

C++ STL
Standard library (Functions etc.)

Nathan Warner



**Northern Illinois
University**

Computer Science
Northern Illinois University
United States

Contents

1	C++ Strings (<string>)	2
1.1	Element Access	2
1.2	Capacity	2
1.3	Modifiers	2
1.4	String Operations	3
1.5	Comparison	3
1.6	Conversions	3
2	C-strings (<cstring>)	4
2.1	Manipulation	4
2.2	Examination	4
2.3	Conversion	4
3	Characters (<cctype>)	5
3.1	Character Classification	5
3.2	Character Conversion:	5
4	Other Containers in C++	6
4.1	6
5	Arrays (<array> and <algorithm>)	7
5.1	From <algorithm>	7
5.2	From <array>	8

1 C++ Strings (<string>)

1.1 Element Access

- **at(size_t pos):** Returns a reference to the character at the specified position. Throws `std::out_of_range` if `pos >= size()`.
- **front():** Returns a reference to the first character.
- **back():** Returns a reference to the last character.
- **c_str():** Returns a `const char*` pointing to the null-terminated character array.
- **data():** Returns a `const char*` pointing to the underlying character array.

1.2 Capacity

- **length(), size():** Returns the number of characters in the string (`size_t`).
- **empty():** Returns true if the string is empty, false otherwise (`bool`).
- **resize(size_t n, char c = char()):** Resizes the string to contain `n` characters.
- **capacity():** Returns the size of the storage space currently allocated (`size_t`).
- **reserve(size_t new_cap = 0):** Reserves storage.
- **max_size():** Returns the maximum number of characters the string can hold (`size_t`).
- **shrink_to_fit():** Reduces memory usage by freeing unused memory (no return value).

1.3 Modifiers

- **append(const string& str):-** Required: String to append. (Has other overloads.)
- **push_back(char c):-** Required: Character to append.
- **assign(const string& str, o:start,o:take[n]characters):-** Required: String to assign. (Has other overloads.)
- **insert(size_t pos, const string& str):-** Required: Position and string to insert. (Has other overloads.)
- **replace(size_t pos, size_t len, const string& str):-** Required: Position, length, and string for replacement. (Has other overloads.)
- **swap(string& str):-** Required: String to swap with.

1.4 String Operations

- **substr**(size_t pos = 0, size_t len = npos):- Optional: Starting position and length.
- **copy**(char* s, size_t len, size_t pos = 0) **const**:- Required: Destination and length. Optional: Starting position.
- **find**(const string& str, size_t pos = 0) **const**:- Required: String to find. Optional: Starting position.
- **rfind**(const string& str, size_t pos = npos) **const**:- Required: String to find. Optional: Starting position from the end.

1.5 Comparison

- **compare**(const string& str) **const**:- Required: String to compare with. (Has other overloads.)

1.6 Conversions

- **stoi**: int
- **stol**: long
- **stoul**: unsigned long
- **stoll**: long long
- **stoull**: unsigned long long
- **stof**: float
- **stod**: double
- **stold**: long double

2 C-strings (<cstring>)

2.1 Manipulation

- **strncpy(char dest, const char src, size_t n):** Copy up to n characters from the string src to dest.
- **strncat(char dest, const char src, size_t n):** Append up to n characters from the string src onto the end of dest.

2.2 Examination

- **strlen(const char s):** Return the length of the string (not including the null terminator).
- **strncmp(const char s1, const char s2, size_t n):** Compare up to n characters of two strings.
- **strchr(const char s, int c):** Return a pointer to the first occurrence of character c in the string s, or nullptr if c is not found.
- **strrchr(const char s, int c):** Return a pointer to the last occurrence of character c in the string s.
- **strstr(const char haystack, const char needle):** Return a pointer to the first occurrence of the substring needle in the string haystack, or nullptr if the substring is not found.

2.3 Conversion

- **atoi:** converts C-String to an integer
- **atol:** converts C-String to a long integer
- **atof:** converts C-String to a double

3 Characters (<cctype>)

3.1 Character Classification

- **isalpha(char c):** Checks if the character is an alphabet (either uppercase or lowercase).
- **isdigit(char c):** Checks if the character is a digit (0-9).
- **isalnum(char c):** Checks if the character is either an alphabet or a digit.
- **isspace(char c):** Checks if the character is a whitespace character (like space, tab, newline, etc.).
- **isupper(char c):** Checks if the character is uppercase.
- **islower(char c):** Checks if the character is lowercase.
- **ispunct(char c):** Checks if the character is a punctuation character.
- **isprint(char c):** Checks if the character is printable.
- **isctrl(char c):** Checks if the character is a control character.

3.2 Character Conversion:

- **toupper(char c):** Converts the character to uppercase (if it's lowercase).
- **tolower(char c):** Converts the character to lowercase (if it's uppercase).

4 Other Containers in C++

So far we have discussed things like arrays and vectors, but we have a few other containers that we can use in C++

4.1

5 Arrays (<array> and <algorithm>)

5.1 From <algorithm>

- **sort(first, last)** -> void: Sorts the elements in the range [first, last).
- **find(first, last, value)** -> Iterator: Returns an iterator to the first occurrence of value. If not found, returns last.
- **copy(first, last, destination_first)** -> Iterator: Copies the elements from [first, last) to the beginning at d_first.
- **fill(first, last, value)** -> void: Assigns value to all the elements in the range [first, last).
- **count(first, last, value)** -> size_t: Counts elements that are equal to value.
- **reverse(first, last)** -> void: Reverses the elements in the range [first, last).
- **replace(first, last, old_value, new_value)** -> void: Replaces all elements equal to old_value with new_value.
- **min_element(first, last)** -> Iterator: Returns an iterator pointing to the first instance of the smallest element.
- **max_element(first, last)** -> Iterator: Returns an iterator pointing to the first instance of the largest element.
- **binary_search(first, last, value)** -> bool: Checks if value exists in the sorted sequence.
- **equal(first1, last1, first2)** -> bool: Checks if two sequences are the same.
- **mismatch(first1, last1, first2)** -> Pair: Returns a pair of iterators pointing to the first unequal elements.

Example: Find function

```
#define _SIZE(a) sizeof(a) / sizeof(a[0])

int arr[] = {5,4,3,2,1};
auto it = std::find(arr, _SIZE(arr), 3); // Pointer to that element in our array

if (it != _SIZE(a)) {
    cout << "Found at index pos: " << it - arr;
}
```

If the element is not found, the find function will return a iterator to the last element in the array, so we check to see if this is the case. When we subtract two pointers, we are computed the distance between those two pointers. Hence it-arr gives the position of our element.

5.2 From <array>

- **at(position)** -> T&: Accesses the element at the specified *position* with bounds checking.
- **operator[position]** -> T&: Accesses the element at the specified *position* without bounds checking.
- **front()** -> T&: Returns a reference to the first element.
- **back()** -> T&: Returns a reference to the last element.
- **data()** -> T*: Returns a direct pointer to the underlying array serving as the element storage.
- **begin()** -> Iterator: Returns an iterator pointing to the first element.
- **end()** -> Iterator: Returns an iterator pointing to one-past-the-last element.
- **cbegin()** -> Const_Iterator: Returns a const iterator pointing to the first element.
- **cend()** -> Const_Iterator: Returns a const iterator pointing to one-past-the-last element.
- **rbegin()** -> Reverse_Iterator: Returns a reverse iterator pointing to the last element.
- **rend()** -> Reverse_Iterator: Returns a reverse iterator pointing to one-past-the-first element.
- **crbegin()** -> Const_Reverse_Iterator: Returns a const reverse iterator pointing to the last element.
- **crend()** -> Const_Reverse_Iterator: Returns a const reverse iterator pointing to one-past-the-first element.
- **empty()** -> bool: Checks if the container has no elements.
- **size()** -> size_t: Returns the number of elements in the container.
- **max_size()** -> size_t: Returns the maximum number of elements the container can hold (same as size).
- **fill(value)** -> void: Fills the array with the specified *value*.
- **swap(other)** -> void: Swaps the contents of the array with those of *other*.