C++ STL

Standard library (Functions etc.)

Nathan Warner

Northern Illinois University

Computer Science
Northern Illinois University
United States

Contents

1 C++ Strings (<string>)

Interlude. Before we begin with the C++ string methods, there are two things to know.

- **size_t**: an unsigned integral type, and it's designed to be able to represent the size of any object in bytes
- **npos**: (Constant) It's the largest possible value representable by the size_type of std::string

Note:-

size_t is used as the return value for methods such as ”find” to indicate unsuccessful

1.1 Element Access

- **at(r:size_t pos):**$\mapsto$ char& — Returns a reference to the character at the specified position. Store return value in char&
- **str.front():**$\mapsto$ char& — returns reference to the first character.
- **str.back():**$\mapsto$ char& — returns a reference to the last character.
- **str.c_str():**$\rightarrow$ const char* pointing to the null-terminated character array.
- **str.data():**$\rightarrow$ const char* pointing to the underlying character array.

1.2 Capacity

- **str.length():**$\mapsto$ size_t — Returns the number of characters in the string
- **str.size():**$\mapsto$ size_t — Returns the number of characters in the string
- **str.empty():**$\mapsto$ bool — Returns true if the string is empty, false otherwise
- **resize(r:size_t n, o:char c):**$\mapsto$ void — Resizes the string to contain n characters.
- **capacity():**$\mapsto$ size_t — Returns the size of the storage space currently allocated
- **reserve(size_t new_cap):**$\mapsto$ void — Reserves storage (increases capacity).
- **max_size():**$\mapsto$ size_t — Returns the maximum number of characters the string can hold
- **shrink_to_fit():**$\mapsto$ void — Reduces memory usage by freeing unused memory

1.3 Modifiers

- **append(string str):**$\mapsto$ std::string& mutated_string — String to append. (Has other overloads.) (Doesnt need to be saved in T&)
- **push_back(char c):**$\mapsto$ void — Character to append.
- **assign(string str, o:start,o:stop):**$\mapsto$ std::string& mutated_string — used to replace the current content of the string with a new set of characters. (Has other overloads.)
- **insert(size_t pos, string str):**$\mapsto$ std::string mutated_str — Position and string to insert. (Has other overloads.)
- **replace(size_t pos, size_t len, string str):**$\mapsto$ mutated_string — Position, length, and string for replacement. (Has other overloads.)
- **swap(string str):**$\mapsto$ void — String to swap with.

1.4 String Operations

- **substr(size_t pos = 0, size_t len = npos):** -> string Optional: Starting position and length.
- **copy(char* s, size_t len, size_t pos = 0) const:** -> size_t (number of characters that were copied) Required: Destination and length. Optional: Starting position.
- **find(substr, pos) const:** -> size_t=npos Required: String to find. Optional: Starting position.
- **rfind(substr, pos = npos) const:** -> size_t=npos Required: String to find. Optional: Starting position from the end.
- **find_first_of(substr, pos)** -> size_t=npos Find character in string (public member function)
- **find_last_of(substr, pos)** -> size_t=npos Find character in string from the end (public member function)
- **find_first_not_of(substr, pos)** -> size_t=npos Find absence of character in string (public member function)
- **find_last_not_of(substr, pos)** -> size_t=npos Find non-matching character in string from the end (public member function)

1.5 Comparison

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

1.6 Conversions

- **c_str**: converts to c-string
- **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.
- **iscntrl(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*.