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 (These are functions) . . . 3

2 C-strings (<cstring>) . . . 4
- 2.1 Manipulation . . . 4
- 2.2 Examination . . . 4
- 2.3 Searching . . . 4
- 2.4 Error . . . 5
- 2.5 Conversion . . . 7

3 Characters (<cctype>) . . . 8
- 3.1 Character Classification . . . 8
- 3.2 Character Conversion: . . . 8

4 Other Containers in C++ . . . 9
- 4.1 . . . 9

5 Arrays (<array> and <algorithm>) . . . 10
- 5.1 From <algorithm> . . . 10
- 5.2 From <array> . . . 11

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

1.2 Capacity

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

1.3 Modifiers

- **append(string str):**↦ std::string& mutated_string — String to append. (Has other overloads.) (Doesnt need to be saved in T&)
- **push_back(char c):**↦ void — Character to append.
- **assign(string str, o:start,o:stop):**↦ 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):**↦ std::string mutated_str — Position and string to insert. (Has other overloads.)
- **replace(size_t pos, size_t len, string str):**↦ mutated_string — Position, length, and string for replacement. (Has other overloads.)
- **swap(string str):**↦ void — String to swap with.
- **pop_back():** ↦ void — Removes the last character in the string

1.4 String Operations

- **substr(pos, o:len):**↦ string — Generate a substr from a string
- **copy(char[] dest, o:len, pos):**↦ size_t (number of characters that were copied) send contents of string to some character array
- **find(substr, o:pos):**↦ size_t=npos find a substr from within a string
- **rfind(substr, o:pos):**↦ size_t=npos find a substr form withing as string, starting search from the end of the string
- **find_first_of(substr (or char), o:pos)**↦ size_t=npos Find character in string (public member function)
- **find_last_of(substr (or char), o:pos)**↦ size_t=npos Find character in string from the end (public member function)
- **find_first_not_of(substr (or char), o:pos)**↦ size_t=npos Find absence of character in string (public member function)
- **find_last_not_of(substr (or char), o:pos)**↦ size_t=npos Find non-matching character in string from the end (public member function)

1.5 Comparison

- **compare(o:pos, o:len, str):**↦ unsigned integral
  - 0: they compare equal
  - <0: Either the value of the first character that does not match is lower in the compared string, or all compared characters match but the compared string is shorter.
  - >0: Either the value of the first character that does not match is greater in the compared string, or all compared characters match but the compared string is longer.

1.6 Conversions (These are functions)

- **stoi(str, o:idx, o:base):**↦ int (idx is a pointer to a size_t object)
- **stol(str, o:idx, o:base):**↦ long
- **stoul(str, o:idx, o:base):**↦ unsigned long
- **stoll(str, o:idx, o:base):**↦ long long
- **stoull(str, o:idx, o:base):**↦ unsigned long long
- **stof(str, o:idx):**↦ float
- **stod(str, o:idx):**↦ double
- **stold(str, o:idx):**↦ long double

2 C-strings (<cstring>)

2.1 Manipulation

- **strncpy(char[] dest, char[] src, size_t n):**↦ const char* — Copy up to n characters from the string src to dest.
- **strcpy(char[] dest, char[] src)**↦ const char* —- Copies the C string pointed by source into the array pointed by destination
- **strncat(char[] dest, char[] src, size_t n):**↦ const char* — Append up to n characters from the string src onto the end of dest.
- **strcat(char[] dest, char[] src)**↦ const char* — Appends a copy of the source string to the destination string.

  Note: these return a const char* comprised of the characters from the src array that were used.

2.2 Examination

- **strnlen(char[] src, size_t maxlen)**↦ size_t — Return the length of the string (not including the null terminator).
- **strlen(char[] src):**↦ size_t — Return the length of the string (not including the null terminator).
- **strncmp(char[] src1, char[] src2, size_t n):**↦ uint (value varies) — Compare up to n characters of two strings.
- **strcmp(char[] src1, char[] src2)**↦ size_t (value varires) — compare two strings
- **strchr(char[] src, char c):**↦ char* —- get pointer to the first occurrence of character c in the string s, or nullptr if c is not found.
- **strrchr(char[] src, char c):**↦ char* — get pointer to the last occurrence of character c in the string s.
- **strstr(char[] src, char[] str):**↦ const char* — Return a pointer to the first occurrence of the substring

2.3 Searching

- **strchr(char[] src, char c):**↦ char* —- get pointer to the first occurrence of character c in the string s, or nullptr if c is not found.
- **strrchr(char[] src, char c):**↦ char* — get pointer to the last occurrence of character c in the string s.
- **strstr(char[] src, char[] str):**↦ const char* — Return a pointer to the first occurrence of the substring
- **strcspn(char[] src, char[] charset):**↦ size_t — Get position of first character found from charset
- **strpbrk(char[] src, char[] charset):**↦ const char* — Return string consisting of first match form character set in string onward
- **strspn(char[] src, char[] charset):**↦ size_t — Returns the length of the initial portion of str1 which consists only of characters that are part of str2.

2.4 Error

- **strerror(errno)**↦ const char* — Get pointer to error message string (we are literally passing in the defined variable *errno*)

```
std::ifstream inf("file.txt");
if (inf.fail()) { cout << strerrror(errno); } // No such file in directory
(Because that is what the error string is set to after inf.fail())
```

we also have control over the error we can pass in, errors that are defined in <cerrno>. Rather than leaving it up to errno. We have:

E2BIG (C++11) Argument list too long (macro constant)
EACCES (C++11) Permission denied (macro constant)
EADDRINUSE (C++11) Address in use (macro constant)
EADDRNOTAVAIL (C++11) Address not available (macro constant)
EAFNOSUPPORT (C++11) Address family not supported (macro constant)
EAGAIN (C++11) Resource unavailable, try again (macro constant)
EALREADY (C++11) Connection already in progress (macro constant)
EBADF (C++11) Bad file descriptor (macro constant)
EBADMSG (C++11) Bad message (macro constant)
EBUSY (C++11) Device or resource busy (macro constant)
ECANCELED (C++11) Operation canceled (macro constant)
ECHILD (C++11) No child processes (macro constant)
ECONNABORTED (C++11) Connection aborted (macro constant)
ECONNREFUSED (C++11) Connection refused (macro constant)
ECONNRESET (C++11) Connection reset (macro constant)
EDEADLK (C++11) Resource deadlock would occur (macro constant)
EDESTADDRREQ (C++11) Destination address required (macro constant)
EDOM Mathematics argument out of domain of function (macro constant)
EEXIST (C++11) File exists (macro constant)
EFAULT (C++11) Bad address (macro constant)
EFBIG (C++11) File too large (macro constant)
EHOSTUNREACH (C++11) Host is unreachable (macro constant)
EIDRM (C++11) Identifier removed (macro constant)
EILSEQ Illegal byte sequence (macro constant)
EINPROGRESS (C++11) Operation in progress (macro constant)
EINTR (C++11) Interrupted function (macro constant)
EINVAL (C++11) Invalid argument (macro constant)
EIO (C++11) I/O error (macro constant)
EISCONN (C++11) Socket is connected (macro constant)
EISDIR (C++11) Is a directory (macro constant)
ELOOP (C++11) Too many levels of symbolic links (macro constant)
EMFILE (C++11) File descriptor value too large (macro constant)
EMLINK (C++11) Too many links (macro constant)
EMSGSIZE (C++11) Message too large (macro constant)

ENAMETOOLONG (C++11) Filename too long (macro constant)
ENETDOWN (C++11) Network is down (macro constant)
ENETRESET (C++11) Connection aborted by network (macro constant)
ENETUNREACH (C++11) Network unreachable (macro constant)
ENFILE (C++11) Too many files open in system (macro constant)
ENOBUFS (C++11) No buffer space available (macro constant)
ENODATA (C++11)(deprecated in C++23) No message is available on the STREAM head read queue (macro constant)
ENODEV (C++11) No such device (macro constant)
ENOENT (C++11) No such file or directory (macro constant)
ENOEXEC (C++11) Executable file format error (macro constant)
ENOLCK (C++11) No locks available (macro constant)
ENOLINK (C++11) Link has been severed (macro constant)
ENOMEM (C++11) Not enough space (macro constant)
ENOMSG (C++11) No message of the desired type (macro constant)
ENOPROTOOPT (C++11) Protocol not available (macro constant)
ENOSPC (C++11) No space left on device (macro constant)
ENOSR (C++11)(deprecated in C++23) No STREAM resources (macro constant)
ENOSTR (C++11)(deprecated in C++23) Not a STREAM (macro constant)
ENOSYS (C++11) Function not supported (macro constant)
ENOTCONN (C++11) The socket is not connected (macro constant)
ENOTDIR (C++11) Not a directory (macro constant)
ENOTEMPTY (C++11) Directory not empty (macro constant)
ENOTRECOVERABLE (C++11) State not recoverable (macro constant)
ENOTSOCK (C++11) Not a socket (macro constant)
ENOTSUP (C++11) Not supported (macro constant)
ENOTTY (C++11) Inappropriate I/O control operation (macro constant)
ENXIO (C++11) No such device or address (macro constant)
EOPNOTSUPP (C++11) Operation not supported on socket (macro constant)
EOVERFLOW (C++11) Value too large to be stored in data type (macro constant)
EOWNERDEAD (C++11) Previous owner died (macro constant)
EPERM (C++11) Operation not permitted (macro constant)
EPIPE (C++11) Broken pipe (macro constant)
EPROTO (C++11) Protocol error (macro constant)
EPROTONOSUPPORT (C++11) Protocol not supported (macro constant)
EPROTOTYPE (C++11) Protocol wrong type for socket (macro constant)
ERANGE Result too large (macro constant)
EROFS (C++11) Read-only file system (macro constant)
ESPIPE (C++11) Invalid seek (macro constant)
ESRCH (C++11) No such process (macro constant)
ETIME (C++11)(deprecated in C++23) Stream ioctl() timeout (macro constant)
ETIMEDOUT (C++11) Connection timed out (macro constant)
ETXTBSY (C++11) Text file busy (macro constant)
EWOULDBLOCK (C++11) Operation would block (macro constant)
EXDEV (C++11) Cross-device link (macro constant)

2.5 Conversion

- **atoi(char[])** - Converts a C-string to an int.
- **atol(char[])** - Converts a C-string to a long.
- **atoll(char[])** - Converts a C-string to a long long.
- **atof(char[])** - Converts a C-string to a double.
- **strtol(char[], entdptr, base)** - Converts a C-string to a long int, with error checking and more flexibility with base representations.
- **strtoul(char[], entdptr, base)** - Converts a C-string to an unsigned long int.
- **strtoll(char[], entdptr, base)** - Converts a C-string to a long long int.
- **strtoull(char[], entdptr, base)** - Converts a C-string to an unsigned long long int.
- **strtof(char[], entdptr)** - Converts a C-string to a float.
- **strtod(char[], entdptr)** - Converts a C-string to a double.
- **strtold(char[], entdptr)** - Converts a C-string to a long double.

Note: Consider the codeblock

```
const char* a = "12.322string";
char* ptr;

double val = strtod(a, &ptr); // ptr will hold the value "string"
```

In this example, you can see that we are making use of the endptr optional parameter, *endptr* is the address to a char pointer. It allows us to send all the string data that is not able to be converted to the requested data type . It allows us to send all the string data that is not able to be converted to the requested data type to this pointer. Only data AFTER the converted data will be sent.

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*.