🧰C++ STL Data Structures – Clean Variable Names, Methods & Usage

A guide to using STL containers with meaningful variable names, methods (push, pop, insert, etc.), and examples.

---

1. 📦`vector<T>` – Dynamic Array

Use Case: Store list of elements with random access and dynamic size.

```
vector<int> numbers;     // declaration
numbers.push_back(10);   // insert at end
numbers.pop_back();      // remove from end
int x = numbers[0];      // access
```

✅Recommended names: `numbers`, `values`, `studentMarks`, `prices`

---

2. 🚀`stack<T>` – LIFO Structure

Use Case: Reversing, backtracking, parsing

```
stack<int> numStack;     // declaration
numStack.push(5);        // push element
numStack.pop();          // pop top
int topVal = numStack.top();  // access top
```

✅Recommended names: `charStack`, `undoStack`, `pathStack`

---

3. 📬`queue<T>` – FIFO Structure

Use Case: Level Order Traversal, Task Scheduling

```
queue<int> taskQueue;
taskQueue.push(1);       // enqueue
int frontVal = taskQueue.front(); // access front
taskQueue.pop();         // dequeue
```

✅Recommended names: `nodeQueue`, `taskQueue`, `dataQueue`

---

4. 🔦 `deque<T>` – Double-Ended Queue

Use Case: Sliding Window, Palindrome Checking

```
deque<int> dq;
dq.push_back(1);        // insert at end
dq.push_front(2);       // insert at front
dq.pop_back();          // remove from end
dq.pop_front();         // remove from front
```

✅Recommended names: `window`, `charDeque`, `buffer`

---

5. 🧺 `set<T>` – Unique Elements, Sorted

Use Case: Store sorted unique items

```
set<int> uniqueSet;
uniqueSet.insert(5);    // insert
uniqueSet.erase(5);     // remove
bool found = uniqueSet.count(5); // check
```

✅Recommended names: `uniqueIds`, `seenValues`, `visitedNodes`

---

6. 📚 `multiset<T>` – Sorted with Duplicates

```
multiset<int> scoreList;
scoreList.insert(10);
scoreList.erase(scoreList.find(10));
```

✅Recommended names: `scoreList`, `valueMultiset`

---

7. 🗃️ `map<Key, Value>` – Key-Value Store (Sorted)

Use Case: Count frequency, dictionary

```
map<string, int> wordCount;
wordCount["apple"]++;
wordCount.erase("apple");
int freq = wordCount["apple"];
```

✅Recommended names: `wordCount`, `userScores`, `indexMap`

---

8. 📂`multimap<Key, Value>` – Duplicate Keys Allowed

```
multimap<string, int> classToMarks;
classToMarks.insert({"Math", 90});
```

✅Recommended names: `categoryMap`, `groupedData`

---

9. 🔍`unordered_set<T>` – Unique, Unordered

```
unordered_set<int> visited;
visited.insert(3);
visited.erase(3);
bool exists = visited.count(3);
```

✅Recommended names: `visited`, `uniqueItems`

---

10. 🕐`unordered_map<Key, Value>` – Hash Table

```
unordered_map<string, int> freqMap;
freqMap["cat"]++;
freqMap.erase("cat");
```

✅Recommended names: `hashTable`, `wordFrequency`, `userMap`

---

11. 📋`priority_queue<T>` – Max Heap (Default)

Use Case: Greedy, Top-K problems

```
priority_queue<int> maxHeap;
priority_queue<int, vector<int>, greater<int>> minHeap;

maxHeap.push(5);
maxHeap.pop();
int maxVal = maxHeap.top();
```

✅Recommended names: `maxHeap`, `minHeap`, `taskPriority`

💡Naming Tips

Use Case	Suggested Variable Names
BFS / Trees	`nodeQueue`, `levelQueue`
Frequency Map	`wordCount`, `freqMap`
Stack Uses	`operationStack`, `undoStack`
Unique Tracking	`seenSet`, `visitedNodes`
Arrays / Lists	`values`, `dataList`

✅ Use **camelCase** or **snake_case** consistently\ ✅Pick names that reflect the container's real-world meaning

---