C++ STL Cheatsheet - Version 1.0.0

Essentials for Competitive Programming and Technical Interviews

By Habibur Rahman | ML Developer, Ex-Software Engineer

February 4, 2024

Containers

Vector (`vector`)

- **Operations**: Access (`O(1)`), Insert/Delete at end (`O(1)` amortized), Insert/Delete elsewhere (`O(n)`).
- **Note**: `vector` is implemented as a dynamic array.

```
#include <vector>
std::vector<int> vec; // Declaration
vec.push_back(10); // Insert at the end - Amortized `O(1)`
vec.size(); // Get the number of elements - `O(1)`
vec.insert(vec.begin() + i, 20); // Insert at index i (0-based) - `O(n)`
vec.erase(vec.begin() + i); // Remove at index i (0-based) - `O(n)`
vec.pop_back(); // Remove the last element - `O(1)`
vec.clear(); // Remove all elements - `O(n)`
```

Set (`set`)

- **Operations**: Insert/Delete/Search (`O(log n)`).
- **Note**: `set` is implemented as a balanced binary search tree (Red-Black Tree).

```
#include <set>
std::set<int> s;
s.insert(1); // Insert an element - `O(log n)`
s.find(1); // Find an element - `O(log n)`
s.erase(1); // Remove an element - `O(log n)`
s.clear(); // Remove all elements - `O(n)`
s.size(); // Get the number of elements - `O(1)`
```

Map (`map`)

- **Operations**: Insert/Delete/Search by key (`O(log n)`).
- **Note**: `map` is implemented as a balanced binary search tree (Red-Black Tree).

```
#include <map>
std::map<int, int> m;
m[1] = 2; // Insert or update a key-value pair - `O(log n)`
m.emplace(1, 2); // Insert a key-value pair - `O(log n)`
m.find(1); // Find an element by key - `O(log n)`
m.erase(1); // Remove an element by key - `O(log n)`
m.clear(); // Remove all elements - `O(n)`
m.size(); // Get the number of elements - `O(1)`
```

Unordered Set (`unordered_set`)

- **Operations**: Insert/Delete/Search (`Average: O(1)`, Worst: `O(n)`).
- **Note**: `unordered_set` is implemented as a hash table.

```
#include <unordered_set>
std::unordered_set<int> us;
us.insert(1); // Insert an element - `Average: O(1)`, Worst: `O(n)`
us.find(1); // Find an element - `Average: O(1)`, Worst: `O(n)`
us.erase(1); // Remove an element - `Average: O(1)`, Worst: `O(n)`
us.clear(); // Remove all elements - `O(n)`
us.size(); // Get the number of elements - `O(1)`
```

Unordered Map (`unordered_map`)

- **Operations**: Insert/Delete/Search by key (`Average: O(1)`, Worst: `O(n)`).
- **Note**: `unordered_map` is implemented as a hash table.

```
#include <unordered_map>
std::unordered_map<int, int> um;
um[1] = 2; // Insert or update a key-value pair-`Average: O(1)`, Worst: `O(n)`
um.emplace(1, 2); // Insert a key-value pair - `Average: O(1)`, Worst: `O(n)`
um.find(1); // Find an element by key - `Average: O(1)`, Worst: `O(n)`
um.erase(1); // Remove an element by key - `Average: O(1)`, Worst: `O(n)`
um.clear(); // Remove all elements - `O(n)`
um.size(); // Get the number of elements - `O(1)`
```

Stack (`stack`)

- **Operations**: Push/Pop/Top (`O(1)`).
- **Note**: `stack` is implemented as a deque (double-ended queue).

```
#include <stack>
std::stack<int> st;
st.push(10); // Push an element - `O(1)`
st.pop(); // Pop an element - `O(1)`
st.top(); // Get the top element - `O(1)`
st.size(); // Get the number of elements - `O(1)`
st.empty(); // Check if the stack is empty - `O(1)`
st.clear(); // Remove all elements - `O(n)`
```

Queue (`queue`)

- **Operations**: Enqueue/Dequeue/Front (`O(1)`).
- **Note**: `queue` is implemented as a deque (double-ended queue).

```
#include <queue>
std::queue<int> q;
q.push(10); // Enqueue an element - `O(1)`
q.pop(); // Dequeue an element - `O(1)`
q.front(); // Get the front element - `O(1)`
q.size(); // Get the number of elements - `O(1)`
q.empty(); // Check if the queue is empty - `O(1)`
q.clear(); // Remove all elements - `O(n)`
```

Priority Queue (`priority_queue`)

- **Operations**: Insert/Delete Max (`O(log n)`), Get Max (`O(1)`).
- **Note**: `priority_queue` is implemented as a heap.

```
#include <queue>
std::priority_queue<int> pq;
pq.push(10); // Insert an element - `O(log n)`
pq.pop(); // Remove the top element - `O(log n)`
pq.top(); // Get the top element - `O(1)`
pq.size(); // Get the number of elements - `O(1)`
pq.empty(); // Check if the priority queue is empty - `O(1)`
pq.clear(); // Remove all elements - `O(n)`
```

Algorithms

Sort

- **Time Complexity**: `O(n log n)`.
- **Note**: By default, it sorts in ascending order, algorithm used is `IntroSort`.

```
#include <algorithm>
std::sort(vec.begin(), vec.end()); // Ascending
std::sort(vec.begin(), vec.end(), std::greater<int>()); // Descending
```

Binary Search (`binary_search`)

- **Time Complexity**: `O(log n)`.
- **Note**: The array must be sorted.

```
std::binary_search(vec.begin(), vec.end(), val);
```

Lower Bound and Upper Bound

- **Time Complexity**: `O(log n)`.
- **Note**: The array must be sorted.

```
auto lb = std::lower_bound(vec.begin(), vec.end(), val); // Not less than val
auto ub = std::upper_bound(vec.begin(), vec.end(), val); // Greater than val
```

Max Element and Min Element

- **Time Complexity**: `O(n)`.
- **Note**: Returns an iterator to the maximum/minimum element.

```
auto max_it = std::max_element(vec.begin(), vec.end());
auto min_it = std::min_element(vec.begin(), vec.end());
```

Utilities

Pair (`pair`)

- Simple container to store two values.

```
#include <utility>
std::pair<int, int> p = {1, 2};
```

Tuple (`tuple`)

- Generalization of pair to hold more than two items.

```
#include <tuple>
std::tuple<int, char, double> t = {1, 'a', 2.0};
```

Swap

- Exchange the values of two variables. `O(1)`.

```
std::swap(a, b);
```

Reverse

- Reverse the elements of a container. `O(n)`.

```
std::reverse(vec.begin(), vec.end());
```

Next Permutation and Previous Permutation

- Rearrange the elements into the next/previous lexicographically greater permutation. `O(n)`.

```
std::next_permutation(vec.begin(), vec.end());
std::prev_permutation(vec.begin(), vec.end());
```

Fill

- Assigns the given value to the elements in the range. `O(n)`.

```
std::fill(vec.begin(), vec.end(), val);
```

Memory Set

- Fills the first `n` bytes of the memory area pointed to by `ptr` with the constant byte `val`. `O(n)`.

```
#include <cstring>
std::memset(ptr, val, n); // Entire with val std::memset(ptr, -1, sizeof(ptr));
```

Min and Max

- Returns the minimum/maximum of two values.

```
std::min(a, b);
std::max(a, b);
```

Absolute

- Returns the absolute value of a number.

```
std::abs(val);
```

String Stream

- Used to manipulate strings as if they were input/output streams.

```
#include <sstream>
std::stringstream ss;
ss<<"Hello";
ss>>str;
```

String Functions

- `stoi`, `stol`, `stoll`, `stoul`, `stoull`, `stof`, `stod`, `stold`: Convert string to integer/long/long long/unsigned long/unsigned long long/float/double/long double.
- `to_string`: Convert number to string.
- `getline`: Read a line from input stream.

```
std::stoi("10");
std::to_string(10);
std::getline(std::cin, str);
```

Count

- Count the number of occurrences of a value in a range.

```
std::count(vec.begin(), vec.end(), val);
```