Strings library

The C++ strings library includes support for two general types of strings:

- `std::basic_string` - a templated class designed to manipulate strings of any character type.
- Null-terminated strings - arrays of characters terminated by a special *null* character.

`std::basic_string`

The templated class `std::basic_string` generalizes how sequences of characters are manipulated and stored. String creation, manipulation, and destruction are all handled by a convenient set of class methods and related functions.

Several specializations of `std::basic_string` are provided for commonly-used types:

Defined in header `<string>`

Type	Definition
`std::string`	`std::basic_string<char>`
`std::wstring`	`std::basic_string<wchar_t>`
`std::u16string`	`std::basic_string<char16_t>`
`std::u32string`	`std::basic_string<char32_t>`

Null-terminated strings

Null-terminated strings are arrays of characters that are terminated by a special *null* character. C++ provides functions to create, inspect, and modify null-terminated strings.

There are three types of null-terminated strings:

- null-terminated byte strings
- null-terminated multibyte strings
- null-terminated wide strings

Additional support

`std::char_traits`

The string library also provides class template `std::char_traits` that defines types and functions for `std::basic_string`. The following specializations are defined:

Defined in header `<string>`

```
template<> class char_traits<char>;
template<> class char_traits<wchar_t>;
template<> class char_traits<char16_t>;    (since C++11)
template<> class char_traits<char32_t>;    (since C++11)
```

See also

C documentation for **`Strings library`**

Null-terminated byte strings

A null-terminated byte string (NTBS) is a sequence of nonzero bytes followed by a byte with value zero (the terminating null character). Each byte in a byte string encodes one character of some character set. For example, the character array `{'\x63', '\x61', '\x74', '\0'}` is an NTBS holding the string `"cat"` in ASCII encoding.

Functions

Character classification

Defined in header `<cctype>`

isalnum	checks if a character is alphanumeric (function)
isalpha	checks if a character is alphabetic (function)
islower	checks if a character is lowercase (function)
isupper	checks if a character is an uppercase character (function)
isdigit	checks if a character is a digit (function)
isxdigit	checks if a character is a hexadecimal character (function)
iscntrl	checks if a character is a control character (function)
isgraph	checks if a character is a graphical character (function)
isspace	checks if a character is a space character (function)
isblank (C++11)	checks if a character is a blank character (function)
isprint	checks if a character is a printing character (function)
ispunct	checks if a character is a punctuation character (function)

Character manipulation

tolower	converts a character to lowercase (function)
toupper	converts a character to uppercase (function)

ASCII values (hex)		characters	iscntrl iswcntrl	isprint iswprint	isspace iswspace	isblank iswblank	isgraph iswgraph	ispunct iswpunct	isalnum iswalnum	isalpha iswalpha	isupper iswupper	islower iswlower	isdigit iswdigit	isxdigit iswxdigit
0 - 8	0x00–0x08	control codes (NUL, etc.)	≠0	0	0	0	0	0	0	0	0	0	0	0
9	0x09	tab (\t)	≠0	0	≠0	≠0	0	0	0	0	0	0	0	0
10 - 13	0x0A–0x0D	whitespaces (\n,\v,\f,\r)	≠0	0	≠0	0	0	0	0	0	0	0	0	0
14 - 31	0x0E–0x1F	control codes	≠0	0	0	0	0	0	0	0	0	0	0	0
32	0x20	space	0	≠0	≠0	≠0	0	0	0	0	0	0	0	0
33 - 47	0x21–0x2F	!"#$%&'()*+,-./	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
48 - 57	0x30–0x39	0123456789	0	≠0	0	0	≠0	0	≠0	0	0	0	≠0	≠0
58 - 64	0x3a–0x40	:;<=>?@	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
65 - 70	0x41–0x46	ABCDEF	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	≠0
71 - 90	0x47–0x5A	GHIJKLMNOPQRSTUVWXYZ	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	0
91 - 96	0x5B–0x60	[\]^_`	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
97 -102	0x61–0x66	abcdef	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	≠0
103-122	0x67–0x7A	ghijklmnopqrstuvwxyz	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	0
123-126	0x7B–0x7E	{|}~	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
127	0x7F	backspace character (DEL)	≠0	0	0	0	0	0	0	0	0	0	0	0

Conversions to numeric formats

Defined in header <cstdlib>

atof	converts a byte string to a floating point value (function)
atoi **atol** **atoll**	converts a byte string to an integer value (function)
strtol **strtoll**	converts a byte string to an integer value (function)
strtoul **strtoull**	converts a byte string to an unsigned integer value (function)
strtof **strtod** **strtold**	converts a byte string to a floating point value (function)

Defined in header <cinttypes>

strtoimax (C++11) **strtoumax** (C++11)	converts a byte string to std::intmax_t or std::uintmax_t (function)

String manipulation

Defined in header <cstring>

strcpy	copies one string to another (function)
strncpy	copies a certain amount of characters from one string to another (function)
strcat	concatenates two strings (function)
strncat	concatenates a certain amount of characters of two strings (function)
strxfrm	transform a string so that strcmp would produce the same result as strcoll (function)

String examination

Defined in header <cstring>

strlen	returns the length of a given string (function)
strcmp	compares two strings

	(function)
strncmp	compares a certain amount of characters of two strings (function)
strcoll	compares two strings in accordance to the current locale (function)
strchr	finds the first occurrence of a character (function)
strrchr	finds the last occurrence of a character (function)
strspn	returns the length of the maximum initial segment that consists of only the characters found in another byte string (function)
strcspn	returns the length of the maximum initial segment that consists of only the characters not found in another byte string (function)
strpbrk	finds the first location of any character from a set of separators (function)
strstr	finds the first occurrence of a substring of characters (function)
strtok	finds the next token in a byte string (function)

Character array manipulation

Defined in header `<cstring>`

memchr	searches an array for the first occurrence of a character (function)
memcmp	compares two buffers (function)
memset	fills a buffer with a character (function)
memcpy	copies one buffer to another (function)
memmove	moves one buffer to another (function)

Miscellaneous

Defined in header `<cstring>`

strerror	returns a text version of a given error code (function)

See also

C documentation for **Null-terminated byte strings**

Null-terminated multibyte strings

A null-terminated multibyte string (NTMBS), or "multibyte string", is a sequence of nonzero bytes followed by a byte with value zero (the terminating null character).

Each character stored in the string may occupy more than one byte. The encoding used to represent characters in a multibyte character string is locale-specific: it may be UTF-8, GB18030, EUC-JP, Shift-JIS, etc. For example, the char array `{'\xe4','\xbd','\xa0','\xe5','\xa5','\xbd','\0'}` is an NTMBS holding the string `"你好"` in UTF-8 multibyte encoding: the first three bytes encode the character 你, the next three bytes encode the character 好. The same string encoded in GB18030 is the char array `{'\xc4', '\xe3', '\xba', '\xc3', '\0'}`, where each of the two characters is encoded as a two-byte sequence.

In some multibyte encodings, any given multibyte character sequence may represent different characters depending on the previous byte sequences, known as "shift sequences". Such encodings are known as state-dependent: knowledge of the current shift state is required to interpret each character. An NTMBS is only valid if it begins and ends in the initial shift state: if a shift sequence was used, the corresponding unshift sequence has to be present before the terminating null character. Examples of such encodings are the 7-bit JIS, BOCU-1 and SCSU.

A multibyte character string is layout-compatible with null-terminated byte string (NTBS), that is, can be stored, copied, and examined using the same facilities, except for calculating the number of characters. If the correct locale is in effect, I/O functions also handle multibyte strings. Multibyte strings can be converted to and from wide strings using the `std::codecvt` member functions, `std::wstring_convert`, or the following locale-dependent conversion functions:

Multibyte/wide character conversions

Defined in header `<cstdlib>`	
`mblen`	returns the number of bytes in the next multibyte character (function)
`mbtowc`	converts the next multibyte character to wide character (function)
`wctomb`	converts a wide character to its multibyte representation (function)
`mbstowcs`	converts a narrow multibyte character string to wide string (function)
`wcstombs`	converts a wide string to narrow multibyte character string (function)
Defined in header `<cwchar>`	
`mbsinit`	checks if the mbstate_t object represents initial shift state (function)
`btowc`	widens a single-byte narrow character to wide character, if possible (function)
`wctob`	narrows a wide character to a single-byte narrow character, if possible (function)
`mbrlen`	returns the number of bytes in the next multibyte character, given state (function)
`mbrtowc`	converts the next multibyte character to wide character, given state (function)
`wcrtomb`	converts a wide character to its multibyte representation, given state (function)
`mbsrtowcs`	converts a narrow multibyte character string to wide string, given state (function)
`wcsrtombs`	converts a wide string to narrow multibyte character string, given state (function)
Defined in header `<cuchar>`	
`mbrtoc16` (C++11)	generate the next 16-bit wide character from a narrow multibyte string (function)
`c16rtomb` (C++11)	convert a 16-bit wide character to narrow multibyte string (function)
`mbrtoc32` (C++11)	generate the next 32-bit wide character from a narrow multibyte string (function)
`c32rtomb` (C++11)	convert a 32-bit wide character to narrow multibyte string (function)

Types

Defined in header `<cwchar>`	
`mbstate_t`	conversion state information necessary to iterate multibyte character strings (class)

Macros

Defined in header <climits>	
MB_LEN_MAX	maximum number of bytes in a multibyte character (macro constant)
Defined in header <cstdlib>	
MB_CUR_MAX	maximum number of bytes in a multibyte character in the current C locale (macro variable)
Defined in header <cuchar>	
__STDC_UTF_16__	indicates that UTF-16 encoding is used by mbrtoc16 and c16rtomb (macro constant)
__STDC_UTF_32__	indicates that UTF-32 encoding is used by mbrtoc32 and c32rtomb (macro constant)

See also

C documentation for **Null-terminated multibyte strings**

Null-terminated wide strings

A null-terminated wide string is a sequence of valid wide characters, ending with a null character.

Functions

Character classification

Defined in header <cwctype>

iswalnum	checks if a wide character is alphanumeric (function)
iswalpha	checks if a wide character is alphabetic (function)
iswlower	checks if a wide character is lowercase (function)
iswupper	checks if a wide character is an uppercase character (function)
iswdigit	checks if a wide character is a digit (function)
iswxdigit	checks if a character is a hexadecimal character (function)
iswcntrl	checks if a wide character is a control character (function)
iswgraph	checks if a wide character is a graphical character (function)
iswspace	checks if a wide character is a space character (function)
iswblank (C++11)	checks if a wide character is a blank character (function)
iswprint	checks if a wide character is a printing character (function)
iswpunct	checks if a wide character is a punctuation character (function)
iswctype	classifies a wide character according to the specified LC_CTYPE category (function)
wctype	looks up a character classification category in the current C locale (function)

Character manipulation

Defined in header <cwctype>

towlower	converts a wide character to lowercase (function)
towupper	converts a wide character to uppercase (function)
towctrans	performs character mapping according to the specified LC_CTYPE mapping category (function)
wctrans	looks up a character mapping category in the current C locale (function)

ASCII values (hex)		characters	iscntrl iswcntrl	isprint iswprint	isspace iswspace	isblank iswblank	isgraph iswgraph	ispunct iswpunct	isalnum iswalnum	isalpha iswalpha	isupper iswupper	islower iswlower	isdigit iswdigit	isxdigit iswxdigit
0 - 8	0x00–0x08	control codes (NUL, etc.)	≠0	0	0	0	0	0	0	0	0	0	0	0
9	0x09	tab (\t)	≠0	0	≠0	≠0	0	0	0	0	0	0	0	0
10 - 13	0x0A–0x0D	whitespaces (\n,\v,\f,\r)	≠0	0	≠0	0	0	0	0	0	0	0	0	0
14 - 31	0x0E–0x1F	control codes	≠0	0	0	0	0	0	0	0	0	0	0	0
32	0x20	space	0	≠0	≠0	≠0	0	0	0	0	0	0	0	0
33 - 47	0x21–0x2F	!"#$%&'()*+,-./	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
48 - 57	0x30–0x39	0123456789	0	≠0	0	0	≠0	0	≠0	0	0	0	≠0	≠0
58 - 64	0x3a–0x40	:;<=>?@	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
65 - 70	0x41–0x46	ABCDEF	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	≠0
71 - 90	0x47–0x5A	GHIJKLMNOPQRSTUVWXYZ	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	0
91 - 96	0x5B–0x60	[\]^_`	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
97 -102	0x61–0x66	abcdef	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	≠0
103-122	0x67–0x7A	ghijklmnopqrstuvwxyz	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	0
123-126	0x7B–0x7E	{|}~	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
127	0x7F	backspace character (DEL)	≠0	0	0	0	0	0	0	0	0	0	0	0

Conversions to numeric formats

Defined in header <cwchar>

wcstol **wcstoll**	converts a wide string to an integer value (function)
wcstoul **wcstoull**	converts a wide string to an unsigned integer value (function)
wcstof **wcstod** **wcstold**	converts a wide string to a floating point value (function)

Defined in header <cinttypes>

wcstoimax (C++11)	converts a wide string to std::intmax_t or std::uintmax_t (function)
wcstoumax (C++11)	

String manipulation

Defined in header <cwchar>

wcscpy	copies one wide string to another (function)
wcsncpy	copies a certain amount of wide characters from one string to another (function)
wcscat	appends a copy of one wide string to another (function)
wcsncat	appends a certain amount of wide characters from one wide string to another (function)
wcsxfrm	transform a wide string so that wcscmp would produce the same result as wcscoll (function)

String examination

Defined in header <cwchar>

wcslen	returns the length of a wide string (function)
wcscmp	compares two wide strings (function)
wcsncmp	compares a certain amount of characters from two wide strings (function)
wcscoll	compares two wide strings in accordance to the current locale (function)

wcschr	finds the first occurrence of a wide character in a wide string (function)
wcsrchr	finds the last occurrence of a wide character in a wide string (function)
wcsspn	returns the length of the maximum initial segment that consists of only the wide characters found in another wide string (function)
wcscspn	returns the length of the maximum initial segment that consists of only the wide *not* found in another wide string (function)
wcspbrk	finds the first location of any wide character in one wide string, in another wide string (function)
wcsstr	finds the first occurrence of a wide string within another wide string (function)
wcstok	finds the next token in a wide string (function)

Wide character array manipulation

Defined in header <cwchar>

wmemcpy	copies a certain amount of wide characters between two non-overlapping arrays (function)
wmemmove	copies a certain amount of wide characters between two, possibly overlapping, arrays (function)
wmemcmp	compares a certain amount of wide characters from two arrays (function)
wmemchr	finds the first occurrence of a wide character in a wide character array (function)
wmemset	copies the given wide character to every position in a wide character array (function)

Types

Defined in header <cwctype>

wctrans_t	scalar type that holds locale-specific character mapping
wctype_t	scalar type that holds locale-specific character classification
wint_t	integer type that can hold any valid wide character and at least one more value

Macros

Defined in header <cwchar>

WEOF	a non-character value of type wint_t used to indicate errors (macro constant)
WCHAR_MIN	the smallest valid value of wchar_t (macro constant)
WCHAR_MAX	the largest valid value of wchar_t (macro constant)

See also

C documentation for **Null-terminated wide strings**	

std::basic_string

Defined in header <string>

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>,
    class Allocator = std::allocator<CharT>
> class basic_string;
```

The class template `basic_string` stores and manipulates sequences of `char`-like objects. The class is dependent neither on the character type nor on the nature of operations on that type. The definitions of the operations are supplied via the `Traits` template parameter - a specialization of `std::char_traits` or a compatible traits class.

The elements of a `basic_string` are stored contiguously, that is, for a basic_string `s`, `&*(s.begin() + n) == &*s.begin() + n` for any n in `[0, s.size())`, or, equivalently, a pointer to `s[0]` can be passed to functions that expect a pointer to the first element of a `CharT[]` array. (since C++11)

`std::basic_string` satisfies the requirements of `AllocatorAwareContainer`, `SequenceContainer` and `ContiguousContainer` (since C++17)

Several typedefs for common character types are provided:

Defined in header <string>

Type	Definition
std::string	`std::basic_string<char>`
std::wstring	`std::basic_string<wchar_t>`
std::u16string (C++11)	`std::basic_string<char16_t>`
std::u32string (C++11)	`std::basic_string<char32_t>`

Template parameters

CharT	-	character type
Traits	-	traits class specifying the operations on the character type
Allocator	-	`Allocator` type used to allocate internal storage

Member types

Member type	Definition
`traits_type`	`Traits`
`value_type`	`Traits::char_type`
`allocator_type`	`Allocator`
`size_type`	`Allocator::size_type` (until C++11) `std::allocator_traits<Allocator>::size_type` (since C++11)
`difference_type`	`Allocator::difference_type` (until C++11) `std::allocator_traits<Allocator>::difference_type` (since C++11)
`reference`	`Allocator::reference` (until C++11) `value_type&` (since C++11)
`const_reference`	`Allocator::const_reference` (until C++11) `const value_type&` (since C++11)
`pointer`	`Allocator::pointer` (until C++11) `std::allocator_traits<Allocator>::pointer` (since C++11)
`const_pointer`	`Allocator::const_pointer` (until C++11) `std::allocator_traits<Allocator>::const_pointer` (since C++11)
`iterator`	`RandomAccessIterator`
`const_iterator`	Constant random access iterator
`reverse_iterator`	`std::reverse_iterator<iterator>`
`const_reverse_iterator`	`std::reverse_iterator<const_iterator>`

Member functions

(constructor)	constructs a `basic_string` (public member function)

operator=	assigns values to the string (public member function)
assign	assign characters to a string (public member function)
get_allocator	returns the associated allocator (public member function)

Element access

at	access specified character with bounds checking (public member function)
operator[]	access specified character (public member function)
front (C++11)	accesses the first character (public member function)
back (C++11)	accesses the last character (public member function)
data	returns a pointer to the first character of a string (public member function)
c_str	returns a non-modifiable standard C character array version of the string (public member function)

Iterators

begin **cbegin** (C++11)	returns an iterator to the beginning (public member function)
end **cend** (C++11)	returns an iterator to the end (public member function)
rbegin **crbegin** (C++11)	returns a reverse iterator to the beginning (public member function)
rend **crend** (C++11)	returns a reverse iterator to the end (public member function)

Capacity

empty	checks whether the string is empty (public member function)
size **length**	returns the number of characters (public member function)
max_size	returns the maximum number of characters (public member function)
reserve	reserves storage (public member function)
capacity	returns the number of characters that can be held in currently allocated storage (public member function)
shrink_to_fit (C++11)	reduces memory usage by freeing unused memory (public member function)

Operations

clear	clears the contents (public member function)
insert	inserts characters (public member function)
erase	removes characters (public member function)
push_back	appends a character to the end (public member function)
pop_back (C++11)	removes the last character (public member function)
append	appends characters to the end (public member function)
operator+=	appends characters to the end (public member function)
compare	compares two strings (public member function)
replace	replaces specified portion of a string (public member function)
substr	returns a substring (public member function)
copy	copies characters (public member function)

resize	changes the number of characters stored (public member function)
swap	swaps the contents (public member function)

Search

find	find characters in the string (public member function)
rfind	find the last occurrence of a substring (public member function)
find_first_of	find first occurrence of characters (public member function)
find_first_not_of	find first absence of characters (public member function)
find_last_of	find last occurrence of characters (public member function)
find_last_not_of	find last absence of characters (public member function)

Constants

npos [static]	special value. The exact meaning depends on the context (public static member constant)

Non-member functions

operator+	concatenates two strings or a string and a char (function template)
operator== **operator!=** **operator<** **operator>** **operator<=** **operator>=**	lexicographically compares two strings (function template)
std::swap(std::basic_string)	specializes the `std::swap` algorithm (function template)

Input/output

operator<< **operator>>**	performs stream input and output on strings (function template)
getline	read data from an I/O stream into a string (function template)

Numeric conversions

stoi (C++11) **stol** (C++11) **stoll** (C++11)	converts a string to a signed integer (function)
stoul (C++11) **stoull** (C++11)	converts a string to an unsigned integer (function)
stof (C++11) **stod** (C++11) **stold** (C++11)	converts a string to a floating point value (function)
to_string (C++11)	converts an integral or floating point value to `string` (function)
to_wstring (C++11)	converts an integral or floating point value to `wstring` (function)

Literals

Defined in inline namespace `std::literals::string_literals`

operator""s (C++14)	Converts a character array literal to `basic_string` (function)

Helper classes

std::hash<std::string>	(C++11)	hash support for strings (class template specialization)
std::hash<std::u16string>	(C++11)	
std::hash<std::u32string>	(C++11)	
std::hash<std::wstring>	(C++11)	

std::char_traits

Defined in header <string>

```
template<
    class CharT
> class char_traits;
```

The char_traits class is a traits class template that abstracts basic character and string operations for a given character type. The defined operation set is such that generic algorithms almost always can be implemented in terms of it. It is thus possible to use such algorithms with almost any possible character or string type, just by supplying customized char_traits class.

The char_traits class template serves as a basis for explicit instantiations. The user can provide a specialization for any custom character types. Several specializations are defined for the standard character types. The values of the member typedefs are as follows.

Instantiation	char_type	int_type	off_type	pos_type	state_type
char_traits<char>	char	int	std::streamoff	std::streampos	std::mbstate_t
char_traits<wchar_t>	wchar_t	std::wint_t	std::wstreamoff	std::wstreampos	std::mbstate_t
char_traits<char16_t> (C++11)	char16_t	std::uint_least16_t	std::streamoff	std::u16streampos	std::mbstate_t
char_traits<char32_t> (C++11)	char32_t	std::uint_least32_t	std::streamoff	std::u32streampos	std::mbstate_t

The char_traits class template satisfies the requirements of CharTraits.

This section is incomplete
Reason: simplify the description, emphasize that char_traits can be user defined

Member types

Type	Definition
char_type	CharT
int_type	an integer type that can hold all values of char_type plus EOF
off_type	*implementation-defined*
pos_type	*implementation-defined*
state_type	*implementation-defined*

Member functions

assign [static]	assigns a character (public static member function)
eq **lt** [static]	compares two characters (public static member function)
move [static]	moves one character sequence onto another (public static member function)
copy [static]	copies a character sequence (public static member function)
compare [static]	lexicographically compares two character sequences (public static member function)
length [static]	returns the length of a character sequence (public static member function)
find [static]	finds a character in a character sequence (public static member function)
to_char_type [static]	converts int_type to equivalent char_type (public static member function)
to_int_type [static]	converts char_type to equivalent int_type (public static member function)
eq_int_type [static]	compares two int_type values (public static member function)
eof [static]	returns an *eof* value (public static member function)
not_eof [static]	checks whether a character is *eof* value (public static member function)

Example

User-defined character traits may be used to provide case-insensitive comparison (http%3A//www.gotw.ca/gotw/029.htm)

Run this code

```
#include <string>
#include <iostream>
#include <cctype>

struct ci_char_traits : public std::char_traits<char> {
    static bool eq(char c1, char c2) {
         return std::toupper(c1) == std::toupper(c2);
     }
    static bool lt(char c1, char c2) {
         return std::toupper(c1) <  std::toupper(c2);
    }
    static int compare(const char* s1, const char* s2, size_t n) {
        while ( n-- != 0 ) {
            if ( std::toupper(*s1) < std::toupper(*s2) ) return -1;
            if ( std::toupper(*s1) > std::toupper(*s2) ) return 1;
            ++s1; ++s2;
        }
        return 0;
    }
    static const char* find(const char* s, int n, char a) {
        auto const ua (std::toupper(a));
        while ( n-- != 0 )
        {
            if (std::toupper(*s) == ua)
                return s;
            s++;
        }
        return nullptr;
    }
};

typedef std::basic_string<char, ci_char_traits> ci_string;

std::ostream& operator<<(std::ostream& os, const ci_string& str) {
    return os.write(str.data(), str.size());
}

int main()
{
    ci_string s1 = "Hello";
    ci_string s2 = "heLLo";
    if (s1 == s2)
        std::cout << s1 << " and " << s2 << " are equal\n";
}
```

Output:

```
Hello and heLLo are equal
```

See also

basic_string	stores and manipulates sequences of characters (class template)