C++ STL Libraries Cheat Sheet

Table of Contents

Algorithm Library

`#include <algorithm>`

Sorting Algorithms

```cpp
// Basic sorting
sort(v.begin(), v.end());                    // Ascending order
sort(v.begin(), v.end(), greater<int>());    // Descending order
sort(v.begin(), v.end(), [](int a, int b) { return a > b; }); // Custom comparator

// Partial sorting
partial_sort(v.begin(), v.begin() + 3, v.end()); // Sort first 3 elements
nth_element(v.begin(), v.begin() + 2, v.end());  // 3rd element in sorted position

// Stability
stable_sort(v.begin(), v.end());             // Maintains relative order of equal elem

// Check if sorted
is_sorted(v.begin(), v.end());               // Returns true if sorted
is_sorted_until(v.begin(), v.end());         // Returns iterator to first unsorted ele
```

Searching Algorithms

```cpp
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

// Linear search
auto it = find(v.begin(), v.end(), 5);       // Find first occurrence
auto it2 = find_if(v.begin(), v.end(), [](int x) { return x > 5; }); // Find with cond
auto it3 = find_if_not(v.begin(), v.end(), [](int x) { return x < 5; }); // Find NOT c

// Binary search (requires sorted container)
bool found = binary_search(v.begin(), v.end(), 5);  // Returns true/false
auto lower = lower_bound(v.begin(), v.end(), 5);    // First position where 5 could be
auto upper = upper_bound(v.begin(), v.end(), 5);    // Last position where 5 could be
auto range = equal_range(v.begin(), v.end(), 5);    // Returns pair of lower_bound and

// Search for subsequence
vector<int> pattern = {3, 4, 5};
auto it4 = search(v.begin(), v.end(), pattern.begin(), pattern.end());
```

Modifying Algorithms

cpp

```cpp
vector<int> v = {1, 2, 3, 4, 5};
vector<int> dest(10);

// Copy operations
copy(v.begin(), v.end(), dest.begin());              // Copy all elements
copy_n(v.begin(), 3, dest.begin());                 // Copy first 3 elements
copy_if(v.begin(), v.end(), dest.begin(), [](int x) { return x > 2; }); // Copy with c

// Fill operations
fill(v.begin(), v.end(), 42);                       // Fill with value
fill_n(v.begin(), 3, 42);                          // Fill first 3 with value
generate(v.begin(), v.end(), []() { return rand(); }); // Generate with function

// Transform operations
transform(v.begin(), v.end(), v.begin(), [](int x) { return x * 2; }); // In-place tra
transform(v.begin(), v.end(), dest.begin(), [](int x) { return x * 2; }); // Transform

// Remove operations
auto new_end = remove(v.begin(), v.end(), 3);       // Remove all 3s (doesn't resize)
v.erase(new_end, v.end());                          // Actually remove elements
auto new_end2 = remove_if(v.begin(), v.end(), [](int x) { return x > 5; }); // Remove

// Replace operations
replace(v.begin(), v.end(), 3, 99);                 // Replace all 3s with 99
replace_if(v.begin(), v.end(), [](int x) { return x > 5; }, 99); // Replace with condi

// Reverse and rotate
reverse(v.begin(), v.end());                        // Reverse container
rotate(v.begin(), v.begin() + 2, v.end());         // Rotate left by 2 positions
```

Set Operations (on sorted ranges)

```cpp
vector<int> v1 = {1, 2, 3, 4, 5};
vector<int> v2 = {3, 4, 5, 6, 7};
vector<int> result(10);

// Set operations
auto end1 = set_union(v1.begin(), v1.end(), v2.begin(), v2.end(), result.begin());
auto end2 = set_intersection(v1.begin(), v1.end(), v2.begin(), v2.end(), result.begin(
auto end3 = set_difference(v1.begin(), v1.end(), v2.begin(), v2.end(), result.begin())
auto end4 = set_symmetric_difference(v1.begin(), v1.end(), v2.begin(), v2.end(), resul

// Check subset/superset
bool is_subset = includes(v2.begin(), v2.end(), v1.begin(), v1.end());
```

Min/Max Operations

```cpp
vector<int> v = {3, 1, 4, 1, 5, 9, 2, 6};

// Single element min/max
int min_val = *min_element(v.begin(), v.end());
int max_val = *max_element(v.begin(), v.end());
auto minmax_pair = minmax_element(v.begin(), v.end());

// Compare values
int smaller = min(a, b);
int larger = max(a, b);
auto pair_result = minmax(a, b);

// Lexicographical comparison
bool is_less = lexicographical_compare(v1.begin(), v1.end(), v2.begin(), v2.end());
```

Permutation Operations

cpp

```cpp
vector<int> v = {1, 2, 3};

// Generate permutations
do {
    // Process current permutation
    for (int x : v) cout << x << " ";
    cout << endl;
} while (next_permutation(v.begin(), v.end()));

// Previous permutation
prev_permutation(v.begin(), v.end());

// Check if permutation
vector<int> v2 = {3, 1, 2};
bool is_perm = is_permutation(v.begin(), v.end(), v2.begin());
```

Heap Operations

cpp

```cpp
vector<int> v = {3, 1, 4, 1, 5, 9, 2, 6};

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

// Heap operations
push_heap(v.begin(), v.end());               // After adding element to end
pop_heap(v.begin(), v.end());                // Move max to end
sort_heap(v.begin(), v.end());               // Sort heap (destroys heap property)

// Check heap
bool is_heap_check = is_heap(v.begin(), v.end());
auto heap_end = is_heap_until(v.begin(), v.end());
```

Vector Library

`#include <vector>`

Construction and Initialization

cpp

```cpp
// Different ways to create vectors
vector<int> v1;                             // Empty vector
vector<int> v2(10);                         // 10 elements, default initialized
vector<int> v3(10, 5);                      // 10 elements, all set to 5
vector<int> v4 = {1, 2, 3, 4, 5};          // Initializer list
vector<int> v5(v4);                         // Copy constructor
vector<int> v6(v4.begin(), v4.end());       // Range constructor
vector<int> v7(move(v4));                   // Move constructor
```

Element Access

cpp

```cpp
vector<int> v = {1, 2, 3, 4, 5};

// Access methods
v[2];                    // Direct access (no bounds checking)
v.at(2);                 // Safe access (throws exception if out of bounds)
v.front();               // First element
v.back();                // Last element
v.data();                // Pointer to underlying array
```

Capacity and Size

cpp

```cpp
// Size information
v.size();                // Number of elements
v.max_size();            // Maximum possible size
v.capacity();            // Current capacity
v.empty();               // Check if empty

// Capacity management
v.reserve(100);          // Reserve space for 100 elements
v.shrink_to_fit();       // Reduce capacity to fit size
v.resize(10);            // Resize to 10 elements
v.resize(15, 42);        // Resize to 15 elements, new elements = 42
```

Modifiers

cpp

```cpp
// Adding elements
v.push_back(10);          // Add to end
v.emplace_back(args);     // Construct in place at end
v.insert(v.begin() + 2, 99);              // Insert at position
v.insert(v.begin(), 3, 88);              // Insert 3 copies of 88
v.insert(v.begin(), {1, 2, 3});          // Insert initializer list
v.emplace(v.begin() + 2, args);          // Construct in place at position

// Removing elements
v.pop_back();             // Remove last element
v.erase(v.begin() + 2);   // Remove element at position
v.erase(v.begin() + 1, v.begin() + 4); // Remove range
v.clear();                // Remove all elements

// Other modifiers
v.assign(5, 10);          // Assign 5 elements with value 10
v.assign({1, 2, 3, 4});   // Assign from initializer list
v.swap(other_vector);     // Swap with another vector
```

Iterators

cpp

```cpp
// Iterator types
vector<int>::iterator it = v.begin();
vector<int>::const_iterator cit = v.cbegin();
vector<int>::reverse_iterator rit = v.rbegin();

// Iterator methods
v.begin() / v.end();      // Forward iterators
v.cbegin() / v.cend();    // Const forward iterators
v.rbegin() / v.rend();    // Reverse iterators
v.crbegin() / v.crend(); // Const reverse iterators
```

Vector<bool> Specialization

cpp

```cpp
vector<bool> bv(10, true);
bv.flip();                // Flip all bits
bv[0].flip();             // Flip specific bit
```

String Library

`#include <string>`

Construction and Assignment

```cpp
// Construction
string s1;                    // Empty string
string s2("Hello");           // From C-string
string s3(s2);                // Copy constructor
string s4(10, 'A');           // 10 'A' characters
string s5(s2, 1, 3);          // Substring from s2
string s6 = {'H', 'e', 'l', 'l', 'o'}; // From initializer list

// Assignment
s1 = "World";
s1 = s2;
s1 = 'X';
s1.assign("Hello");
s1.assign(s2, 1, 3);          // Assign substring
s1.assign(10, 'A');           // Assign 10 'A's
```

Element Access

```cpp
string s = "Hello World";

s[0];                    // Direct access
s.at(0);                 // Safe access (bounds checking)
s.front();               // First character
s.back();                // Last character
s.data();                // Pointer to C-string
s.c_str();               // Null-terminated C-string
```

Capacity and Size

cpp

```cpp
s.size();                // Number of characters
s.length();              // Same as size()
s.max_size();            // Maximum possible size
s.empty();               // Check if empty
s.capacity();            // Current capacity
s.reserve(100);          // Reserve capacity
s.shrink_to_fit();       // Reduce capacity to fit
s.resize(10);            // Resize string
s.resize(15, 'X');       // Resize and fill with 'X'
```

String Operations

cpp

```cpp
string s1 = "Hello";
string s2 = "World";

// Concatenation
string s3 = s1 + " " + s2;
s1 += s2;
s1.append(s2);
s1.append(s2, 1, 3);     // Append substring
s1.append(3, 'X');       // Append 3 'X's

// Insertion
s1.insert(5, " Beautiful");
s1.insert(5, s2);
s1.insert(5, s2, 1, 3);  // Insert substring
s1.insert(5, 3, 'X');    // Insert 3 'X's

// Replacement
s1.replace(0, 5, "Hi");
s1.replace(0, 5, s2);
s1.replace(0, 5, s2, 1, 3);

// Erasure
s1.erase(5, 3);          // Erase 3 characters from position 5
s1.erase(5);             // Erase from position 5 to end
s1.pop_back();           // Remove last character
s1.clear();              // Clear string
```

String Searching

cpp

```cpp
string s = "Hello World Hello";

// Find operations
size_t pos = s.find("World");           // Find first occurrence
size_t pos2 = s.find("World", 5);       // Find starting from position 5
size_t pos3 = s.find('o');              // Find character
size_t pos4 = s.rfind("Hello");         // Find last occurrence

// Find character from set
size_t pos5 = s.find_first_of("aeiou"); // Find first vowel
size_t pos6 = s.find_last_of("aeiou");  // Find last vowel
size_t pos7 = s.find_first_not_of("Helo "); // Find first char not in set
size_t pos8 = s.find_last_not_of("Helo "); // Find last char not in set

// Check result
if (pos != string::npos) {
    // Found
}
```

String Comparison

cpp

```cpp
string s1 = "Hello";
string s2 = "World";

// Comparison operators
bool equal = (s1 == s2);
bool less = (s1 < s2);
bool greater = (s1 > s2);

// Compare method
int result = s1.compare(s2);        // Returns <0, 0, or >0
int result2 = s1.compare(1, 3, s2); // Compare substring
int result3 = s1.compare(1, 3, s2, 1, 3); // Compare substrings
```

Substring Operations

```cpp
string s = "Hello World";

string sub = s.substr();              // Entire string
string sub2 = s.substr(6);            // From position 6 to end
string sub3 = s.substr(6, 5);         // 5 characters from position 6
```

String Conversion

```cpp
// String to number
string num_str = "123";
int i = stoi(num_str);                // String to int
long l = stol(num_str);               // String to long
float f = stof(num_str);              // String to float
double d = stod(num_str);             // String to double

// Number to string
int num = 123;
string s = to_string(num);            // Int to string
string s2 = to_string(3.14);          // Double to string
```

String Iterators

cpp

```cpp
string s = "Hello";

// Iterator operations
for (auto it = s.begin(); it != s.end(); ++it) {
    cout << *it;
}

// Range-based for loop
for (char c : s) {
    cout << c;
}

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

File Stream Library

`#include <fstream>`, `#include <iostream>`

File Stream Types

cpp

```cpp
ifstream inFile;          // Input file stream
ofstream outFile;         // Output file stream
fstream file;             // Both input and output
```

Opening and Closing Files

cpp

```cpp
// Opening files
ifstream inFile("input.txt");
ofstream outFile("output.txt");
fstream file("data.txt", ios::in | ios::out);

// Alternative opening
ifstream inFile2;
inFile2.open("input.txt");

// File modes
ios::in         // Input
ios::out        // Output
ios::app        // Append
ios::ate        // At end
ios::trunc      // Truncate
ios::binary     // Binary mode

// Opening with modes
ofstream outFile("file.txt", ios::out | ios::app);

// Closing files
inFile.close();
outFile.close();
file.close();
```

File State and Error Checking

cpp

```cpp
// Check if file is open
if (inFile.is_open()) {
    // File operations
}

// Check stream state
if (inFile.good()) { /* All good */ }
if (inFile.eof()) { /* End of file */ }
if (inFile.fail()) { /* Operation failed */ }
if (inFile.bad()) { /* Read/write error */ }

// Clear error flags
inFile.clear();
```

Reading from Files

cpp

```cpp
ifstream inFile("input.txt");

// Reading methods
string line;
getline(inFile, line);                // Read entire line

string word;
inFile >> word;                       // Read word (whitespace delimited)

char ch;
inFile >> ch;                         // Read character
inFile.get(ch);                       // Read character including whitespace

// Reading numbers
int number;
inFile >> number;

// Reading entire file
string content((istreambuf_iterator<char>(inFile)),
               istreambuf_iterator<char>());

// Line by line reading
while (getline(inFile, line)) {
    cout << line << endl;
}

// Word by word reading
while (inFile >> word) {
    cout << word << " ";
}
```

Writing to Files

cpp

```cpp
ofstream outFile("output.txt");

// Writing methods
outFile << "Hello World" << endl;
outFile << 123 << " " << 45.67 << endl;

string text = "Sample text";
outFile << text << endl;

// Writing characters
outFile.put('A');

// Writing binary data
int data = 42;
outFile.write(reinterpret_cast<char*>(&data), sizeof(data));
```

File Positioning

cpp

```cpp
fstream file("data.txt", ios::in | ios::out);

// Get current position
streampos pos = file.tellg();       // Get position (input)
streampos pos2 = file.tellp();      // Get position (output)

// Set position
file.seekg(0, ios::beg);            // Go to beginning
file.seekg(0, ios::end);            // Go to end
file.seekg(10, ios::cur);           // Move 10 positions from current
file.seekg(pos);                    // Go to specific position

// For output
file.seekp(0, ios::beg);
file.seekp(0, ios::end);
```

Binary File Operations

```cpp
// Binary file handling
ifstream binFile("data.bin", ios::binary);
ofstream binOut("output.bin", ios::binary);

// Reading binary data
int data;
binFile.read(reinterpret_cast<char*>(&data), sizeof(data));

// Writing binary data
int value = 42;
binOut.write(reinterpret_cast<const char*>(&value), sizeof(value));

// Reading/writing arrays
int array[10];
binFile.read(reinterpret_cast<char*>(array), sizeof(array));
binOut.write(reinterpret_cast<const char*>(array), sizeof(array));
```

String Streams

`#include <sstream>`

cpp

```cpp
// String stream for parsing
string data = "123 45.67 Hello";
istringstream iss(data);
int i;
double d;
string s;
iss >> i >> d >> s;

// String stream for formatting
ostringstream oss;
oss << "Number: " << 123 << ", Value: " << 45.67;
string result = oss.str();

// String stream for conversion
stringstream ss;
ss << 123;
string num_as_string = ss.str();
ss.str("");  // Clear the stream
ss << "456";
int num;
ss >> num;
```

Utility Library

`#include <utility>`

Pair Operations

cpp

```cpp
// Creating pairs
pair<int, string> p1(1, "Hello");
pair<int, string> p2 = make_pair(2, "World");
auto p3 = make_pair(3, "Auto");

// Accessing elements
cout << p1.first << " " << p1.second << endl;

// Comparison (lexicographical)
bool equal = (p1 == p2);
bool less = (p1 < p2);

// Swapping
p1.swap(p2);
swap(p1, p2);
```

Move Semantics

cpp

```cpp
// Move operations
vector<int> v1 = {1, 2, 3, 4, 5};
vector<int> v2 = move(v1);          // v1 is now empty

// Forward (perfect forwarding)
template<typename T>
void wrapper(T&& arg) {
    func(forward<T>(arg));
}
```

Numeric Library

`#include <numeric>`

Accumulation Operations

cpp

```cpp
vector<int> v = {1, 2, 3, 4, 5};

// Basic accumulation
int sum = accumulate(v.begin(), v.end(), 0);
int product = accumulate(v.begin(), v.end(), 1, multiplies<int>());

// Custom operation
int result = accumulate(v.begin(), v.end(), 0,
                       [](int a, int b) { return a + b * b; });

// Inner product
vector<int> v2 = {2, 3, 4, 5, 6};
int dot_product = inner_product(v.begin(), v.end(), v2.begin(), 0);
```

Sequence Generation

cpp

```cpp
vector<int> v(10);

// Fill with incremental values
iota(v.begin(), v.end(), 1);        // Fills with 1, 2, 3, ..., 10

// Partial sums
vector<int> partial_sums(v.size());
partial_sum(v.begin(), v.end(), partial_sums.begin());

// Adjacent differences
vector<int> differences(v.size());
adjacent_difference(v.begin(), v.end(), differences.begin());
```

Iterator Library

`#include <iterator>`

Iterator Types and Operations

cpp

```cpp
vector<int> v = {1, 2, 3, 4, 5};

// Iterator categories
// Input Iterator: Can read, single pass
// Output Iterator: Can write, single pass
// Forward Iterator: Can read/write, multi-pass
// Bidirectional Iterator: Can move backward
// Random Access Iterator: Can jump to any position

// Iterator operations
auto it = v.begin();
advance(it, 3);                          // Move iterator 3 positions
int distance = distance(v.begin(), v.end()); // Distance between iterators
auto it2 = next(it);                     // Get next iterator
auto it3 = prev(it);                     // Get previous iterator
```

Iterator Adaptors

cpp

```cpp
vector<int> v = {1, 2, 3, 4, 5};

// Reverse iterator
for (auto it = v.rbegin(); it != v.rend(); ++it) {
    cout << *it << " ";
}

// Insert iterators
vector<int> dest;
copy(v.begin(), v.end(), back_inserter(dest));     // Insert at back
copy(v.begin(), v.end(), front_inserter(dest));    // Insert at front (deque, list)
copy(v.begin(), v.end(), inserter(dest, dest.begin())); // Insert at position

// Stream iterators
istream_iterator<int> input(cin);
istream_iterator<int> end_input;
ostream_iterator<int> output(cout, " ");

copy(input, end_input, output);     // Copy from cin to cout
```

Functional Library

`#include <functional>`

Function Objects

```cpp
// Arithmetic operations
plus<int> add;
minus<int> subtract;
multiplies<int> multiply;
divides<int> divide;
modulus<int> mod;

int result = add(5, 3);             // 8

// Comparison operations
equal_to<int> eq;
not_equal_to<int> ne;
greater<int> gt;
less<int> lt;
greater_equal<int> ge;
less_equal<int> le;

bool result2 = gt(5, 3);            // true

// Logical operations
logical_and<bool> and_op;
logical_or<bool> or_op;
logical_not<bool> not_op;
```

Function Wrapper

cpp

```
// std::function can hold any callable
function<int(int, int)> func;

// Assign function pointer
func = [](int a, int b) { return a + b; };
int result = func(3, 4);

// Assign member function
struct Calculator {
    int add(int a, int b) { return a + b; }
};

Calculator calc;
function<int(int, int)> member_func = bind(&Calculator::add, calc, placeholders::_1, p
```

Binding

cpp

```
// Bind function arguments
auto add = [](int a, int b, int c) { return a + b + c; };

// Bind some arguments
auto add_5_and_3 = bind(add, 5, 3, placeholders::_1);
int result = add_5_and_3(2);        // 10

// Bind with different order
auto reordered = bind(add, placeholders::_2, placeholders::_1, 10);
int result2 = reordered(5, 3);      // 18 (3 + 5 + 10)
```

Memory Library

`#include <memory>`

Smart Pointers

```cpp
// unique_ptr - exclusive ownership
unique_ptr<int> ptr1 = make_unique<int>(42);
unique_ptr<int> ptr2 = move(ptr1);          // Transfer ownership
int value = *ptr2;
ptr2.reset();                               // Delete and set to nullptr
ptr2.reset(new int(100));                   // Delete old, assign new

// shared_ptr - shared ownership
shared_ptr<int> sptr1 = make_shared<int>(42);
shared_ptr<int> sptr2 = sptr1;              // Shared ownership
cout << sptr1.use_count() << endl;          // Reference count: 2
sptr1.reset();                              // Decrease reference count
// Object deleted when last shared_ptr is destroyed

// weak_ptr - non-owning observer
weak_ptr<int> wptr = sptr2;
if (auto locked = wptr.lock()) {            // Convert to shared_ptr if still valid
    cout << *locked << endl;
}
```

Memory Management

```cpp
// Allocator
allocator<int> alloc;
int* ptr = alloc.allocate(10);              // Allocate space for 10 ints
alloc.construct(ptr, 42);                   // Construct object
alloc.destroy(ptr);                         // Destroy object
alloc.deallocate(ptr, 10);                  // Deallocate space

// Uninitialized memory operations
vector<int> source = {1, 2, 3, 4, 5};
int* raw_memory = static_cast<int*>(malloc(5 * sizeof(int)));

uninitialized_copy(source.begin(), source.end(), raw_memory);
uninitialized_fill_n(raw_memory, 5, 99);
```

Random Library

`#include <random>`

Random Number Generation

cpp

```cpp
// Random number engine
random_device rd;                            // Hardware random number generator
mt19937 gen(rd());                           // Mersenne Twister generator
default_random_engine engine;               // Default engine

// Distributions
uniform_int_distribution<int> int_dist(1, 100);        // Uniform int [1, 100]
uniform_real_distribution<double> real_dist(0.0, 1.0); // Uniform real [0, 1)
normal_distribution<double> normal_dist(0.0, 1.0);     // Normal (mean=0, stddev=1)
binomial_distribution<int> binomial_dist(10, 0.5);     // Binomial
poisson_distribution<int> poisson_dist(4.0);           // Poisson

// Generate random numbers
int random_int = int_dist(gen);
double random_real = real_dist(gen);
double random_normal = normal_dist(gen);

// Shuffle
vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
shuffle(v.begin(), v.end(), gen);

// Random sampling
vector<int> sample_result(3);
sample(v.begin(), v.end(), sample_result.begin(), 3, gen);
```

Quick Reference - Common Patterns

Reading File Line by Line

cpp

```cpp
ifstream file("data.txt");
string line;
while (getline(file, line)) {
    // Process line
}
```

Parsing String with Delimiter

cpp

```cpp
string data = "apple,banana,orange";
stringstream ss(data);
string item;
while (getline(ss, item, ',')) {
    cout << item << endl;
}
```

Finding and Replacing in String

cpp

```cpp
string text = "Hello World Hello";
string old_word = "Hello";
string new_word = "Hi";
size_t pos = 0;
while ((pos = text.find(old_word, pos)) != string::npos) {
    text.replace(pos, old_word.length(), new_word);
    pos += new_word.length();
}
```

Custom Comparator for Sorting

cpp

```cpp
vector<pair<int, string>> v = {{3, "c"}, {1, "a"}, {2, "b"}};
sort(v.begin(), v.end(), [](const auto& a, const auto& b) {
    return a.first < b.first;  // Sort by first element
});
```

Lambda with Capture

cpp

```cpp
int multiplier = 10;
auto lambda = [multiplier](int x) { return x * multiplier; };
```