`
- **Example**:

```cpp
#include <map>
std::multimap<int, std::string> mm;
mm.insert({1, "one"});
mm.insert({1, "uno"});  // Duplicate keys allowed
```

8. Stack

- **Description**: A container adapter that gives the behavior of a stack (LIFO: Last In First Out).
- **Key Operations**: `push`, `pop`, `top`, `empty`, `size`
- **Example**:

```cpp
#include <stack>
std::stack<int> stk;
stk.push(10);
stk.push(20);
int top_element = stk.top();  // 20
stk.pop();  // Removes 20
```

9. Queue

- **Description**: A container adapter that gives the behavior of a queue (FIFO: First In First Out).
- **Key Operations**: `push`, `pop`, `front`, `back`, `empty`, `size`
- **Example**:

```cpp
#include <queue>
std::queue<int> q;
q.push(10);
q.push(20);
int front_element = q.front();  // 10
q.pop();  // Removes 10
```

10. Priority Queue

- **Description**: A container adapter that provides the behavior of a priority queue, where the highest (or lowest) element is always accessible.
- **Key Operations**: `push`, `pop`, `top`, `empty`, `size`
- **Example**:

```
#include <queue>
std::priority_queue<int> pq;
pq.push(10);
pq.push(20);
int top_element = pq.top();  // 20 (default: max-heap)
pq.pop();  // Removes 20
```

11. Array

- **Description**: A fixed-size container that provides direct access to its elements by index.
- **Key Operations**: `at`, `size`, `begin`, `end`
- **Example**:

```
#include <array>
std::array<int, 3> arr = {1, 2, 3};
int first_element = arr[0];  // 1
```

12. Unordered Set

- **Description**: A collection of unique elements, but without any specific order, implemented as a hash table.
- **Key Operations**: `insert`, `erase`, `find`, `count`, `size`, `begin`, `end`
- **Example**:

```
#include <unordered_set>
std::unordered_set<int> us;
us.insert(10);
us.insert(20);
```

13. Unordered Map

- **Description**: A collection of key-value pairs, where keys are unique, but there is no specific order. It uses hashing for fast lookup.
- **Key Operations**: `insert`, `erase`, `find`, `at`, `begin`, `end`
- **Example**:

```
#include <unordered_map>
std::unordered_map<int, std::string> um;
um[1] = "one";  // Insert key-value pair
um[2] = "two";
```

14. Unordered Multiset

- **Description**: A collection of elements that can have duplicates, but no specific order, implemented as a hash table.
- **Key Operations**: `insert`, `erase`, `find`, `count`, `begin`, `end`
- **Example**:

```
#include <unordered_set>
std::unordered_multiset<int> ums;
ums.insert(10);
ums.insert(10);  // Duplicates allowed
```

15. Unordered Multimap

- **Description**: A collection of key-value pairs, where keys can be duplicated, and no specific order is enforced. It uses hashing.
- **Key Operations**: `insert`, `erase`, `find`, `count`, `begin`, `end`
- **Example**:

```
#include <unordered_map>
std::unordered_multimap<int, std::string> umm;
umm.insert({1, "one"});
umm.insert({1, "uno"});  // Duplicate keys allowed
```

Summary Table of Containers:

Container	Description	Key Operations
Vector	Dynamic array (resizable)	`push_back`, `pop_back`, `size`, `begin`, `end`
List	Doubly linked list	`push_front`, `push_back`, `pop_front`, `pop_back`, `size`
Deque	Double-ended queue	`push_front`, `push_back`, `pop_front`, `pop_back`, `at`, `size`
Set	Collection of unique elements (ordered)	`insert`, `erase`, `find`, `count`, `size`, `begin`, `end`
Map	Collection of key-value pairs (unique keys, ordered)	`insert`, `erase`, `find`, `at`, `size`, `begin`, `end`
Multiset	Collection of elements (duplicates allowed, ordered)	`insert`, `erase`, `count`, `find`, `size`, `begin`, `end`
Multimap	Collection of key-value pairs (duplicate keys allowed, ordered)	`insert`, `erase`, `find`, `at`, `size`, `begin`, `end`
Stack	Container adapter (LIFO)	`push`, `pop`, `top`, `empty`, `size`
Queue	Container adapter (FIFO)	`push`, `pop`, `front`, `back`, `empty`, `size`
Priority Queue	Container adapter (ordered, based on priority)	`push`, `pop`, `top`, `empty`, `size`
Array	Fixed-size array	`at`, `size`, `begin`, `end`
Unordered Set	Hash table (unique elements, unordered)	`insert`, `erase`, `find`, `count`, `size`, `begin`, `end`
Unordered Map	Hash table (unique keys, unordered)	`insert`, `erase`, `find`, `at`, `size`, `begin`, `end`
Unordered Multiset	Hash table (duplicate elements, unordered)	`insert`, `erase`, `find`, `count`, `size`, `begin`, `end`
Unordered Multimap	Hash table (duplicate keys allowed, unordered)	`insert`, `erase`, `find`, `count`, `size`, `begin`, `end`