basic_string<charT, traits, Alloc>

Containers

Category: containers

Type

Component type: type

Description

The `basic_string` class represents a Sequence of characters. It contains all the usual operations of a Sequence, and, additionally, it contains standard string operations such as search and concatenation.

The `basic_string` class is parameterized by character type, and by that type's Character Traits. Most of the time, however, there is no need to use the `basic_string` template directly. The types `string` and `wstring` are typedefs for, respectively, `basic_string<char>` and `basic_string<wchar_t>`.

Some of `basic_string`'s member functions use an unusual method of specifying positions and ranges. In addition to the conventional method using iterators, many of `basic_string`'s member functions use a single value `pos` of type `size_type` to represent a position (in which case the position is `begin() + pos`, and many of `basic_string`'s member functions use two values, `pos` and `n`, to represent a range. In that case `pos` is the beginning of the range and `n` is its size. That is, the range is `[begin() + pos, begin() + pos + n)`.

Note that the C++ standard does not specify the complexity of `basic_string` operations. In this implementation, `basic_string` has performance characteristics very similar to those of `vector`: access to a single character is *O(1)*, while copy and concatenation are *O(N)*. By contrast, `rope` has very different performance characteristics: most `rope` operations have logarithmic complexity.

Note also that, according to the C++ standard, `basic_string` has very unusual iterator invalidation semantics. Iterators may be invalidated by `swap`, `reserve`, `insert`, and `erase` (and by functions that are equivalent to `insert` and/or `erase`, such as `clear`, `resize`, `append`, and `replace`). Additionally, however, the first call to *any* non-const member function, including the non-const version of `begin()` or `operator[]`, may invalidate iterators. (The intent of these iterator invalidation rules is to give implementors greater freedom in implementation techniques.) In this implementation, `begin()`, `end()`, `rbegin()`, `rend()`, `operator[]`, `c_str()`, and `data()` do not invalidate iterators. In this implementation, iterators are only invalidated by member functions that explicitly change the string's contents.

Example

```
int main() {
  string s(10u, ' ');           // Create a string of ten blanks.

  const char* A = "this is a test";
  s += A;
  cout << "s = " << (s + '\n');
  cout << "As a null-terminated sequence: " << s.c_str() << endl;
  cout << "The sixteenth character is " << s[15] << endl;

  reverse(s.begin(), s.end());
  s.push_back('\n');
  cout << s;
```

```
}
```

Definition

Defined in the standard header string.

Template parameters

Parameter	Description	Default
`charT`	The string's value type: the type of character it contains.	
`traits`	The Character Traits type, which encapsulates basic character operations.	`char_traits<charT>`
`Alloc`	The string's allocator, used for internal memory management.	`alloc`

Model of

Random Access Container, Sequence.

Type requirements

In addition to the type requirements imposed by Random Access Container and Sequence:

- `charT` is a POD ("plain ol' data") type.
- `traits` is a Character Traits type whose value type is `charT`

Public base classes

None.

Members

Member	Where defined	Description
`value_type`	Container	The type of object, `charT`, stored in the string.
`pointer`	Container	Pointer to `charT`.
`reference`	Container	Reference to `charT`
`const_reference`	Container	Const reference to `charT`
`size_type`	Container	An unsigned integral type.
`difference_type`	Container	A signed integral type.
`static const size_type npos`	`basic_string`	The largest possible value of type `size_type`. That is, `size_type(-1)`.
`iterator`	Container	Iterator used to iterate through a string. A `basic_string` supplies Random Access Iterators.
`const_iterator`	Container	Const iterator used to iterate through a string.

`reverse_iterator`	Reversible Container	Iterator used to iterate backwards through a string.
`const_reverse_iterator`	Reversible Container	Const iterator used to iterate backwards through a string.
`iterator begin()`	Container	Returns an `iterator` pointing to the beginning of the string.
`iterator end()`	Container	Returns an `iterator` pointing to the end of the string.
`const_iterator begin() const`	Container	Returns a `const_iterator` pointing to the beginning of the string.
`const_iterator end() const`	Container	Returns a `const_iterator` pointing to the end of the string.
`reverse_iterator rbegin()`	Reversible Container	Returns a `reverse_iterator` pointing to the beginning of the reversed string.
`reverse_iterator rend()`	Reversible Container	Returns a `reverse_iterator` pointing to the end of the reversed string.
`const_reverse_iterator rbegin() const`	Reversible Container	Returns a `const_reverse_iterator` pointing to the beginning of the reversed string.
`const_reverse_iterator rend() const`	Reversible Container	Returns a `const_reverse_iterator` pointing to the end of the reversed string.
`size_type size() const`	Container	Returns the size of the string.
`size_type length() const`	`basic_string`	Synonym for `size()`.
`size_type max_size() const`	Container	Returns the largest possible size of the string.
`size_type capacity() const`	`basic_string`	See below.
`bool empty() const`	Container	`true` if the string's size is `0`.
`reference operator[](size_type n)`	Random Access Container	Returns the `n`'th character.
`const_reference operator[](size_type n) const`	Random Access Container	Returns the `n`'th character.
`const charT* c_str() const`	`basic_string`	Returns a pointer to a null-terminated array of characters representing the string's contents.
`const charT* data() const`	`basic_string`	Returns a pointer to an array of characters (not necessarily null-terminated) representing the string's contents.

`basic_string()`	Container	Creates an empty string.
`basic_string(const basic_string& s,` `  size_type pos = 0, size_type n = npos)`	Container, `basic_string`	Generalization of the copy constructor.
`basic_string(const charT*)`	`basic_string`	Construct a string from a null-terminated character array.
`basic_string(const charT* s, size_type n)`	`basic_string`	Construct a string from a character array and a length.
`basic_string(size_type n, charT c)`	Sequence	Create a string with `n` copies of `c`.
`template <class InputIterator>` `basic_string(InputIterator first, InputIterator last)`	Sequence	Create a string from a range.
`~basic_string()`	Container	The destructor.
`basic_string& operator=(const basic_string&)`	Container	The assignment operator
`basic_string& operator=(const charT* s)`	`basic_string`	Assign a null-terminated character array to a string.
`basic_string& operator=(charT c)`	`basic_string`	Assign a single character to a string.
`void reserve(size_t)`	`basic_string`	See below.
`void swap(basic_string&)`	Container	Swaps the contents of two strings.
`iterator insert(iterator pos,` `  const T& x)`	Sequence	Inserts `x` before `pos`.

`template <class InputIterator>`
`void insert(iterator pos,`
`  InputIterator f, InputIterator l)`
	Sequence	Inserts the range `[first, last)` before `pos`.

`void insert(iterator pos,` `  size_type n, const T& x)`	Sequence	Inserts `n` copies of `x` before `pos`.
`basic_string& insert(size_type pos, const basic_string& s)`	`basic_string`	Inserts `s` before `pos`.

`basic_string& insert(size_type pos,`
`  const basic_string& s,`
`  size_type pos1, size_type n)`
	`basic_string`	Inserts a substring of `s` before `pos`.

`basic_string& insert(size_type pos, const charT* s)`	`basic_string`	Inserts `s` before `pos`.
`basic_string& insert(size_type pos, const charT* s, size_type n)`	`basic_string`	Inserts the first n characters of `s` before `pos`.
`basic_string& insert(size_type pos, size_type n, charT c)`	`basic_string`	Inserts `n` copies of `c` before `pos`.
`basic_string& append(const basic_string& s)`	`basic_string`	Append `s` to `*this`.
`basic_string& append(const basic_string& s,` `  size_type pos, size_type n)`	`basic_string`	Append a substring of `s` to `*this`.
`basic_string& append(const charT* s)`	`basic_string`	Append `s` to `*this`.
`basic_string& append(const charT* s, size_type n)`	`basic_string`	Append the first `n` characters of `s` to `*this`.
`basic_string& append(size_type n, charT c)`	`basic_string`	Append `n` copies of `c` to `*this`.
`template <class InputIterator>` `basic_string& append(InputIterator first, InputIterator last)`	`basic_string`	Append a range to `*this`.

`void push_back(charT c)`	`basic_string`	Append a single character to `*this`.
`basic_string& operator+=(const basic_string& s)`	`basic_string`	Equivalent to `append(s)`.
`basic_string& operator+=(const charT* s)`	`basic_string`	Equivalent to `append(s)`
`basic_string& operator+=(charT c)`	`basic_string`	Equivalent to `push_back(c)`
`iterator erase(iterator p)`	Sequence	Erases the character at position `p`
`iterator erase(iterator first, iterator last)`	Sequence	Erases the range `[first, last)`
`basic_string& erase(size_type pos = 0, size_type n = npos)`	`basic_string`	Erases a range.
`void clear()`	Sequence	Erases the entire container.
`void resize(size_type n, charT c = charT())`	Sequence	Appends characters, or erases characters from the end, as necessary to make the string's length exactly `n` characters.
`basic_string& assign(const basic_string&)`	`basic_string`	Synonym for `operator=`
`basic_string& assign(const basic_string& s,` `size_type pos, size_type n)`	`basic_string`	Assigns a substring of `s` to `*this`
`basic_string& assign(const charT* s, size_type n)`	`basic_string`	Assigns the first `n` characters of `s` to `*this`.
`basic_string& assign(const charT* s)`	`basic_string`	Assigns a null-terminated array of characters to `*this`.
`basic_string& assign(size_type n, charT c)`	Sequence	Erases the existing characters and replaces them by `n` copies of `c`.
`template <class InputIterator>` `basic_string& assign(InputIterator first, InputIterator last)`	Sequence	Erases the existing characters and replaces them by `[first, last)`
`basic_string& replace(size_type pos, size_type n,` `const basic_string& s)`	`basic_string`	Replaces a substring of `*this` with the string `s`.

`basic_string& replace(size_type pos, size_type n,`
`const basic_string& s,`
`size_type pos1, size_type n1)`
	`basic_string`	Replaces a substring of `*this` with a substring of `s`.

`basic_string& replace(size_type pos, size_type n,` `const charT* s, size_type n1)`	`basic_string`	Replaces a substring of `*this` with the first `n1` characters of `s`.
`basic_string& replace(size_type pos, size_type n,` `const charT* s)`	`basic_string`	Replaces a substring of `*this` with a null-terminated character array.
`basic_string& replace(size_type pos, size_type n,` `size_type n1, charT c)`	`basic_string`	Replaces a substring of `*this` with `n1` copies of `c`.
`basic_string& replace(iterator first, iterator last,` `const basic_string& s)`	`basic_string`	Replaces a substring of `*this` with the string `s`.
`basic_string& replace(iterator first, iterator last,` `const charT* s, size_type n)`	`basic_string`	Replaces a substring of `*this` with the first `n` characters of `s`.
`basic_string& replace(iterator first, iterator last,` `const charT* s)`	`basic_string`	Replaces a substring of `*this` with a null-terminated

		character array.
basic_string& replace(iterator first, iterator last, size_type n, charT c)	basic_string	Replaces a substring of *this with n copies of c.
template <class InputIterator> basic_string& replace(iterator first, iterator last, InputIterator f, InputIterator l)	basic_string	Replaces a substring of *this with the range [f, l)
size_type copy(charT* buf, size_type n, size_type pos = 0) const	basic_string	Copies a substring of *this to a buffer.
size_type find(const basic_string& s, size_type pos = 0) const	basic_string	Searches for s as a substring of *this, beginning at character pos of *this.
size_type find(const charT* s, size_type pos, size_type n) const	basic_string	Searches for the first n characters of s as a substring of *this, beginning at character pos of *this.
size_type find(const charT* s, size_type pos = 0) const	basic_string	Searches for a null-terminated character array as a substring of *this, beginning at character pos of *this.
size_type find(charT c, size_type pos = 0) const	basic_string	Searches for the character c, beginning at character position pos.
size_type rfind(const basic_string& s, size_type pos = npos) const	basic_string	Searches backward for s as a substring of *this, beginning at character position min(pos, size())
size_type rfind(const charT* s, size_type pos, size_type n) const	basic_string	Searches backward for the first n characters of s as a substring of *this, beginning at character position min(pos, size())
size_type rfind(const charT* s, size_type pos = npos) const	basic_string	Searches backward for a null-terminated character array as a substring of *this, beginning at character min(pos, size())
size_type rfind(charT c, size_type pos = npos) const	basic_string	Searches backward for the character c, beginning at character position min(pos, size().
size_type find_first_of(const basic_string& s, size_type pos = 0) const	basic_string	Searches within *this, beginning at pos, for the first character that is equal to any character within s.
size_type find_first_of(const charT* s, size_type pos, size_type n) const	basic_string	Searches within *this, beginning at pos, for the first character that is equal to any character within the first n characters of s.
size_type find_first_of(const charT* s, size_type pos = 0) const	basic_string	Searches within *this, beginning at pos, for the first

		character that is equal to any character within `s`.
`size_type find_first_of(charT c, size_type pos = 0) const`	`basic_string`	Searches within `*this`, beginning at `pos`, for the first character that is equal to `c`.
`size_type find_first_not_of(const basic_string& s, size_type pos = 0) const`	`basic_string`	Searches within `*this`, beginning at `pos`, for the first character that is not equal to any character within `s`.
`size_type find_first_not_of(const charT* s, size_type pos, size_type n) const`	`basic_string`	Searches within `*this`, beginning at `pos`, for the first character that is not equal to any character within the first `n` characters of `s`.
`size_type find_first_not_of(const charT* s, size_type pos = 0) const`	`basic_string`	Searches within `*this`, beginning at `pos`, for the first character that is not equal to any character within `s`.
`size_type find_first_not_of(charT c, size_type pos = 0) const`	`basic_string`	Searches within `*this`, beginning at `pos`, for the first character that is not equal to `c`.
`size_type find_last_of(const basic_string& s, size_type pos = npos) const`	`basic_string`	Searches backward within `*this`, beginning at `min(pos, size())`, for the first character that is equal to any character within `s`.
`size_type find_last_of(const charT* s, size_type pos, size_type n) const`	`basic_string`	Searches backward within `*this`, beginning at `min(pos, size())`, for the first character that is equal to any character within the first `n` characters of `s`.
`size_type find_last_of(const charT* s, size_type pos = npos) const`	`basic_string`	Searches backward `*this`, beginning at `min(pos, size())`, for the first character that is equal to any character within `s`.
`size_type find_last_of(charT c, size_type pos = npos) const`	`basic_string`	Searches backward `*this`, beginning at `min(pos, size())`, for the first character that is equal to `c`.
`size_type find_last_not_of(const basic_string& s, size_type pos = npos) const`	`basic_string`	Searches backward within `*this`, beginning at `min(pos, size())`, for the first character that is not equal to any character within `s`.
`size_type find_last_not_of(const charT* s, size_type pos, size_type n) const`	`basic_string`	Searches backward within `*this`, beginning at `min(pos, size())`, for the first character that is not equal to any

		character within the first `n` characters of `s`.
`size_type find_last_not_of(const charT* s, size_type pos = npos) const`	`basic_string`	Searches backward `*this`, beginning at `min(pos, size())`, for the first character that is not equal to any character within `s`.
`size_type find_last_not_of(charT c, size_type pos = npos) const`	`basic_string`	Searches backward `*this`, beginning at `min(pos, size())`, for the first character that is not equal to `c`.
`basic_string substr(size_type pos = 0, size_type n = npos) const`	`basic_string`	Returns a substring of `*this`.
`int compare(const basic_string& s) const`	`basic_string`	Three-way lexicographical comparison of `s` and `*this`.
`int compare(size_type pos, size_type n, const basic_string& s) const`	`basic_string`	Three-way lexicographical comparison of `s` and a substring of `*this`.
`int compare(size_type pos, size_type n, const basic_string& s, size_type pos1, size_type n1) const`	`basic_string`	Three-way lexicographical comparison of a substring of `s` and a substring of `*this`.
`int compare(const charT* s) const`	`basic_string`	Three-way lexicographical comparison of `s` and `*this`.
`int compare(size_type pos, size_type n, const charT* s, size_type len = npos) const`	`basic_string`	Three-way lexicographical comparison of the first `min(len, traits::length(s)` characters of `s` and a substring of `*this`.
`template <class charT, class traits, class Alloc> basic_string<charT, traits, Alloc> operator+(const basic_string<charT, traits, Alloc>& s1, const basic_string<charT, traits, Alloc>& s2)`	`basic_string`	String concatenation. A global function, not a member function.
`template <class charT, class traits, class Alloc> basic_string<charT, traits, Alloc> operator+(const charT* s1, const basic_string<charT, traits, Alloc>& s2)`	`basic_string`	String concatenation. A global function, not a member function.
`template <class charT, class traits, class Alloc> basic_string<charT, traits, Alloc> operator+(const basic_string<charT, traits, Alloc>& s1, const charT* s2)`	`basic_string`	String concatenation. A global function, not a member function.
`template <class charT, class traits, class Alloc> basic_string<charT, traits, Alloc> operator+(charT c, const basic_string<charT, traits, Alloc>& s2)`	`basic_string`	String concatenation. A global function, not a member function.
`template <class charT, class traits, class Alloc> basic_string<charT, traits, Alloc> operator+(const basic_string<charT, traits, Alloc>& s1, charT c)`	`basic_string`	String concatenation. A global function, not a member function.
`template <class charT, class traits, class Alloc> bool operator==(const basic_string<charT, traits, Alloc>& s1, const basic_string<charT, traits, Alloc>& s2)`	Container	String equality. A global function, not a member function.
`template <class charT, class traits, class Alloc> bool operator==(const charT* s1, const basic_string<charT, traits, Alloc>& s2)`	`basic_string`	String equality. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`bool operator==(const basic_string<charT, traits, Alloc>& s1,`
`const charT* s2)`
	`basic_string`	String equality. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`bool operator!=(const basic_string<charT, traits, Alloc>& s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	Container	String inequality. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`bool operator!=(const charT* s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	`basic_string`	String inequality. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`bool operator!=(const basic_string<charT, traits, Alloc>& s1,`
`const charT* s2)`
	`basic_string`	String inequality. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`bool operator<(const basic_string<charT, traits, Alloc>& s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	Container	String comparison. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`bool operator<(const charT* s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	`basic_string`	String comparison. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`bool operator<(const basic_string<charT, traits, Alloc>& s1,`
`const charT* s2)`
	`basic_string`	String comparison. A global function, not a member function.

`template <class charT, class traits, class Alloc>`
`void swap(basic_string<charT, traits, Alloc>& s1,`
`basic_string<charT, traits, Alloc>& s2)`
	Container	Swaps the contents of two strings.

`template <class charT, class traits, class Alloc>`
`basic_istream<charT, traits>&`
`operator>>(basic_istream<charT, traits>& is,`
`basic_string<charT, traits, Alloc>& s)`
	`basic_string`	Reads `s` from the input stream `is`

`template <class charT, class traits, class Alloc>`
`basic_ostream(charT, traits)&`
`operator<<(basic_ostream<charT, traits>& os,`
`basic_string<charT, traits, Alloc>& s)`
	`basic_string`	Writes `s` to the output stream `os`

`template <class charT, class traits, class Alloc>`
`basic_istream<charT, traits>&`
`getline(basic_istream<charT, traits>& is,`
`basic_string<charT, traits, Alloc>& s,`
`charT delim)`
	`basic_string`	Reads a string from the input stream `is`, stopping when it reaches `delim`

`template <class charT, class traits, class Alloc>`
`basic_istream<charT, traits>&`
`getline(basic_istream<charT, traits>& is,`
`basic_string<charT, traits, Alloc>& s)`
	`basic_string`	Reads a single line from the input stream `is`

New members

These members are not defined in the Random Access Container and Sequence: requirements, but are specific to `basic_string`.

Member	Description
`static const size_type npos`	The largest possible value of type `size_type`. That is, `size_type(-1)`.
`size_type length() const`	Equivalent to `size()`.

`size_type capacity() const`	Number of elements for which memory has been allocated. That is, the size to which the string can grow before memory must be reallocated. `capacity()` is always greater than or equal to `size()`.
`const charT* c_str() const`	Returns a pointer to a null-terminated array of characters representing the string's contents. For any string `s` it is guaranteed that the first `s.size()` characters in the array pointed to by `s.c_str()` are equal to the character in `s`, and that `s.c_str()[s.size()]` is a null character. Note, however, that it not necessarily the first null character. Characters within a string are permitted to be null.
`const charT* data() const`	Returns a pointer to an array of characters, not necessarily null-terminated, representing the string's contents. `data()` is permitted, but not required, to be identical to `c_str()`. The first `size()` characters of that array are guaranteed to be identical to the characters in `*this`. The return value of `data()` is never a null pointer, even if `size()` is zero.
`basic_string(const basic_string& s, size_type pos = 0, size_type n = npos)`	Constructs a string from a substring of `s`. The substring begins at character position `pos` and terminates at character position `pos + n` or at the end of `s`, whichever comes first. This constructor throws `out_of_range` if `pos > s.size()`. Note that when `pos` and `n` have their default values, this is just a copy constructor.
`basic_string(const charT* s)`	Equivalent to `basic_string(s, s + traits::length(s))`.
`basic_string(const charT* s, size_type n)`	Equivalent to `basic_string(s, s + n)`.
`basic_string& operator=(const charT* s)`	Equivalent to `operator=(basic_string(s))`.
`basic_string& operator=(charT c)`	Assigns to `*this` a string whose size is `1` and whose

	contents is the single character `c`.
`void reserve(size_t n)`	Requests that the string's capacity be changed; the postcondition for this member function is that, after it is called, `capacity() >= n`. You may request that a string decrease its capacity by calling `reserve()` with an argument less than the current capacity. (If you call `reserve()` with an argument less than the string's size, however, the capacity will only be reduced to `size()`. A string's size can never be greater than its capacity.) `reserve()` throws `length_error` if `n > max_size()`.
`basic_string& insert(size_type pos, const basic_string& s)`	If `pos > size()`, throws `out_of_range`. Otherwise, equivalent to `insert(begin() + pos, s.begin(), s.end())`.
`basic_string& insert(size_type pos, const basic_string& s, size_type pos1, size_type n)`	If `pos > size()` or `pos1 > s.size()`, throws `out_of_range`. Otherwise, equivalent to `insert(begin() + pos, s.begin() + pos1, s.begin() + pos1 + min(n, s.size() - pos1))`.
`basic_string& insert(size_type pos, const charT* s)`	If `pos > size()`, throws `out_of_range`. Otherwise, equivalent to `insert(begin() + pos, s, s + traits::length(s))`
`basic_string& insert(size_type pos, const charT* s, size_type n)`	If `pos > size()`, throws `out_of_range`. Otherwise, equivalent to `insert(begin() + pos, s, s + n)`.
`basic_string& insert(size_type pos, size_type n, charT c)`	If `pos > size()`, throws `out_of_range`. Otherwise, equivalent to `insert(begin() + pos, n, c)`.
`basic_string& append(const basic_string& s)`	Equivalent to `insert(end(), s.begin(), s.end())`.
`basic_string& append(const basic_string& s, size_type pos, size_type n)`	If `pos > s.size()`, throws `out_of_range`. Otherwise, equivalent to `insert(end(), s.begin() + pos, s.begin() + pos + min(n, s.size() - pos))`.
`basic_string& append(const charT* s)`	

	Equivalent to `insert(end(), s, s + traits::length(s))`.
`basic_string& append(const charT* s, size_type n)`	Equivalent to `insert(end(), s, s + n)`.
`basic_string& append(size_type n, charT c)`	Equivalent to `insert(end(), n, c)`.
`template <class InputIterator>` `basic_string& append(InputIterator first, InputIterator last)`	Equivalent to `insert(end(), first, last)`.
`void push_back(charT c)`	Equivalent to `insert(end(), c)`
`basic_string& operator+=(const basic_string& s)`	Equivalent to `append(s)`.
`basic_string& operator+=(const charT* s)`	Equivalent to `append(s)`
`basic_string& operator+=(charT c)`	Equivalent to `push_back(c)`
`basic_string& erase(size_type pos = 0, size_type n = npos)`	If `pos > size()`, throws `out_of_range`. Otherwise, equivalent to `erase(begin() + pos, begin() + pos + min(n, size() - pos))`.
`basic_string& assign(const basic_string& s)`	Synonym for `operator=`
`basic_string& assign(const basic_string& s,` `size_type pos, size_type n)`	Equivalent to (but probably faster than) `clear()` followed by `insert(0, s, pos, n)`.
`basic_string& assign(const charT* s, size_type n)`	Equivalent to (but probably faster than) `clear()` followed by `insert(0, s, n)`.
`basic_string& assign(const charT* s)`	Equivalent to (but probably faster than) `clear()` followed by `insert(0, s)`.
`basic_string& replace(size_type pos, size_type n,` `const basic_string& s)`	Equivalent to `erase(pos, n)` followed by `insert(pos, s)`.
`basic_string& replace(size_type pos, size_type n,` `const basic_string& s, size_type pos1, size_type n1)`	Equivalent to `erase(pos, n)` followed by `insert(pos, s, pos1, n1)`.
`basic_string& replace(size_type pos, size_type n,` `const charT* s, size_type n1)`	Equivalent to `erase(pos, n)` followed by `insert(pos, s, n1)`.
`basic_string& replace(size_type pos, size_type n,` `const charT* s)`	Equivalent to `erase(pos, n)` followed by `insert(pos, s)`.
`basic_string& replace(size_type pos, size_type n,` `size_type n1, charT c)`	Equivalent to `erase(pos, n)` followed by `insert(pos, n1, c)`.
`basic_string& replace(iterator first, iterator last,` `const basic_string& s)`	Equivalent to `insert(erase(first, last), s.begin(), s.end())`.
`basic_string& replace(iterator first, iterator last,` `const charT* s, size_type n)`	Equivalent to `insert(erase(first, last), s, s + n)`.
`basic_string& replace(iterator first, iterator last,`	

`const charT* s)`	Equivalent to `insert(erase(first, last), s, s + traits::length(s))`.
`basic_string& replace(iterator first, iterator last, size_type n, charT c)`	Equivalent to `insert(erase(first, last), n, c)`.
`template <class InputIterator> basic_string& replace(iterator first, iterator last, InputIterator f, InputIterator l)`	Equivalent to `insert(erase(first, last), f, l)`.
`size_type copy(charT* buf, size_type n, size_type pos = 0) const`	Copies at most `n` characters from `*this` to a character array. Throws `out_of_range` if `pos > size()`. Otherwise, equivalent to `copy(begin() + pos, begin() + pos + min(n, size()), buf)`. Note that this member function does nothing other than copy characters from `*this` to `buf`; in particular, it does not terminate `buf` with a null character.
`size_type find(const basic_string& s, size_type pos = 0) const`	Searches for `s` as a substring of `*this`, beginning at character position `pos`. It is almost the same as `search`, except that `search` tests elements for equality using `operator==` or a user-provided function object, while this member function uses `traits::eq`. Returns the lowest character position `N` such that `pos <= N` and `pos + s.size() <= size()` and such that, for every `i` less than `s.size()`, `(*this)[N + i]` compares equal to `s[i]`. Returns `npos` if no such position `N` exists. Note that it is legal to call this member function with arguments such that `s.size() > size() - pos`, but such a search will always fail.
`size_type find(const charT* s, size_type pos, size_type n) const`	Searches for the first `n` characters of `s` as a substring of `*this`, beginning at character `pos` of `*this`. This is equivalent to `find(basic_string(s, n), pos)`.
`size_type find(const charT* s, size_type pos = 0) const`	Searches for a null-terminated character array as a substring of `*this`, beginning at

	character `pos` of `*this`. This is equivalent to `find(basic_string(s), pos)`.
`size_type find(charT c, size_type pos = 0) const`	Searches for the character `c`, beginning at character position `pos`. That is, returns the first character position `N` greater than or equal to `pos`, and less than `size()`, such that `(*this)[N]` compares equal to `c`. Returns `npos` if no such character position `N` exists.
`size_type rfind(const basic_string& s, size_type pos = npos) const`	Searches backward for `s` as a substring of `*this`. It is almost the same as `find_end`, except that `find_end` tests elements for equality using `operator==` or a user-provided function object, while this member function uses `traits::eq`. This member function returns the largest character position `N` such that `N <= pos` and `N + s.size() <= size()`, and such that, for every `i` less than `s.size()`, `(*this)[N + i]` compares equal to `s[i]`. Returns `npos` if no such position `N` exists. Note that it is legal to call this member function with arguments such that `s.size() > size()`, but such a search will always fail.
`size_type rfind(const charT* s, size_type pos, size_type n) const`	Searches backward for the first `n` characters of `s` as a substring of `*this`. Equivalent to `rfind(basic_string(s, n), pos)`.
`size_type rfind(const charT* s, size_type pos = npos) const`	Searches backward for a null-terminated character array as a substring of `*this`. Equivalent to `rfind(basic_string(s), pos)`.
`size_type rfind(charT c, size_type pos = npos) const`	Searches backward for the character `c`. That is, returns the largest character position `N` such that `N <= pos` and `N < size()`, and such that `(*this)[N]` compares equal to `c`. Returns `npos` if no such character position exists.
`size_type find_first_of(const basic_string& s, size_type pos = 0) const`	Searches within `*this`,

	beginning at `pos`, for the first character that is equal to any character within `s`. This is similar to the standard algorithm `find_first_of`, but differs because `find_first_of` compares characters using `operator==` or a user-provided function object, while this member function uses `traits::eq`. Returns the smallest character position `N` such that `pos <= N < size()`, and such that `(*this)[N]` compares equal to some character within `s`. Returns `npos` if no such character position exists.
`size_type find_first_of(const charT* s, size_type pos, size_type n) const`	Searches within `*this`, beginning at `pos`, for the first character that is equal to any character within the range `[s, s+n)`. That is, returns the smallest character position `N` such that `pos <= N < size()`, and such that `(*this)[N]` compares equal to some character in `[s, s+n)`. Returns `npos` if no such character position exists.
`size_type find_first_of(const charT* s, size_type pos = 0) const`	Equivalent to `find_first_of(s, pos, traits::length(s))`.
`size_type find_first_of(charT c, size_type pos = 0) const`	Equivalent to `find(c, pos)`.
`size_type find_first_not_of(const basic_string& s, size_type pos = 0) const`	Searches within `*this`, beginning at `pos`, for the first character that is not equal to any character within `s`. Returns the smallest character position `N` such that `pos <= N < size()`, and such that `(*this)[N]` does not compare equal to any character within `s`. Returns `npos` if no such character position exists.
`size_type find_first_not_of(const charT* s, size_type pos, size_type n) const`	Searches within `*this`, beginning at `pos`, for the first character that is not equal to any character within the range `[s, s+n)`. That is, returns the smallest character position `N` such that `pos <= N < size()`,

	and such that `(*this)[N]` does not compare equal to any character in `[s, s+n)`. Returns `npos` if no such character position exists.
`size_type find_first_not_of(const charT* s, size_type pos = 0) const`	Equivalent to `find_first_not_of(s, pos, traits::length(s))`.
`size_type find_first_not_of(charT c, size_type pos = 0) const`	Returns the smallest character position `N` such that `pos <= N < size()`, and such that `(*this)[N]` does not compare equal to `c`. Returns `npos` if no such character position exists.
`size_type find_last_of(const basic_string& s, size_type pos = npos) const`	Searches backward within `*this` for the first character that is equal to any character within `s`. That is, returns the largest character position `N` such that `N <= pos` and `N < size()`, and such that `(*this)[N]` compares equal to some character within `s`. Returns `npos` if no such character position exists.
`size_type find_last_of(const charT* s, size_type pos, size_type n) const`	Searches backward within `*this` for the first character that is equal to any character within the range `[s, s+n)`. That is, returns the largest character position `N` such that `N <= pos` and `N < size()`, and such that `(*this)[N]` compares equal to some character within `[s, s+n)`. Returns `npos` if no such character position exists.
`size_type find_last_of(const charT* s, size_type pos = npos) const`	Equivalent to `find_last_of(s, pos, traits::length(s))`.
`size_type find_last_of(charT c, size_type pos = npos) const`	Equivalent to `rfind(c, pos)`.
`size_type find_last_not_of(const basic_string& s, size_type pos = npos) const`	Searches backward within `*this` for the first character that is not equal to any character within `s`. That is, returns the largest character position `N` such that `N <= pos` and `N < size()`, and such that `(*this)[N]` does not compare equal to any character within `s`. Returns `npos` if no such character position exists.
`size_type find_last_not_of(const charT* s, size_type pos, size_type n) const`	Searches backward within `*this` for the first character

	that is not equal to any character within `[s, s+n)`. That is, returns the largest character position `N` such that `N <= pos` and `N < size()`, and such that `(*this)[N]` does not compare equal to any character within `[s, s+n)`. Returns `npos` if no such character position exists.
`size_type find_last_not_of(const charT* s, size_type pos = npos) const`	Equivalent to `find_last_of(s, pos, traits::length(s))`.
`size_type find_last_not_of(charT c, size_type pos = npos) const`	Searches backward `*this` for the first character that is not equal to `c`. That is, returns the largest character position `N` such that `N <= pos` and `N < size()`, and such that `(*this)[N]` does not compare equal to `c`.
`basic_string substr(size_type pos = 0, size_type n = npos) const`	Equivalent to `basic_string(*this, pos, n)`.
`int compare(const basic_string& s) const`	Three-way lexicographical comparison of `s` and `*this`, much like `strcmp`. If `traits::compare(data, s.data(), min(size(), s.size()))` is nonzero, then it returns that nonzero value. Otherwise returns a negative number if `size() < s.size()`, a positive number if `size() > s.size()`, and zero if the two are equal.
`int compare(size_type pos, size_type n, const basic_string& s) const`	Three-way lexicographical comparison of `s` and a substring of `*this`. Equivalent to `basic_string(*this, pos, n).compare(s)`.
`int compare(size_type pos, size_type n, const basic_string& s, size_type pos1, size_type n1) const`	Three-way lexicographical comparison of a substring of `s` and a substring of `*this`. Equivalent to `basic_string(*this, pos, n).compare(basic_string(s, pos1, n1))`.
`int compare(const charT* s) const`	Three-way lexicographical comparison of `s` and `*this`. Equivalent to `compare(basic_string(s))`.
`int compare(size_type pos, size_type n, const charT* s, size_type len = npos) const`	Three-way lexicographical comparison of the first `min(len, traits::length(s)`

	characters of `s` and a substring of `*this`. Equivalent to `basic_string(*this, pos, n).compare(basic_string(s, min(len, traits::length(s))))`.

`template <class charT, class traits, class Alloc>`
`basic_string<charT, traits, Alloc>`
`operator+(const basic_string<charT, traits, Alloc>& s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	String concatenation. Equivalent to creating a temporary copy of `s`, appending `s2`, and then returning the temporary copy.

`template <class charT, class traits, class Alloc>`
`basic_string<charT, traits, Alloc>`
`operator+(const charT* s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	String concatenation. Equivalent to creating a temporary `basic_string` object from `s1`, appending `s2`, and then returning the temporary object.

`template <class charT, class traits, class Alloc>`
`basic_string<charT, traits, Alloc>`
`operator+(const basic_string<charT, traits, Alloc>& s1,`
`const charT* s2)`
	String concatenation. Equivalent to creating a temporary copy of `s`, appending `s2`, and then returning the temporary copy.

`template <class charT, class traits, class Alloc>`
`basic_string<charT, traits, Alloc>`
`operator+(charT c,`
`const basic_string<charT, traits, Alloc>& s2)`
	String concatenation. Equivalent to creating a temporary object with the constructor `basic_string(1, c)`, appending `s2`, and then returning the temporary object.

`template <class charT, class traits, class Alloc>`
`basic_string<charT, traits, Alloc>`
`operator+(const basic_string<charT, traits, Alloc>& s1,`
`charT c)`
	String concatenation. Equivalent to creating a temporary object, appending `c` with `push_back`, and then returning the temporary object.

`template <class charT, class traits, class Alloc>`
`bool operator==(const charT* s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	String equality. Equivalent to `basic_string(s1).compare(s2) == 0`.

`template <class charT, class traits, class Alloc>`
`bool operator==(const basic_string<charT, traits, Alloc>& s1,`
`const charT* s2)`
	String equality. Equivalent to `basic_string(s1).compare(s2) == 0`.

`template <class charT, class traits, class Alloc>`
`bool operator!=(const charT* s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	String inequality. Equivalent to `basic_string(s1).compare(s2) != 0`.

`template <class charT, class traits, class Alloc>`
`bool operator!=(const basic_string<charT, traits, Alloc>& s1,`
`const charT* s2)`
	String inequality. Equivalent to `basic_string(s1).compare(s2) != 0`.

`template <class charT, class traits, class Alloc>`
`bool operator<(const charT* s1,`
`const basic_string<charT, traits, Alloc>& s2)`
	String comparison. Equivalent to `` `(s1 == s2)` ``.
	In addition returns whether or not `s1` is lexographically lesser than `s2`.

`template <class charT, class traits, class Alloc>`	String comparison. Equivalent

`bool operator<(const basic_string<charT, traits, Alloc>& s1, const charT* s2)`	to `(s1 == s2)`. In addition returns whether or not `s1` is lexographically lesser than `s2`.
`template <class charT, class traits, class Alloc> basic_istream<charT, traits>& operator>>(basic_istream<charT, traits>& is, basic_string<charT, traits, Alloc>& s)`	Reads `s` from the input stream `is`. Specifically, it skips whitespace, and then replaces the contents of `s` with characters read from the input stream. It continues reading characters until it encounters a whitespace character (in which case that character is not extracted), or until end-of-file, or, if `is.width()` is nonzero, until it has read `is.width()` characters. This member function resets `is.width()` to zero.
`template <class charT, class traits, class Alloc> basic_ostream<charT, traits>& operator<<(basic_ostream<charT, traits>& is, const basic_string<charT, traits, Alloc>& s)`	Writes `s` to the output stream `is`. It writes `max(s.size(), is.width())` characters, padding as necessary. This member function resets `is.width()` to zero.
`template <class charT, class traits, class Alloc> basic_istream<charT, traits>& getline(basic_istream<charT, traits>& is, basic_string<charT, traits, Alloc>& s, charT delim)`	Replaces the contents of `s` with characters read from the input stream. It continues reading characters until it encounters the character `delim` (in which case that character is extracted but not stored in `s`), or until end of file. Note that `getline`, unlike `operator>>`, does not skip whitespace. As the name suggests, it is most commonly used to read an entire line of text precisely as the line appears in an input file.
`template <class charT, class traits, class Alloc> basic_istream<charT, traits>& getline(basic_istream<charT, traits>& is, basic_string<charT, traits, Alloc>& s)`	Equivalent to `getline(is, s, is.widen('\n'))`.

Notes

See also

`rope`, `vector`, Character Traits
