✅ ✅ C++ STL Detailed Cheatsheet ✅ ✅

📦 I. Container Classes (Store elements directly)

These are core data structures that manage collections of objects.

🔹 1. vector – Dynamic Array

- **Header**: #include <vector>
- **Syntax**: vector<int> v;
- **Key Features**: Fast random access, dynamic resizing

Operations:

- v.push_back(x) – Add element to end
- v.pop_back() – Remove last element
- v[i], v.at(i) – Access by index
- v.insert(pos, val) – Insert at position
- v.erase(pos) – Erase at position
- v.clear() – Remove all elements
- v.size(), v.empty() – Size & emptiness check
- v.front(), v.back() – First and last element
- sort(v.begin(), v.end()) – Sort the vector

🔹 2. list – Doubly Linked List

- **Header**: #include <list>
- **Syntax**: list<int> l;
- **Key Features**: Fast insertion/deletion from both ends

Operations:

- l.push_back(x), l.push_front(x) – Add at back/front
- l.pop_back(), l.pop_front() – Remove from back/front
- l.insert(it, val) – Insert before iterator
- l.erase(it) – Erase element
- l.sort() – Sort the list
- l.reverse() – Reverse the list
- l.size(), l.empty() – Size and emptiness check

🔹 **3. deque – Double-Ended Queue**

- **Header**: #include <deque>
- **Syntax**: deque<int> d;
- **Key Features**: Fast insertion/deletion at both ends

Operations:

- d.push_back(x), d.push_front(x) – Add at ends
- d.pop_back(), d.pop_front() – Remove from ends
- d[i] – Access by index
- d.size(), d.empty() – Size and empty check
- d.front(), d.back() – Access ends

🔹 **4. set – Unique Sorted Elements**

- **Header**: #include <set>
- **Syntax**: set<int> s;
- **Key Features**: No duplicates, auto-sorted

Operations:

- s.insert(x) – Insert element
- s.erase(x) – Erase element
- s.find(x) – Find element (returns iterator)
- s.count(x) – Check existence (0 or 1)
- s.lower_bound(x), s.upper_bound(x) – Range queries

🔹 **5. map – Key-Value Pairs (Sorted by Key)**

- **Header**: #include <map>
- **Syntax**: map<int, string> m;
- **Key Features**: Sorted keys, fast access

Operations:

- m[key] = value; – Insert/update
- m.at(key) – Access value
- m.erase(key) – Remove key

- m.find(key) – Iterator to key
- m.count(key) – 0 or 1

🔹 **6. unordered_set / unordered_map**

- **Header**: #include <unordered_set> / #include <unordered_map>
- **Syntax**: unordered_set<int> us;, unordered_map<int, string> um;
- **Key Features**: Faster avg lookup, no order

Operations (same as set/map):

- insert(), erase(), find(), count()
- No sorting supported

🔁 **II. Container Adapters (Use existing containers internally)**

These are not containers themselves but provide restricted access based on rules.

🔹 **1. stack – LIFO**

- **Header**: #include <stack>
- **Syntax**: stack<int> st;
- **Internally uses**: deque (default)

Operations:

- st.push(x) – Push element
- st.pop() – Pop top
- st.top() – Access top element
- st.empty(), st.size() – Status check

🔹 **2. queue – FIFO**

- **Header**: #include <queue>
- **Syntax**: queue<int> q;

Operations:

- q.push(x) – Enqueue
- q.pop() – Dequeue
- q.front(), q.back() – Access ends
- q.empty(), q.size() – Status check

🔹 **3. priority_queue – Max Heap by default**

- **Header**: #include <queue>
- **Syntax**: priority_queue<int> pq;
- **Min Heap**: priority_queue<int, vector<int>, greater<int>> pq;

Operations:

- pq.push(x) – Insert
- pq.pop() – Remove top
- pq.top() – Access top element
- pq.empty(), pq.size() – Status check

🧠 **Bonus: STL Utility Functions (#include <algorithm>)**

Function	Purpose	Syntax Example
sort()	Sort a container	sort(v.begin(), v.end());
reverse()	Reverse order	reverse(v.begin(), v.end());
find()	Linear search	find(v.begin(), v.end(), 5);
count()	Count occurrences	count(v.begin(), v.end(), 2);
binary_search()	Sorted search	binary_search(v.begin(), v.end(), 3);
max_element()	Max value	*max_element(v.begin(), v.end())
min_element()	Min value	*min_element(v.begin(), v.end())

Let me know if you want a **PDF version**, or if you'd like a **visual chart** version for wall pin-up 📌 !