C++ Syntax Quick Reference - STL & Data Structures

VECTORS (Dynamic Arrays)

Creation & Basic Operations

```
#include <vector>
#include <algorithm>
#include <numeric>

// Initialize
vector<int> arr;
vector<int> arr(n);          // n elements with default value (0 for int)
vector<int> arr(n, val);     // n elements with value val
vector<int> arr = {1, 2, 3}; // Initializer list
vector<int> arr(other);      // Copy constructor

// 2D vector
vector<vector<int>> matrix(rows, vector<int>(cols, 0));

// Add elements
arr.push_back(x);              // Add to end - O(1)
arr.insert(arr.begin() + i, x); // Insert at index - O(n)
arr.emplace_back(x);           // Construct in place - O(1)

// Remove elements
arr.pop_back();                 // Remove last - O(1)
arr.erase(arr.begin() + i);     // Remove at index - O(n)
arr.erase(arr.begin() + i, arr.begin() + j); // Remove range [i, j)
arr.clear();                    // Remove all

// Access
arr[i];                         // No bounds checking
arr.at(i);                      // With bounds checking (throws exception)
arr.front();                    // First element
arr.back();                     // Last element

// Size operations
arr.size();                     // Number of elements
arr.empty();                    // Check if empty
arr.resize(new_size);           // Change size
```

```
arr.reserve(capacity);          // Reserve space (avoids reallocations)

// Slicing (using iterators)
vector<int> sub(arr.begin() + start, arr.begin() + end); // [start, end)

// Common methods
sort(arr.begin(), arr.end());                    // Sort ascending
sort(arr.rbegin(), arr.rend());                  // Sort descending
reverse(arr.begin(), arr.end());                 // Reverse
auto it = find(arr.begin(), arr.end(), x);       // Find element
int count = count(arr.begin(), arr.end(), x);    // Count occurrences
auto minIt = min_element(arr.begin(), arr.end());
auto maxIt = max_element(arr.begin(), arr.end());
int sum = accumulate(arr.begin(), arr.end(), 0);
```

Vector Tricks

```
// Remove duplicates from sorted vector
sort(arr.begin(), arr.end());
arr.erase(unique(arr.begin(), arr.end()), arr.end());

// Swap vectors
arr1.swap(arr2);

// Fill with value
fill(arr.begin(), arr.end(), value);

// Copy
vector<int> copy = arr;                          // Deep copy
vector<int> copy(arr.begin(), arr.end());        // Using iterators
```

STRINGS

Creation & Operations

```
#include <string>

// Create
string s = "hello";
string s(5, 'a');              // "aaaaa"
string s = to_string(123);     // "123"

// Common operations
```

```
s.length(), s.size();          // Length
s.empty();                     // Check if empty
s.clear();                     // Clear string
s.substr(start, len);          // Substring
s.find("ll");                  // Returns pos or string::npos
s.rfind("ll");                 // Find from right
s.replace(pos, len, "new");    // Replace substring
s.append(" world");            // Append
s += " world";                 // Append

// Character operations
s[i];                          // Access character
s.front();                     // First character
s.back();                      // Last character
s.push_back('!');              // Add character to end
s.pop_back();                  // Remove last character

// Case conversion
transform(s.begin(), s.end(), s.begin(), ::tolower);
transform(s.begin(), s.end(), s.begin(), ::toupper);

// Trimming
s.erase(0, s.find_first_not_of(" \t\n\r")); // Left trim
s.erase(s.find_last_not_of(" \t\n\r") + 1); // Right trim

// Split string
vector<string> split(string s, char delim) {
    vector<string> result;
    stringstream ss(s);
    string item;
    while (getline(ss, item, delim)) {
        result.push_back(item);
    }
    return result;
}

// Join strings
string joined;
for (int i = 0; i < words.size(); i++) {
    joined += words[i];
    if (i < words.size() - 1) joined += " ";
}

// String comparison
s1 == s2;                      // Equal
s1 < s2;                       // Lexicographical comparison
s1.compare(s2);                // Returns 0, <0, or >0
```

String Manipulation

```
// Reverse
reverse(s.begin(), s.end());
string reversed(s.rbegin(), s.rend());

// Sort characters
sort(s.begin(), s.end());

// Check palindrome
bool isPalindrome(string s) {
    return equal(s.begin(), s.begin() + s.size()/2, s.rbegin());
}

// Count characters
unordered_map<char, int> freq;
for (char c : s) {
    freq[c]++;
}
```

HASH TABLES

unordered_map (Hash Map)

```
#include <unordered_map>

// Create
unordered_map<int, int> m;
unordered_map<string, int> m = {{"a", 1}, {"b", 2}};

// Access
m[key];                         // Creates key if doesn't exist
m.at(key);                      // Throws exception if doesn't exist

// Add/Update
m[key] = value;
m.insert({key, value});
m.emplace(key, value);

// Remove
m.erase(key);
m.clear();
```

```
// Check existence
if (m.count(key)) { }          // Returns 0 or 1
if (m.find(key) != m.end()) { }

// Iterate
for (auto& [key, value] : m) {
    cout << key << ": " << value << endl;
}
for (auto& pair : m) {
    cout << pair.first << ": " << pair.second << endl;
}

// Size
m.size();
m.empty();
```

map (Ordered Map - Red-Black Tree)

```
#include <map>

// Same interface as unordered_map, but maintains sorted order
map<int, int> m;

// Iterate in sorted order
for (auto& [key, value] : m) {
    // Keys are sorted
}

// Find operations
auto it = m.lower_bound(key); // First element >= key
auto it = m.upper_bound(key); // First element > key
```

SETS

unordered_set (Hash Set)

```
#include <unordered_set>

// Create
unordered_set<int> s;
unordered_set<int> s = {1, 2, 3};

// Add
```

```
s.insert(x);
s.emplace(x);

// Remove
s.erase(x);
s.clear();

// Check membership
if (s.count(x)) { }
if (s.find(x) != s.end()) { }

// Size
s.size();
s.empty();

// Iterate
for (int x : s) {
    cout << x << endl;
}

// Set operations (requires converting to set or manual implementation)
// Union, intersection, difference are easier with std::set
```

set (Ordered Set - Red-Black Tree)

```
#include <set>

// Same interface as unordered_set, but maintains sorted order
set<int> s;

// Iterate in sorted order
for (int x : s) {
    // Elements are sorted
}

// Find operations
auto it = s.lower_bound(x); // First element >= x
auto it = s.upper_bound(x); // First element > x

// Set operations
set<int> result;
set_union(s1.begin(), s1.end(), s2.begin(), s2.end(),
          inserter(result, result.begin()));
set_intersection(s1.begin(), s1.end(), s2.begin(), s2.end(),
                 inserter(result, result.begin()));
set_difference(s1.begin(), s1.end(), s2.begin(), s2.end(),
```

```
                inserter(result, result.begin()));
```

STACKS

```
#include <stack>

// Create
stack<int> st;

// Push
st.push(x);
st.emplace(x);

// Pop
if (!st.empty()) {
    st.pop();
}

// Peek
if (!st.empty()) {
    int top = st.top();
}

// Check empty
st.empty();

// Size
st.size();
```

Using Vector as Stack

```
vector<int> st;

// Push
st.push_back(x);

// Pop
if (!st.empty()) {
    st.pop_back();
}

// Peek
if (!st.empty()) {
    int top = st.back();
```

```
}

// Empty
st.empty();
```

QUEUES

queue (FIFO)

```
#include <queue>

// Create
queue<int> q;

// Enqueue
q.push(x);
q.emplace(x);

// Dequeue
if (!q.empty()) {
    q.pop();
}

// Peek
if (!q.empty()) {
    int front = q.front();
    int back = q.back();
}

// Size/Empty
q.size();
q.empty();
```

deque (Double-ended Queue)

```
#include <deque>

// Create
deque<int> dq;

// Add elements
dq.push_back(x);    // Add to right
dq.push_front(x);   // Add to left
```

```
dq.emplace_back(x);
dq.emplace_front(x);

// Remove elements
dq.pop_back();      // Remove from right
dq.pop_front();     // Remove from left

// Access
dq[i];              // Random access
dq.front();
dq.back();

// Size/Empty
dq.size();
dq.empty();
```

priority_queue (Heap)

```
#include <queue>

// Min heap (default is max heap)
priority_queue<int, vector<int>, greater<int>> minHeap;

// Max heap
priority_queue<int> maxHeap;

// Push
minHeap.push(x);
minHeap.emplace(x);

// Pop
if (!minHeap.empty()) {
    minHeap.pop();
}

// Peek
if (!minHeap.empty()) {
    int top = minHeap.top();
}

// Size/Empty
minHeap.size();
minHeap.empty();

// Custom comparator (for pairs, etc.)
auto cmp = [](pair<int, int> a, pair<int, int> b) {
```

```
    return a.first > b.first; // Min heap by first element
};
priority_queue<pair<int, int>, vector<pair<int, int>>, decltype(cmp)> pq(cmp);
```

Heap Operations on Vectors

```
#include <algorithm>

vector<int> heap;

// Make heap (max heap by default)
make_heap(heap.begin(), heap.end());

// Min heap
make_heap(heap.begin(), heap.end(), greater<int>());

// Push
heap.push_back(x);
push_heap(heap.begin(), heap.end());

// Pop
pop_heap(heap.begin(), heap.end());
heap.pop_back();

// Access top
int top = heap.front();
```

PAIRS AND TUPLES

pair

```
#include <utility>

// Create
pair<int, int> p = {1, 2};
pair<int, string> p = make_pair(1, "hello");
auto p = make_pair(1, 2);

// Access
p.first;
p.second;

// Structured binding (C++17)
```

```
auto [a, b] = p;

// Comparison (lexicographical)
p1 < p2; // Compares first, then second
```

tuple

```
#include <tuple>

// Create
tuple<int, int, int> t = {1, 2, 3};
auto t = make_tuple(1, 2, 3);

// Access
get<0>(t); // First element
get<1>(t); // Second element

// Structured binding (C++17)
auto [a, b, c] = t;

// Tie (for unpacking)
int a, b, c;
tie(a, b, c) = t;

// Ignore some elements
tie(a, ignore, c) = t;
```

USEFUL ALGORITHMS

Sorting & Searching

```
#include <algorithm>

// Sort
sort(arr.begin(), arr.end());                    // Ascending
sort(arr.begin(), arr.end(), greater<int>());    // Descending

// Custom comparator
sort(arr.begin(), arr.end(), [](int a, int b) {
    return a > b; // Descending
});

// Stable sort (maintains relative order of equal elements)
```

```
stable_sort(arr.begin(), arr.end());

// Partial sort (only sort first k elements)
partial_sort(arr.begin(), arr.begin() + k, arr.end());

// nth_element (partition around nth element)
nth_element(arr.begin(), arr.begin() + k, arr.end());

// Binary search (array must be sorted)
bool found = binary_search(arr.begin(), arr.end(), target);
auto it = lower_bound(arr.begin(), arr.end(), target); // First >= target
auto it = upper_bound(arr.begin(), arr.end(), target); // First > target

// Linear search
auto it = find(arr.begin(), arr.end(), target);
if (it != arr.end()) {
    int index = it - arr.begin();
}
```

Min/Max

```
// Element-wise
auto minIt = min_element(arr.begin(), arr.end());
auto maxIt = max_element(arr.begin(), arr.end());

// Value comparison
int minimum = min(a, b);
int maximum = max(a, b);
int minimum = min({a, b, c, d}); // From initializer list

// Min/max pair
auto [minIt, maxIt] = minmax_element(arr.begin(), arr.end());
```

Permutations & Combinations

```
// Next permutation
do {
    // Process current permutation
} while (next_permutation(arr.begin(), arr.end()));

// Previous permutation
do {
    // Process current permutation
} while (prev_permutation(arr.begin(), arr.end()));
```

Numeric Operations

```
#include <numeric>

// Sum
int sum = accumulate(arr.begin(), arr.end(), 0);
long long sum = accumulate(arr.begin(), arr.end(), 0LL); // For large sums

// Product
int product = accumulate(arr.begin(), arr.end(), 1, multiplies<int>());

// GCD and LCM (C++17)
int g = gcd(a, b);
int l = lcm(a, b);

// Partial sum (prefix sum)
vector<int> prefix(arr.size());
partial_sum(arr.begin(), arr.end(), prefix.begin());

// Iota (fill with incrementing values)
vector<int> arr(n);
iota(arr.begin(), arr.end(), 0); // Fills with 0, 1, 2, ..., n-1
```

Other Useful Algorithms

```
// Fill
fill(arr.begin(), arr.end(), value);

// Count
int cnt = count(arr.begin(), arr.end(), value);
int cnt = count_if(arr.begin(), arr.end(), [](int x) { return x > 0; });

// Reverse
reverse(arr.begin(), arr.end());

// Rotate
rotate(arr.begin(), arr.begin() + k, arr.end()); // Rotate left by k

// Remove duplicates (works on sorted arrays)
auto last = unique(arr.begin(), arr.end());
arr.erase(last, arr.end());

// Partition
auto it = partition(arr.begin(), arr.end(), [](int x) { return x % 2 == 0; });
// Even numbers before it, odd numbers after
```

```
// Check if sorted
bool is_sorted_asc = is_sorted(arr.begin(), arr.end());

// All of, any of, none of
bool all_positive = all_of(arr.begin(), arr.end(), [](int x) { return x > 0; });
bool any_negative = any_of(arr.begin(), arr.end(), [](int x) { return x < 0; });
bool none_zero = none_of(arr.begin(), arr.end(), [](int x) { return x == 0; });
```

COMMON MATH OPERATIONS

```
#include <cmath>

// Basic
abs(x);                         // Absolute value
pow(x, y);                      // x^y
sqrt(x);                        // Square root
cbrt(x);                        // Cube root

// Rounding
ceil(x);                        // Round up
floor(x);                       // Round down
round(x);                       // Round to nearest
trunc(x);                       // Truncate decimal

// Trig (radians)
sin(x), cos(x), tan(x);
asin(x), acos(x), atan(x);

// Logarithms
log(x);                         // Natural log
log10(x);                       // Base 10 log
log2(x);                        // Base 2 log

// Other
exp(x);                         // e^x
fmod(x, y);                     // Floating point modulo
hypot(x, y);                    // sqrt(x² + y²)

// Constants
M_PI;                           // π (may need -D_USE_MATH_DEFINES on some compile

// Min/Max
min(a, b);
max(a, b);
```

ITERATION

Range-based For Loop

```
// Iterate by value
for (int x : arr) {
    cout << x << endl;
}

// Iterate by reference (modify elements)
for (int& x : arr) {
    x *= 2;
}

// Iterate by const reference (no copy, can't modify)
for (const int& x : arr) {
    cout << x << endl;
}

// With auto
for (auto x : arr) { }
for (auto& x : arr) { }
for (const auto& x : arr) { }

// Iterate pairs/tuples
for (auto& [key, value] : map) {
    cout << key << ": " << value << endl;
}
```

Iterator Loops

```
// Forward
for (auto it = arr.begin(); it != arr.end(); ++it) {
    cout << *it << endl;
}

// Reverse
for (auto it = arr.rbegin(); it != arr.rend(); ++it) {
    cout << *it << endl;
}
```

COMMON TRICKS

Swap Variables

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

Multiple Assignment

```
int a = 1, b = 2, c = 3;
a = b = c = 0;
```

Ternary Operator

```
int value = (condition) ? x : y;
```

Integer Division

```
int quotient = a / b;
int remainder = a % b;

// Division and modulo at once
auto [q, r] = div(a, b); // Returns div_t with quot and rem fields
```

Bit Operations

```
x & y;    // AND
x | y;    // OR
x ^ y;    // XOR
~x;       // NOT
x << n;   // Left shift (multiply by 2^n)
x >> n;   // Right shift (divide by 2^n)

// Set bit
x |= (1 << i);

// Clear bit
x &= ~(1 << i);

// Toggle bit
x ^= (1 << i);
```

```
// Check bit
bool isSet = (x & (1 << i)) != 0;

// Count set bits
int count = __builtin_popcount(x);          // For int
int count = __builtin_popcountll(x);        // For long long

// Leading zeros
int lz = __builtin_clz(x);                  // For int
int lz = __builtin_clzll(x);                // For long long

// Trailing zeros
int tz = __builtin_ctz(x);                  // For int
int tz = __builtin_ctzll(x);                // For long long
```

String Multiplication

```
string s(5, 'a'); // "aaaaa"
```

Check Multiple Conditions

```
if (x == 1 || x == 2 || x == 3) { }

// Using set
set<int> valid = {1, 2, 3, 4, 5};
if (valid.count(x)) { }

// Range check
if (x >= 1 && x <= 10) { }
```

ASCII / CHARACTER OPERATIONS

```
#include <cctype>

// Character to ASCII
int ascii = 'a';              // 97
int ascii = 'A';              // 65
int ascii = '0';              // 48

// ASCII to character
char c = 97;                  // 'a'
char c = static_cast<char>(97);
```

```
// Character checks
isalpha(c);                  // Letter
isdigit(c);                  // Digit
isalnum(c);                  // Letter or digit
islower(c);                  // Lowercase
isupper(c);                  // Uppercase
isspace(c);                  // Whitespace

// Case conversion
char lower = tolower(c);
char upper = toupper(c);
```

INPUT/OUTPUT FOR HACKERRANK

Fast Input/Output

```
#include <iostream>
using namespace std;

// Speed up cin/cout
ios_base::sync_with_stdio(false);
cin.tie(NULL);

// Read single values
int n;
cin >> n;

// Read array
vector<int> arr(n);
for (int i = 0; i < n; i++) {
    cin >> arr[i];
}

// Read string
string s;
cin >> s;              // Reads until whitespace
getline(cin, s);       // Reads entire line

// Output
cout << value << endl;
cout << value << "\n"; // Slightly faster than endl
```

Using scanf/printf (Faster)

```
#include <cstdio>

// Input
int n;
scanf("%d", &n);

double x;
scanf("%lf", &x);

char s[100];
scanf("%s", s);

// Output
printf("%d\n", n);
printf("%lf\n", x);
printf("%.2f\n", x);  // 2 decimal places
```

GOTCHAS TO REMEMBER

Array/Vector Initialization

```
// WRONG - All rows point to same array
vector<vector<int>> matrix(rows, vector<int>(cols)); // Actually this is CORRECT

// 2D arrays - C-style
int arr[rows][cols];

// Initialize with value
vector<vector<int>> matrix(rows, vector<int>(cols, value));
```

Pass by Reference vs Value

```
// Pass by value (creates copy)
void func(vector<int> arr) { }

// Pass by reference (no copy, can modify)
void func(vector<int>& arr) { }

// Pass by const reference (no copy, can't modify)
void func(const vector<int>& arr) { }
```

Integer Overflow

```
int a = 1000000;
int b = 1000000;
int c = a * b; // OVERFLOW!

// Use long long
long long c = (long long)a * b;
// Or
long long c = 1LL * a * b;
```

Division

```
int a = 7, b = 2;
int result = a / b;         // 3 (integer division)
double result = a / b;      // 3.0 (still integer division)
double result = (double)a / b; // 3.5 (float division)
```

Comparison in Loops

```
// WRONG - unsigned comparison
for (int i = arr.size() - 1; i >= 0; i--) { } // Infinite loop if arr.size() is u

// RIGHT
for (int i = (int)arr.size() - 1; i >= 0; i--) { }
// Or
for (size_t i = arr.size(); i-- > 0; ) { }
```

COMMON EDGE CASES

```
// Empty input
if (arr.empty()) {
    return default_value;
}

// Single element
if (arr.size() == 1) {
    return arr[0];
}

// Negative numbers
if (x < 0) {
    handle_negative();
}
```

```
// Out of bounds
if (i >= 0 && i < arr.size()) {
    safe_access();
}

// Division by zero
if (denominator != 0) {
    result = numerator / denominator;
}

// Overflow check
if (a > INT_MAX - b) {
    // a + b would overflow
}
```

TEMPLATE FOR COMPETITIVE PROGRAMMING

```
#include <bits/stdc++.h>
using namespace std;

#define ll long long
#define vi vector<int>
#define vll vector<long long>
#define pii pair<int, int>
#define pb push_back
#define mp make_pair
#define F first
#define S second
#define all(x) x.begin(), x.end()

const int MOD = 1e9 + 7;
const int INF = 1e9;
const ll LLINF = 1e18;

void solve() {
    // Solution here
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t = 1;
    // cin >> t; // Uncomment for multiple test cases
```

```
    while (t--) {
        solve();
    }

    return 0;
}
```