class template

<string>

**std::basic\_string**

template < class charT,

class traits = char\_traits<charT>, // basic\_string::traits\_type

class Alloc = allocator<charT> // basic\_string::allocator\_type

> class basic\_string;

Generic string class

The basic\_string is the generalization of class [string](http://www.cplusplus.com/string) for any character type (see [string](http://www.cplusplus.com/string) for a description).

**Template parameters**

charT

Character type.  
The string is formed by a sequence of characters of this type.  
This shall be a non-array [POD type](http://www.cplusplus.com/is_pod).

traits

[Character traits](http://www.cplusplus.com/char_traits) class that defines essential properties of the characters used by [basic\_string](http://www.cplusplus.com/basic_string) objects (see [char\_traits](http://www.cplusplus.com/char_traits)).  
traits::char\_type shall be the same as charT.  
Aliased as member type basic\_string::traits\_type.

Alloc

Type of the allocator object used to define the storage allocation model. By default, the [allocator](http://www.cplusplus.com/allocator) class template is used, which defines the simplest memory allocation model and is value-independent.  
Aliased as member type basic\_string::allocator\_type.

Note: Because the first template parameter is not aliased as any member type, charT is used throughout this reference to refer to this type.

**Template instantiations**

[**string**](http://www.cplusplus.com/reference/string/string/)

String class (class )

[**wstring**](http://www.cplusplus.com/reference/string/wstring/)

Wide string (class )

[**u16string**](http://www.cplusplus.com/reference/string/u16string/)

String of 16-bit characters (class )

[**u32string**](http://www.cplusplus.com/reference/string/u32string/)

String of 32-bit characters (class )

**Member types**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |  |
| --- | --- | --- |
| **member type** | **definition** | **notes** |
| traits\_type | The second template parameter (traits) | defaults to: [char\_traits](http://www.cplusplus.com/char_traits)<charT> |
| allocator\_type | The third template parameter (Alloc) | defaults to: [allocator](http://www.cplusplus.com/allocator)<charT> |
| value\_type | traits\_type::char\_type | shall be the same as charT |
| reference | allocator\_type::reference | for the default [allocator](http://www.cplusplus.com/allocator): charT& |
| const\_reference | allocator\_type::const\_reference | for the default [allocator](http://www.cplusplus.com/allocator): const charT& |
| pointer | allocator\_type::pointer | for the default [allocator](http://www.cplusplus.com/allocator): charT\* |
| const\_pointer | allocator\_type::const\_pointer | for the default [allocator](http://www.cplusplus.com/allocator): const charT\* |
| iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to charT | convertible to const\_iterator |
| const\_iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to const charT |  |
| reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<iterator> |  |
| const\_reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<const\_iterator> |  |
| difference\_type | allocator\_type::difference\_type | usually the same as [ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) |
| size\_type | allocator\_type::difference\_type | usually the same as [size\_t](http://www.cplusplus.com/size_t) |

**Member functions**

[**(constructor)**](http://www.cplusplus.com/reference/string/basic_string/basic_string/)

Construct basic\_string object (public member function )

[**(destructor)**](http://www.cplusplus.com/reference/string/basic_string/%7Ebasic_string/)

String destructor (public member function )

[**operator=**](http://www.cplusplus.com/reference/string/basic_string/operator=/)

String assignment (public member function )

**Iterators**:

[**begin**](http://www.cplusplus.com/reference/string/basic_string/begin/)

Return iterator to beginning (public member function )

[**end**](http://www.cplusplus.com/reference/string/basic_string/end/)

Return iterator to end (public member function )

[**rbegin**](http://www.cplusplus.com/reference/string/basic_string/rbegin/)

Return reverse iterator to reverse beginning (public member function )

[**rend**](http://www.cplusplus.com/reference/string/basic_string/rend/)

Return reverse iterator to reverse end (public member function )

[**cbegin**](http://www.cplusplus.com/reference/string/basic_string/cbegin/)

Return const\_iterator to beginning (public member function )

[**cend**](http://www.cplusplus.com/reference/string/basic_string/cend/)

Return const\_iterator to end (public member function )

[**crbegin**](http://www.cplusplus.com/reference/string/basic_string/crbegin/)

Return const\_reverse\_iterator to reverse beginning (public member function )

[**crend**](http://www.cplusplus.com/reference/string/basic_string/crend/)

Return const\_reverse\_iterator to reverse end (public member function )

**Capacity**:

[**size**](http://www.cplusplus.com/reference/string/basic_string/size/)

Return size (public member function )

[**length**](http://www.cplusplus.com/reference/string/basic_string/length/)

Return length of string (public member function )

[**max\_size**](http://www.cplusplus.com/reference/string/basic_string/max_size/)

Return maximum size (public member function )

[**resize**](http://www.cplusplus.com/reference/string/basic_string/resize/)

Resize string (public member function )

[**capacity**](http://www.cplusplus.com/reference/string/basic_string/capacity/)

Return size of allocated storage (public member function )

[**reserve**](http://www.cplusplus.com/reference/string/basic_string/reserve/)

Request a change in capacity (public member function )

[**clear**](http://www.cplusplus.com/reference/string/basic_string/clear/)

Clear string (public member function )

[**empty**](http://www.cplusplus.com/reference/string/basic_string/empty/)

Test whether string is empty (public member function )

[**shrink\_to\_fit**](http://www.cplusplus.com/reference/string/basic_string/shrink_to_fit/)

Shrink to fit (public member function )

**Element access**:

[**operator[]**](http://www.cplusplus.com/reference/string/basic_string/operator%5b%5d/)

Get character of string (public member function )

[**at**](http://www.cplusplus.com/reference/string/basic_string/at/)

Get character of string (public member function )

[**back**](http://www.cplusplus.com/reference/string/basic_string/back/)

Access last character (public member function )

[**front**](http://www.cplusplus.com/reference/string/basic_string/front/)

Access first character (public member function )

**Modifiers**:

[**operator+=**](http://www.cplusplus.com/reference/string/basic_string/operator+=/)

Append to string (public member function )

[**append**](http://www.cplusplus.com/reference/string/basic_string/append/)

Append to string (public member function )

[**push\_back**](http://www.cplusplus.com/reference/string/basic_string/push_back/)

Append character to string (public member function )

[**assign**](http://www.cplusplus.com/reference/string/basic_string/assign/)

Assign content to string (public member function )

[**insert**](http://www.cplusplus.com/reference/string/basic_string/insert/)

Insert into string (public member function )

[**erase**](http://www.cplusplus.com/reference/string/basic_string/erase/)

Erase characters from string (public member function )

[**replace**](http://www.cplusplus.com/reference/string/basic_string/replace/)

Replace portion of string (public member function )

[**swap**](http://www.cplusplus.com/reference/string/basic_string/swap/)

Swap string values (public member function )

[**pop\_back**](http://www.cplusplus.com/reference/string/basic_string/pop_back/)

Delete last character (public member function )

**String operations**:

[**c\_str**](http://www.cplusplus.com/reference/string/basic_string/c_str/)

Get C-string equivalent

[**data**](http://www.cplusplus.com/reference/string/basic_string/data/)

Get string data (public member function )

[**get\_allocator**](http://www.cplusplus.com/reference/string/basic_string/get_allocator/)

Get allocator (public member function )

[**copy**](http://www.cplusplus.com/reference/string/basic_string/copy/)

Copy sequence of characters from string (public member function )

[**find**](http://www.cplusplus.com/reference/string/basic_string/find/)

Find first occurrence in string (public member function )

[**rfind**](http://www.cplusplus.com/reference/string/basic_string/rfind/)

Find last occurrence in string (public member function )

[**find\_first\_of**](http://www.cplusplus.com/reference/string/basic_string/find_first_of/)

Find character in string (public member function )

[**find\_last\_of**](http://www.cplusplus.com/reference/string/basic_string/find_last_of/)

Find character in string from the end (public member function )

[**find\_first\_not\_of**](http://www.cplusplus.com/reference/string/basic_string/find_first_not_of/)

Find non-matching character in string (public member function )

[**find\_last\_not\_of**](http://www.cplusplus.com/reference/string/basic_string/find_last_not_of/)

Find non-matching character in string from the end (public member function )

[**substr**](http://www.cplusplus.com/reference/string/basic_string/substr/)

Generate substring (public member function )

[**compare**](http://www.cplusplus.com/reference/string/basic_string/compare/)

Compare strings (public member function )

**Non-member function overloads**

[**operator+**](http://www.cplusplus.com/reference/string/basic_string/operator+/)

Concatenate strings (function template )

[**relational operators**](http://www.cplusplus.com/reference/string/basic_string/operators/)

Relational operators for basic\_string (function template )

[**swap**](http://www.cplusplus.com/reference/string/basic_string/swap-free/)

Exchanges the values of two strings (function template )

[**operator>>**](http://www.cplusplus.com/reference/string/basic_string/operator%3E%3E/)

Extract string from stream (function template )

[**operator<<**](http://www.cplusplus.com/reference/string/basic_string/operator%3C%3C/)

Insert string into stream (function template )

[**getline**](http://www.cplusplus.com/reference/string/basic_string/getline/)

Get line from stream into string (function template )

**Member constants**

[**npos**](http://www.cplusplus.com/reference/string/basic_string/npos/)

Maximum value of size\_type (public static member constant )

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::basic\_string**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **default (1)** | explicit basic\_string (const allocator\_type& alloc = allocator\_type()); |
| **copy (2)** | basic\_string (const basic\_string& str); |
| **substring (3)** | basic\_string (const basic\_string& str, size\_type pos, size\_type len = npos,  const allocator\_type& alloc = allocator\_type()); |
| **from c-string (4)** | basic\_string (const charT\* s, const allocator\_type& alloc = allocator\_type()); |
| **from sequence (5)** | basic\_string (const charT\* s, size\_type n,  const allocator\_type& alloc = allocator\_type()); |
| **fill (6)** | basic\_string (size\_type n, charT c,  const allocator\_type& alloc = allocator\_type()); |
| **range (7)** | template <class InputIterator>  basic\_string (InputIterator first, InputIterator last,  const allocator\_type& alloc = allocator\_type()); |

Construct basic\_string object

Constructs a [basic\_string](http://www.cplusplus.com/basic_string) object, initializing its value depending on the constructor version used:

(1) empty string constructor (default constructor)

Constructs an [empty](http://www.cplusplus.com/basic_string::empty) string, with a [length](http://www.cplusplus.com/basic_string::length) of zero characters.

(2) copy constructors

Constructs a copy of *str*.

(3) substring constructor

Copies the portion of *str* that begins at the character position *pos* and spans *len* characters (or until the end of *str*, if either *str* is too short or if *len* is [basic\_string::npos](http://www.cplusplus.com/basic_string::npos)).

(4) from c-string

Copies the null-terminated character sequence (C-string) pointed by *s*.  
The [length](http://www.cplusplus.com/string::length) is determined by calling [traits\_type::length](http://www.cplusplus.com/char_traits::length)(s).

(5) from buffer

Copies the first *n* characters from the array of characters pointed by *s*.

(6) fill constructor

Fills the string with *n* consecutive copies of character *c*.

(7) range constructor

Copies the sequence of characters in the range [first,last), in the same order.

(8) initializer list

Copies each of the characters in *il*, in the same order.

(9) move contructors

Acquires the contents of *str*.  
*str* is left in an unspecified but valid state.

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

The [basic\_string](http://www.cplusplus.com/basic_string) object keeps an internal copy of *alloc*, which is used to allocate and free storage for the characters it contains throughout its lifetime.  
The copy constructor *(2)* creates an object that keeps and uses a copy of *str*'s allocator.

**Parameters**

alloc

Allocator object.  
The container keeps and uses an internal copy of this allocator.  
Member type allocator\_type is the internal allocator type used by the container, defined in [basic\_string](http://www.cplusplus.com/basic_string) as an alias of its third template parameter (Alloc).  
If allocator\_type is an instantiation of the default [allocator](http://www.cplusplus.com/allocator) (which has no state), this is not relevant.

str

Another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc), whose value is either copied or acquired.

pos

Position of the first character in *str* that is copied to the object as a substring.  
If this is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character in *str* is denoted by a value of 0 (not 1).

len

Length of the substring to be copied (if the string is shorter, as many characters as possible are copied).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of *str*.

s

Pointer to an array of characters (such as a *c-string*).

n

Number of characters to copy.

c

Character to fill the string with. Each of the *n* characters in the string will be initialized to a copy of this value.

first, last

[Input iterators](http://www.cplusplus.com/InputIterator) to the initial and final positions in a range. The range used is [first,last), which includes all the characters between *first* and *last*, including the character pointed by *first* but not the character pointed by *last*.  
The function template argument InputIterator shall be an [input iterator](http://www.cplusplus.com/InputIterator) type that points to elements of a type convertible to charT.  
If InputIterator is an integral type, the arguments are casted to the proper types so that signature (5) is used instead.

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
These objects are automatically constructed from *initializer list* declarators.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | *// string constructor*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string s0 ("Initial string");  *// constructors used in the same order as described above:*  std::string s1;  std::string s2 (s0);  std::string s3 (s0, 8, 3);  std::string s4 ("A character sequence", 6);  std::string s5 ("Another character sequence");  std::string s6 (10, 'x');  std::string s7a (10, 42);  std::string s7b (s0.begin(), s0.begin()+7);  std::cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;  std::cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6: " << s6;  std::cout << "\ns7a: " << s7a << "\ns7b: " << s7b << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| s1:  s2: Initial string  s3: str  s4: A char  s5: Another character sequence  s6: xxxxxxxxxx  s7a: \*\*\*\*\*\*\*\*\*\*  s7b: Initial |

**Complexity**

* [C++98](javascript:switch3.select(1))
* [C++11](javascript:switch3.select(2))

Unspecified.

**Iterator validity**

The *move constructors (9)* may invalidate iterators, pointers and references related to *str*.

**Data races**

The *move constructors (9)* modify *str*.

**Exception safety**

For the *move constructors (9)*, if the constructed object uses the same allocator as *str* and the proper allocator's constructor is not-throwing, the function does not throw exceptions.  
In all other cases, there are no effects in case an exception is thrown (strong guarantee).  
  
If s is a null pointer, if n == npos, or if the range specified by [first,last) is not valid, it causes *undefined behavior*.  
  
If *pos* is greater then *str*'s [length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function fails when attempting to allocate storage.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::~basic\_string**

~basic\_string();

String destructor

Destroys the [basic\_string](http://www.cplusplus.com/basic_string) object.  
  
This deallocates all the storage [capacity](http://www.cplusplus.com/basic_string::capacity) allocated by the [basic\_string](http://www.cplusplus.com/basic_string) using its [allocator](http://www.cplusplus.com/basic_string::get_allocator).

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

All iterators, pointers and references are invalidated.

**Data races**

The object is modified.

**Exception safety**

**No-throw guarantee:** never throws exceptions.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::operator=**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | basic\_string& operator= (const basic\_string& str); |
| **c-string (2)** | basic\_string& operator= (const charT\* s); |
| **character (3)** | basic\_string& operator= (charT c); |

String assignment

Assigns a new value to the string, replacing its current contents.  
  
(See member function [assign](http://www.cplusplus.com/basic_string::assign) for additional assignment options).

**Parameters**

str

A [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc), whose value is either copied *(1)* or moved *(5)* if different from \*this (if moved, *str* is left in an unspecified but valid state).

s

Pointer to a null-terminated sequence of characters.  
The sequence is copied as the new value for the string.  
The [length](http://www.cplusplus.com/string::length) is determined by calling [traits\_type::length](http://www.cplusplus.com/char_traits::length)(s).

c

A character.  
The string value is set to a single copy of this character (the [string length](http://www.cplusplus.com/basic_string::length) becomes 1).

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
These objects are automatically constructed from *initializer list* declarators.  
The characters are copied, in the same order.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Return Value**

\*this

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | *// string assigning*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str1, str2, str3;  str1 = "Test string: "; *// c-string*  str2 = 'x'; *// single character*  str3 = str1 + str2; *// string*  std::cout << str3 << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Test string: x |

**Complexity**

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Unspecified.

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.  
The *move assignment (5)* modifies *str*.

**Exception safety**

For the *move assignment (5)*, the function does not throw exceptions (no-throw guarantee).  
In all other cases, there are no effects in case an exception is thrown (strong guarantee).  
  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::begin**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

iterator begin();

const\_iterator begin() const;

Return iterator to beginning

Returns an iterator pointing to the first character of the string.

**Parameters**

none

**Return Value**

An iterator to the beginning of the string.  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const\_iterator. Otherwise, it returns an iterator.  
  
Member types iterator and const\_iterator are [random access iterator](http://www.cplusplus.com/RandomAccessIterator) types (pointing to a character and to a const character, respectively).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::begin/end*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  *for* ( std::string::iterator it=str.begin(); it!=str.end(); ++it)  std::cout << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Test string |

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

**Complexity**

Unspecified.

**Iterator validity**

Generally, no changes.  
On some implementations, the non-const version may invalidate all iterators, pointers and references on the first access to string characters after the object has been constructed or modified.

**Data races**

The object is accessed, and in some implementations, the non-const version modifies it on the first access to string characters after the object has been constructed or modified.  
The iterator returned can be used to access or modify characters.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::end**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

iterator end();

const\_iterator end() const;

Return iterator to end

Returns an iterator pointing to the *past-the-end* character of the string.  
  
The *past-the-end* character is a theoretical character that would follow the last character in the string. It shall not be dereferenced.  
  
Because the ranges used by functions of the standard library do not include the element pointed by their closing iterator, this function is often used in combination with [basic\_string::begin](http://www.cplusplus.com/basic_string::begin) to specify a range including all the characters in the string.  
  
If the object is an [empty string](http://www.cplusplus.com/basic_string::empty), this function returns the same as [basic\_string::begin](http://www.cplusplus.com/basic_string::begin).

**Parameters**

none

**Return Value**

An iterator to the past-the-end of the string.  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const\_iterator. Otherwise, it returns an iterator.  
  
Member types iterator and const\_iterator are [random access iterator](http://www.cplusplus.com/RandomAccessIterator) types (pointing to a character and to a const character, respectively).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::begin/end*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  *for* ( std::string::iterator it=str.begin(); it!=str.end(); ++it)  std::cout << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Test string |

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

**Complexity**

Unspecified.

**Iterator validity**

Generally, no changes.  
On some implementations, the non-const version may invalidate all iterators, pointers and references on the first access to string characters after the object has been constructed or modified.

**Data races**

The object is accessed, and in some implementations, the non-const version modifies it on the first access to string characters after the object has been constructed or modified.  
The iterator returned can be used to access or modify characters.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::rbegin**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

reverse\_iterator rbegin();

const\_reverse\_iterator rbegin() const;

Return reverse iterator to reverse beginning

Returns a *reverse iterator* pointing to the last character of the string (i.e., its *reverse beginning*).  
  
*Reverse iterators* iterate backwards: increasing them moves them towards the beginning of the string.  
  
rbegin points to the character right before the one that would be pointed to by member [end](http://www.cplusplus.com/basic_string::end).

**Parameters**

none

**Return Value**

A reverse iterator to the *reverse beginning* of the string.  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const\_iterator. Otherwise, it returns an iterator.  
  
Member types reverse\_iterator and const\_reverse\_iterator are reverse [random access iterator](http://www.cplusplus.com/RandomAccessIterator) types (pointing to a character and to a const character, respectively).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | *// string::rbegin/rend*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("now step live...");  *for* (std::string::reverse\_iterator rit=str.rbegin(); rit!=str.rend(); ++rit)  std::cout << \*rit;  *return* 0;  } |

This code prints out the reversed content of a string character by character using a reverse iterator that iterates between rbegin and rend. Notice how even though the reverse iterator is increased, the iteration goes backwards through the string (this is a feature of reverse iterators).  
The actual output is:

|  |
| --- |
| ...evil pets won |

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

**Complexity**

Unspecified.

**Iterator validity**

Generally, no changes.  
On some implementations, the non-const version may invalidate all iterators, pointers and references on the first access to string characters after the object has been constructed or modified.

**Data races**

The object is accessed, and in some implementations, the non-const version modifies it on the first access to string characters after the object has been constructed or modified.  
The iterator returned can be used to access or modify characters.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::rend**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

reverse\_iterator rend();

const\_reverse\_iterator rend() const;

Return reverse iterator to reverse end

Returns a *reverse iterator* pointing to the theoretical element preceding the first character of the string (which is considered its *reverse end*).  
  
The range between [basic\_string::rbegin](http://www.cplusplus.com/basic_string::rbegin) and basic\_string::rend contains all the characters of the [basic\_string](http://www.cplusplus.com/basic_string) (in reverse order).

**Parameters**

none

**Return Value**

A reverse iterator to the *reverse end* of the string.  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const\_iterator. Otherwise, it returns an iterator.  
  
Member types reverse\_iterator and const\_reverse\_iterator are reverse [random access iterator](http://www.cplusplus.com/RandomAccessIterator) types (pointing to a character and to a const character, respectively).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | *// string::rbegin/rend*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("now step live...");  *for* (std::string::reverse\_iterator rit=str.rbegin(); rit!=str.rend(); ++rit)  std::cout << \*rit;  *return* 0;  } |

This code prints out the reversed content of a string character by character using a reverse iterator that iterates between rbegin and rend. Notice how even though the reverse iterator is increased, the iteration goes backwards through the string (this is a feature of reverse iterators).  
The actual output is:

|  |
| --- |
| ...evil pets won |

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

**Complexity**

Unspecified.

**Iterator validity**

Generally, no changes.  
On some implementations, the non-const version may invalidate all iterators, pointers and references on the first access to string characters after the object has been constructed or modified.

**Data races**

The object is accessed, and in some implementations, the non-const version modifies it on the first access to string characters after the object has been constructed or modified.  
The iterator returned can be used to access or modify characters.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::cbegin**

const\_iterator cbegin() const noexcept;

Return const\_iterator to beginning

Returns a const\_iterator pointing to the first character of the string.  
  
A const\_iterator is an iterator that points to const content. This iterator can be increased and decreased (unless it is itself also const), just like the iterator returned by [basic\_string::begin](http://www.cplusplus.com/basic_string::begin), but it cannot be used to modify the contents it points to, even if the [basic\_string](http://www.cplusplus.com/basic_string) object is not itself const.

**Parameters**

none

**Return Value**

A const\_iterator to the beginning of the string.  
  
Member type const\_iterator is a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) type that points to a const character.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::cbegin/cend*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Lorem ipsum");  *for* (*auto* it=str.cbegin(); it!=str.cend(); ++it)  std::cout << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Lorem ipsum |

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::cend**

const\_iterator cend() const noexcept;

Return const\_iterator to end

Returns a const\_iterator pointing to the *past-the-end* character of the string.  
  
A const\_iterator is an iterator that points to const content. This iterator can be increased and decreased (unless it is itself also const), just like the iterator returned by [basic\_string::end](http://www.cplusplus.com/basic_string::end), but it cannot be used to modify the contents it points to, even if the [basic\_string](http://www.cplusplus.com/basic_string) object is not itself const.  
  
If the object is an [empty string](http://www.cplusplus.com/basic_string::empty), this function returns the same as [basic\_string::cbegin](http://www.cplusplus.com/basic_string::cbegin).  
  
The value returned shall not be dereferenced.

**Parameters**

none

**Return Value**

A const\_iterator to the past-the-end of the string.  
  
Member type const\_iterator is a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) type that points to a const character.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::cbegin/cend*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Lorem ipsum");  *for* (*auto* it=str.cbegin(); it!=str.cend(); ++it)  std::cout << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Lorem ipsum |

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::crbegin**

const\_reverse\_iterator crbegin() const noexcept;

Return const\_reverse\_iterator to reverse beginning

Returns a const\_reverse\_iterator pointing to the last character of the string (i.e., its *reverse beginning*).

**Parameters**

none

**Return Value**

A const\_reverse\_iterator to the *reverse beginning* of the string.  
  
Member type const\_reverse\_iterator is a reverse [random access iterator](http://www.cplusplus.com/RandomAccessIterator) type that points to a const character.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::crbegin/crend*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("lorem ipsum");  *for* (*auto* rit=str.crbegin(); rit!=str.crend(); ++rit)  std::cout << \*rit;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| muspi merol |

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::crend**

const\_reverse\_iterator crend() const noexcept;

Return const\_reverse\_iterator to reverse end

Returns a const\_reverse\_iterator pointing to the theoretical character preceding the first character of the string (which is considered its *reverse end*).

**Parameters**

none

**Return Value**

A const\_reverse\_iterator to the *reverse end* of the string.  
  
Member type const\_reverse\_iterator is a reverse [random access iterator](http://www.cplusplus.com/RandomAccessIterator) type that points to a const character.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::crbegin/crend*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("lorem ipsum");  *for* (*auto* rit=str.crbegin(); rit!=str.crend(); ++rit)  std::cout << \*rit;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| muspi merol |

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::size**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

size\_type size() const;

Return size

Returns the length of the string, in terms of number of characters.  
  
This is the number of actual characters that conform the contents of the [basic\_string](http://www.cplusplus.com/basic_string), which is not necessarily equal to its storage [capacity](http://www.cplusplus.com/basic_string::capacity).  
  
Both basic\_string::size and [basic\_string::length](http://www.cplusplus.com/basic_string::length) are synonyms and return the same value.

**Parameters**

none

**Return Value**

The number of characters in the string.  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | *// string::size*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  std::cout << "The size of str is " << str.size() << " characters.\n";  *return* 0;  } |

Output:

|  |
| --- |
| The size of str is 11 characters |

**Complexity**

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Unspecified.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::length**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

size\_type length() const;

Return length of string

Returns the length of the string, in terms of number of characters.  
  
This is the number of actual characters that conform the contents of the [basic\_string](http://www.cplusplus.com/basic_string), which is not necessarily equal to its storage [capacity](http://www.cplusplus.com/basic_string::capacity).  
  
Both [basic\_string::size](http://www.cplusplus.com/basic_string::size) and basic\_string::length are synonyms and return the same value.

**Parameters**

none

**Return Value**

The number of characters in the string.  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | *// string::length*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  std::cout << "The size of str is " << str.length() << " characters.\n";  *return* 0;  } |

Output:

|  |
| --- |
| The size of str is 11 characters |

**Complexity**

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Unspecified.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::max\_size**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

size\_type max\_size() const;

Return maximum size

Returns the maximum length the [basic\_string](http://www.cplusplus.com/basic_string) can reach.  
  
This is the maximum potential [length](http://www.cplusplus.com/basic_string::length) the string can reach due to known system or library implementation limitations, but the object is not guaranteed to be able to reach that length: it can still fail to allocate storage at any point before that length is reached.

**Parameters**

none

**Return Value**

The maximum length the [basic\_string](http://www.cplusplus.com/basic_string) can reach.  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// comparing size, length, capacity and max\_size*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  std::cout << "size: " << str.size() << "\n";  std::cout << "length: " << str.length() << "\n";  std::cout << "capacity: " << str.capacity() << "\n";  std::cout << "max\_size: " << str.max\_size() << "\n";  *return* 0;  } |

A possible output for this program could be:

|  |
| --- |
| size: 11  length: 11  capacity: 15  max\_size: 4294967291 |

**Complexity**

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::resize**

void resize (size\_type n);

void resize (size\_type n, charT c);

Resize string

Resizes the string to a [length](http://www.cplusplus.com/basic_string::length) of *n* characters.  
  
If *n* is smaller than the current [string length](http://www.cplusplus.com/basic_string::length), the current value is shortened to its first *n* character, removing the characters beyond the *n*th.  
  
If *n* is greater than the current [string length](http://www.cplusplus.com/basic_string::length), the current content is extended by inserting at the end as many characters as needed to reach a size of *n*. If *c* is specified, the new elements are initialized as copies of *c*, otherwise, they are *value-initialized characters* (null characters).

**Parameters**

n

New [string length](http://www.cplusplus.com/basic_string::length), expressed in number of characters.  
Member type size\_type is an unsigned integral type.

c

Character used to fill the new character space added to the string (in case the string is expanded).  
Member type size\_type is an unsigned integral type.

**Return Value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | *// resizing string*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("I like to code in C");  std::cout << str << '\n';  std::string::size\_type sz = str.size();  str.resize (sz+2,'+');  std::cout << str << '\n';  str.resize (14);  std::cout << str << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| I like to code in C  I like to code in C++  I like to code |

**Complexity**

Unspecified, but generally up to linear in the new [string length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If *n* is greater than [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::capacity**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

size\_type capacity() const;

Return size of allocated storage

Returns the size of the storage space currently allocated for the [basic\_string](http://www.cplusplus.com/basic_string), expressed in terms of characters.  
  
This *capacity* is not necessarily equal to the [string length](http://www.cplusplus.com/basic_string::length). It can be equal or greater, with the extra space allowing the object to optimize its operations when new characters are added to the [basic\_string](http://www.cplusplus.com/basic_string).  
  
Notice that this *capacity* does not suppose a limit on the [length](http://www.cplusplus.com/basic_string::length) of the [basic\_string](http://www.cplusplus.com/basic_string). When this *capacity* is exhausted and more is needed, it is automatically expanded by the object (reallocating it storage space). The theoretical limit on the [length](http://www.cplusplus.com/basic_string::length) of a [basic\_string](http://www.cplusplus.com/basic_string) is given by member [max\_size](http://www.cplusplus.com/basic_string::max_size).  
  
The *capacity* of a [basic\_string](http://www.cplusplus.com/basic_string) can be altered any time the object is modified, even if this modification implies a reduction in size or if the capacity has not been exhausted (this is in contrast with the guarantees given to *capacity* in [vector containers](http://www.cplusplus.com/vector)).  
  
The *capacity* of a [basic\_string](http://www.cplusplus.com/basic_string) can be explicitly altered by calling member [reserve](http://www.cplusplus.com/basic_string::reserve).

**Parameters**

none

**Return Value**

The size of the storage capacity currently allocated for the [basic\_string](http://www.cplusplus.com/basic_string).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// comparing size, length, capacity and max\_size*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  std::cout << "size: " << str.size() << "\n";  std::cout << "length: " << str.length() << "\n";  std::cout << "capacity: " << str.capacity() << "\n";  std::cout << "max\_size: " << str.max\_size() << "\n";  *return* 0;  } |

A possible output for this program could be:

|  |
| --- |
| size: 11  length: 11  capacity: 15  max\_size: 429496729 |

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::reserve**

void reserve (size\_type n = 0);

Request a change in capacity

Requests that the [string capacity](http://www.cplusplus.com/basic_string::capacity) be adapted to a planned change in [size](http://www.cplusplus.com/basic_string::size) to a [length](http://www.cplusplus.com/basic_string::length) of up to *n* characters.  
  
If *n* is greater than the current [string capacity](http://www.cplusplus.com/basic_string::capacity), the function causes the container to increase its [capacity](http://www.cplusplus.com/basic_string::capacity) to *n* characters (or greater).  
  
In all other cases, it is taken as a non-binding request to shrink the [string capacity](http://www.cplusplus.com/basic_string::capacity): the container implementation is free to optimize otherwise and leave the [basic\_string](http://www.cplusplus.com/basic_string) with a [capacity](http://www.cplusplus.com/basic_string::capacity) greater than *n*.  
  
This function has no effect on the [string length](http://www.cplusplus.com/basic_string::length) and cannot alter its content.

**Parameters**

n

Planned [length](http://www.cplusplus.com/basic_string::length) for the [basic\_string](http://www.cplusplus.com/basic_string).  
Note that the resulting [string capacity](http://www.cplusplus.com/basic_string::capacity) may be equal or greater than *n*.  
Member type size\_type is an unsigned integral type.

**Return Value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | *// string::reserve*  *#include <iostream>*  *#include <fstream>*  *#include <string>*  *int* main ()  {  std::string str;  std::ifstream file ("test.txt",std::ios::in|std::ios::ate);  *if* (file) {  std::ifstream::streampos filesize = file.tellg();  str.reserve(filesize);  file.seekg(0);  *while* (!file.eof())  {  str += file.get();  }  std::cout << str;  }  *return* 0;  } |

This example reserves enough capacity in the [basic\_string](http://www.cplusplus.com/basic_string) object to store an entire file, which is then read character by character. By reserving a [capacity](http://www.cplusplus.com/basic_string::capacity) for the [basic\_string](http://www.cplusplus.com/basic_string) of at least the size of the entire file, we try to avoid all the automatic reallocations that the object *str* could suffer each time that inserting a new character would make its [length](http://www.cplusplus.com/basic_string::length) surpass its [capacity](http://www.cplusplus.com/basic_string::capacity).

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If *n* is greater than the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::clear**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

void clear();

Clear string

Erases the contents of the [basic\_string](http://www.cplusplus.com/basic_string), which becomes an [empty string](http://www.cplusplus.com/basic_string::empty) (with a [length](http://www.cplusplus.com/basic_string::length) of 0 characters).

**Parameters**

none

**Return value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | *// string::clear*  *#include <iostream>*  *#include <string>*  *int* main ()  {  *char* c;  std::string str;  std::cout << "Please type some lines of text. Enter a dot (.) to finish:\n";  *do* {  c = std::cin.get();  str += c;  *if* (c=='\n')  {  std::cout << str;  str.clear();  }  } *while* (c!='.');  *return* 0;  } |

This program repeats every line introduced by the user until a the line contains a dot ('.'). Every newline character ('\n') triggers the repetition of the line and the clearing of the current string content.

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::empty**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

bool empty() const;

Test whether string is empty

Returns whether the [basic\_string](http://www.cplusplus.com/basic_string) is empty (i.e. whether its [length](http://www.cplusplus.com/basic_string::length) is 0).  
  
This function does not modify the value of the string in any way. To clear the content of a [basic\_string](http://www.cplusplus.com/basic_string), see [basic\_string::clear](http://www.cplusplus.com/basic_string::clear).

**Parameters**

none

**Return Value**

true if the [string length](http://www.cplusplus.com/basic_string::length) is 0, false otherwise.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | *// string::empty*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string content;  std::string line;  std::cout << "Please introduce a text. Enter an empty line to finish:\n";  *do* {  getline(std::cin,line);  content += line + '\n';  } *while* (!line.empty());  std::cout << "The text you introduced was:\n" << content;  *return* 0;  } |

This program reads the user input line by line and stores it into string content until an empty line is introduced.

**Complexity**

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::shrink\_to\_fit**

void shrink\_to\_fit();

Shrink to fit

Requests the [basic\_string](http://www.cplusplus.com/basic_string) to reduce its [capacity](http://www.cplusplus.com/basic_string::capacity) to fit its [size](http://www.cplusplus.com/basic_string::size).  
  
The request is non-binding, and the container implementation is free to optimize otherwise and leave the [basic\_string](http://www.cplusplus.com/basic_string) with a [capacity](http://www.cplusplus.com/basic_string::capacity) greater than its [size](http://www.cplusplus.com/basic_string::size).  
  
This function has no effect on the [string length](http://www.cplusplus.com/basic_string::length) and cannot alter its content.

**Parameters**

none

**Return value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | *// string::shrink\_to\_fit*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str (100,'x');  std::cout << "1. capacity of str: " << str.capacity() << '\n';  str.resize(10);  std::cout << "2. capacity of str: " << str.capacity() << '\n';  str.shrink\_to\_fit();  std::cout << "3. capacity of str: " << str.capacity() << '\n';  *return* 0;  } |

Possible output:

|  |
| --- |
| 1. capacity of str: 100  2. capacity of str: 100  3. capacity of str: 10 |

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::operator[]**

reference operator[] (size\_type pos);

const\_reference operator[] (size\_type pos) const;

Get character of string

Returns a reference to the character at position *pos* in the [basic\_string](http://www.cplusplus.com/basic_string).

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

If *pos* is equal to the [string length](http://www.cplusplus.com/basic_string::length), the function returns a reference to a null character (charT()).

**Parameters**

pos

Value with the position of a character within the string.  
Note: The first character in a [basic\_string](http://www.cplusplus.com/basic_string) is denoted by a value of 0 (not 1).  
Member type size\_type is an unsigned integral type.

**Return value**

The character at the specified position in the string.  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const\_reference. Otherwise, it returns a reference.  
  
Member types reference and const\_reference are the reference types to the characters in the container; They shall be aliases of charT& and const charT& respectively.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::operator[]*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  *for* (*int* i=0; i<str.length(); ++i)  {  std::cout << str[i];  }  *return* 0;  } |

This code prints out the content of a string character by character using the offset operator on *str*:

|  |
| --- |
| Test string |

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

**Complexity**

Unspecified.

**Iterator validity**

Generally, no changes.  
On some implementations, the non-const version may invalidate all iterators, pointers and references on the first access to string characters after the object has been constructed or modified.

**Data races**

The object is accessed, and in some implementations, the non-const version modifies it on the first access to string characters after the object has been constructed or modified.  
The reference returned can be used to access or modify characters.

**Exception safety**

If *pos* is not greater than the [string length](http://www.cplusplus.com/basic_string::length), the function never throws exceptions (no-throw guarantee).  
Otherwise, it causes *undefined behavior*.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::at**

reference at (size\_type pos);

const\_reference at (size\_type pos) const;

Get character of string

Returns a reference to the character at position *pos* in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
The function automatically checks whether *pos* is the valid position of a character in the string (i.e., whether *pos* is less than the [string length](http://www.cplusplus.com/basic_string::length)), throwing an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception if it is not.

**Parameters**

pos

Value with the position of a character within the string.  
Note: The first character in a [basic\_string](http://www.cplusplus.com/basic_string) is denoted by a value of 0 (not 1).  
If it is not the position of a character, an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
Member type size\_type is an unsigned integral type.

**Return value**

The character at the specified position in the string.  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const\_reference. Otherwise, it returns a reference.  
  
Member types reference and const\_reference are the reference types to the characters in the container; They shall be aliases of charT& and const charT& respectively.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::at*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Test string");  *for* (*unsigned* i=0; i<str.length(); ++i)  {  std::cout << str.at(i);  }  *return* 0;  } |

This code prints out the content of a string character by character using the at member function:

|  |
| --- |
| Test string |

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

**Complexity**

Unspecified.

**Iterator validity**

Generally, no changes.  
On some implementations, the non-const version may invalidate all iterators, pointers and references on the first access to string characters after the object has been constructed or modified.

**Data races**

The object is accessed, and in some implementations, the non-const version modifies it on the first access to string characters after the object has been constructed or modified.  
The reference returned can be used to access or modify characters.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If *pos* is not less than the [string length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::back**

charT& back();

const charT& back() const;

Access last character

Returns a reference to the last character of the [basic\_string](http://www.cplusplus.com/basic_string).  
  
This function shall not be called on [empty strings](http://www.cplusplus.com/basic_string::empty).

**Parameters**

none

**Return value**

A reference to the last character in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const charT&. Otherwise, it returns a charT&.  
  
charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | *// string::back*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("hello world.");  str.back() = '!';  std::cout << str << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| hello world! |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed (neither the const nor the non-const versions modify the container).  
The reference returned can be used to access or modify characters. Concurrently accessing or modifying different characters is safe.

**Exception safety**

If the [basic\_string](http://www.cplusplus.com/basic_string) is not [empty](http://www.cplusplus.com/basic_string::empty), the function never throws exceptions (no-throw guarantee).  
Otherwise, it causes *undefined behavior*.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::front**

charT& front();

const charT& front() const;

Access first character

Returns a reference to the first character of the [basic\_string](http://www.cplusplus.com/basic_string).  
  
Unlike member [basic\_string::begin](http://www.cplusplus.com/basic_string::begin), which returns an iterator to this same character, this function returns a direct reference.  
  
This function shall not be called on [empty strings](http://www.cplusplus.com/basic_string::empty).

**Parameters**

none

**Return value**

A reference to the first character in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If the [basic\_string](http://www.cplusplus.com/basic_string) object is const-qualified, the function returns a const charT&. Otherwise, it returns a charT&.  
  
charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | *// string::front*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("test string");  str.front() = 'T';  std::cout << str << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Test string |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed (neither the const nor the non-const versions modify the container).  
The reference returned can be used to access or modify characters. Concurrently accessing or modifying different characters is safe.

**Exception safety**

If the [basic\_string](http://www.cplusplus.com/basic_string) is not [empty](http://www.cplusplus.com/basic_string::empty), the function never throws exceptions (no-throw guarantee).  
Otherwise, it causes *undefined behavior*.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::operator+=**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | basic\_string& operator+= (const basic\_string& str); |
| **c-string (2)** | basic\_string& operator+= (const charT\* s); |
| **character (3)** | basic\_string& operator+= (charT c); |

Append to string

Extends the [basic\_string](http://www.cplusplus.com/basic_string) by appending additional characters at the end of its current value:  
  
(See member function [append](http://www.cplusplus.com/basic_string::append) for additional appending options).

**Parameters**

str

A [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc), whose value is copied at the end.

s

Pointer to a null-terminated sequence of characters.  
The sequence is copied at the end of the string.  
The [length](http://www.cplusplus.com/string::length) is determined by calling [traits\_type::length](http://www.cplusplus.com/char_traits::length)(s).

c

A character, which is appended to the current value of the string.

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
These objects are automatically constructed from *initializer list* declarators.  
The characters are appended to the string, in the same order.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Return Value**

\*this

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | *// string::operator+=*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string name ("John");  std::string family ("Smith");  name += " K. "; *// c-string*  name += family; *// string*  name += '\n'; *// character*  std::cout << name;  *return* 0;  } |

Output:

|  |
| --- |
| John K. Smith |

**Complexity**

Unspecified, but generally up to linear in the new [string length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::append**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | basic\_string& append (const basic\_string& str); |
| **substring (2)** | basic\_string& append (const basic\_string& str, size\_type subpos, size\_type sublen); |
| **c-string (3)** | basic\_string& append (const charT\* s); |
| **buffer (4)** | basic\_string& append (const charT\* s, size\_type n); |
| **fill (5)** | basic\_string& append (size\_type n, charT c); |
| **range (6)** | template <class InputIterator>  basic\_string& append (InputIterator first, InputIterator last); |

Append to string

Extends the [basic\_string](http://www.cplusplus.com/basic_string) by appending additional characters at the end of its current value:

(1) string

Appends a copy of *str*.

(2) substring

Appends a copy of a substring of *str*. The substring is the portion of *str* that begins at the character position *subpos* and spans *sublen* characters (or until the end of *str*, if either *str* is too short or if *sublen* is [basic\_string::npos](http://www.cplusplus.com/basic_string::npos)).

(3) c-string

Appends a copy of the string formed by the null-terminated character sequence (C-string) pointed by *s*.  
The length of this character sequence is determined by calling [traits\_type::length](http://www.cplusplus.com/char_traits::length)(s).

(4) buffer

Appends a copy of the first *n* characters in the array of characters pointed by *s*.

(5) fill

Appends *n* consecutive copies of character *c*.

(6) range

Appends a copy of the sequence of characters in the range [first,last), in the same order.

(7) initializer list

Appends a copy of each of the characters in *il*, in the same order.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc), whose value is appended.

subpos

Position of the first character in *str* that is copied to the object as a substring.  
If this is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character in *str* is denoted by a value of 0 (not 1).

sublen

Length of the substring to be copied (if the string is shorter, as many characters as possible are copied).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of *str*.

s

Pointer to an array of characters (such as a *c-string*).

n

Number of characters to copy.

c

Character value, repeated *n* times.

first, last

[Input iterators](http://www.cplusplus.com/InputIterator) to the initial and final positions in a range. The range used is [first,last), which includes all the characters between *first* and *last*, including the character pointed by *first* but not the character pointed by *last*.  
The function template argument InputIterator shall be an [input iterator](http://www.cplusplus.com/InputIterator) type that points to elements of a type convertible to charT.  
If InputIterator is an integral type, the arguments are casted to the proper types so that signature (5) is used instead.

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
These objects are automatically constructed from *initializer list* declarators.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

\*this

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | *// appending to string*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str;  std::string str2="Writing ";  std::string str3="print 10 and then 5 more";  *// used in the same order as described above:*  str.append(str2); *// "Writing "*  str.append(str3,6,3); *// "10 "*  str.append("dots are cool",5); *// "dots "*  str.append("here: "); *// "here: "*  str.append(10u,'.'); *// ".........."*  str.append(str3.begin()+8,str3.end()); *// " and then 5 more"*  str.append<*int*>(5,0x2E); *// "....."*  std::cout << str << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Writing 10 dots here: .......... and then 5 more..... |

**Complexity**

Unspecified, but generally up to linear in the new [string length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If s does not point to an array long enough, or if the range specified by [first,last) is not valid, it causes *undefined behavior*.  
  
If *subpos* is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::push\_back**

void push\_back (charT c);

Append character to string

Appends character *c* to the end of the [basic\_string](http://www.cplusplus.com/basic_string), increasing its [length](http://www.cplusplus.com/basic_string::length) by one.

**Parameters**

c

Character added to the [basic\_string](http://www.cplusplus.com/basic_string).  
charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Return value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | *// string::push\_back*  *#include <iostream>*  *#include <fstream>*  *#include <string>*  *int* main ()  {  std::string str;  std::ifstream file ("test.txt",std::ios::in);  *if* (file) {  *while* (!file.eof()) str.push\_back(file.get());  }  std::cout << str << '\n';  *return* 0;  } |

This example reads an entire file character by character, appending each character to a string object using push\_back.

**Complexity**

Unspecified; Generally amortized constant, but up to linear in the new [string length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::assign**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | basic\_string& assign (const basic\_string& str); |
| **substring (2)** | basic\_string& assign (const basic\_string& str, size\_type subpos, size\_type sublen); |
| **c-string (3)** | basic\_string& assign (const charT\* s); |
| **buffer (4)** | basic\_string& assign (const charT\* s, size\_type n); |
| **fill (5)** | basic\_string& assign (size\_type n, charT c); |
| **range (6)** | template <class InputIterator>  basic\_string& assign (InputIterator first, InputIterator last); |

Assign content to string

Assigns a new value to the string, replacing its current contents.

(1) string

Copies *str*.

(2) substring

Copies the portion of *str* that begins at the character position *subpos* and spans *sublen* characters (or until the end of *str*, if either *str* is too short or if *sublen* is [basic\_string::npos](http://www.cplusplus.com/basic_string::npos)).

(3) c-string

Copies the null-terminated character sequence (C-string) pointed by *s*.  
The [length](http://www.cplusplus.com/basic_string::length) is determined by calling [traits\_type::length](http://www.cplusplus.com/char_traits::length)(s).

(4) buffer

Copies the first *n* characters from the array of characters pointed by *s*.

(5) fill

Replaces the current value by *n* consecutive copies of character *c*.

(6) range

Copies the sequence of characters in the range [first,last), in the same order.

(7) initializer list

Copies each of the characters in *il*, in the same order.

(8) move

Acquires the contents of *str*.  
*str* is left in an unspecified but valid state.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc), whose value is either copied or moved.

subpos

Position of the first character in *str* that is copied to the object as a substring.  
If this is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character in *str* is denoted by a value of 0 (not 1).

sublen

Length of the substring to be copied (if the string is shorter, as many characters as possible are copied).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of *str*.

s

Pointer to an array of characters (such as a *c-string*).

n

Number of characters to copy.

c

Character value, repeated *n* times.

first, last

[Input iterators](http://www.cplusplus.com/InputIterator) to the initial and final positions in a range. The range used is [first,last), which includes all the characters between *first* and *last*, including the character pointed by *first* but not the character pointed by *last*.  
The function template argument InputIterator shall be an [input iterator](http://www.cplusplus.com/InputIterator) type that points to elements of a type convertible to charT.  
If InputIterator is an integral type, the arguments are casted to the proper types so that signature (5) is used instead.

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
These objects are automatically constructed from *initializer list* declarators.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

\*this

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 | *// string::assign*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str;  std::string base="The quick brown fox jumps over a lazy dog.";  *// used in the same order as described above:*  str.assign(base);  std::cout << str << '\n';  str.assign(base,10,9);  std::cout << str << '\n'; *// "brown fox"*  str.assign("pangrams are cool",7);  std::cout << str << '\n'; *// "pangram"*  str.assign("c-string");  std::cout << str << '\n'; *// "c-string"*  str.assign(10,'\*');  std::cout << str << '\n'; *// "\*\*\*\*\*\*\*\*\*\*"*  str.assign<*int*>(10,0x2D);  std::cout << str << '\n'; *// "----------"*  str.assign(base.begin()+16,base.end()-12);  std::cout << str << '\n'; *// "fox jumps over"*  *return* 0;  } |

Output:

|  |
| --- |
| The quick brown fox jumps over a lazy dog.  brown fox  pangram  c-string  \*\*\*\*\*\*\*\*\*\*  ----------  fox jumps over |

**Complexity**

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Unspecified.

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.  
The *move assign* form *(8)*, modifies *str*.

**Exception safety**

For the *move assign (8)*, the function does not throw exceptions (no-throw guarantee).  
In all other cases, there are no effects in case an exception is thrown (strong guarantee).  
  
If s does not point to an array long enough, or if the range specified by [first,last) is not valid, it causes *undefined behavior*.  
  
If *subpos* is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::insert**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | basic\_string& insert (size\_type pos, const basic\_string& str); |
| **substring (2)** | basic\_string& insert (size\_type pos, const basic\_string& str,  size\_type subpos, size\_type sublen); |
| **c-string (3)** | basic\_string& insert (size\_type pos, const charT\* s); |
| **buffer (4)** | basic\_string& insert (size\_type pos, const charT\* s, size\_type n); |
| **fill (5)** | basic\_string& insert (size\_type pos, size\_type n, charT c);  void insert (iterator p, size\_type n, charT c); |
| **single character (6)** | iterator insert (iterator p, charT c); |
| **range (7)** | template <class InputIterator>  void insert (iterator p, InputIterator first, InputIterator last); |

Insert into string

Inserts additional characters into the [basic\_string](http://www.cplusplus.com/basic_string) right before the character indicated by *pos* (or *p*):

(1) string

Inserts a copy of *str*.

(2) substring

Inserts a copy of a substring of *str*. The substring is the portion of *str* that begins at the character position *subpos* and spans *sublen* characters (or until the end of *str*, if either *str* is too short or if *sublen* is [npos](http://www.cplusplus.com/basic_string::npos)).

(3) c-string

Inserts a copy of the string formed by the null-terminated character sequence (C-string) pointed by *s*.  
The length of this character sequence is determined by calling [traits\_type::length](http://www.cplusplus.com/char_traits::length)(s).

(4) buffer

Inserts a copy of the first *n* characters in the array of characters pointed by *s*.

(5) fill

Inserts *n* consecutive copies of character *c*.

(6) single character

Inserts character *c*.

(7) range

Inserts a copy of the sequence of characters in the range [first,last), in the same order.

(8) initializer list

Inserts a copy of each of the characters in *il*, in the same order.

**Parameters**

pos

Insertion point: The new contents are inserted before the character at position *pos*.  
If this is greater than the object's [length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character is denoted by a value of 0 (not 1).

str

Another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc).

subpos

Position of the first character in *str* that is inserted into the object as a substring.  
If this is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character in *str* is denoted by a value of 0 (not 1).

sublen

Length of the substring to be copied (if the string is shorter, as many characters as possible are copied).  
A value of [npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of *str*.

s

Pointer to an array of characters (such as a *c-string*).

n

Number of characters to insert.

c

Character value.

p

Iterator pointing to the insertion point: The new contents are inserted before the character pointed by *p*.  
iterator is a member type, defined as a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) type that points to characters of the [basic\_string](http://www.cplusplus.com/basic_string).

first, last

[Input iterators](http://www.cplusplus.com/InputIterator) to the initial and final positions in a range. The range used is [first,last), which includes all the characters between *first* and *last*, including the character pointed by *first* but not the character pointed by *last*.  
The function template argument InputIterator shall be an [input iterator](http://www.cplusplus.com/InputIterator) type that points to elements of a type convertible to charT.

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
These objects are automatically constructed from *initializer list* declarators.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return value**

The signatures returning a reference to [basic\_string](http://www.cplusplus.com/basic_string), return \*this.  
Those returning an iterator, return an iterator pointing to the first character inserted.  
  
Member type iterator is a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) type that points to characters of the [basic\_string](http://www.cplusplus.com/basic_string).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | *// inserting into a string*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str="to be question";  std::string str2="the ";  std::string str3="or not to be";  std::string::iterator it;  *// used in the same order as described above:*  str.insert(6,str2); *// to be (the )question*  str.insert(6,str3,3,4); *// to be (not )the question*  str.insert(10,"that is cool",8); *// to be not (that is )the question*  str.insert(10,"to be "); *// to be not (to be )that is the question*  str.insert(15,1,':'); *// to be not to be(:) that is the question*  it = str.insert(str.begin()+5,','); *// to be(,) not to be: that is the question*  str.insert (str.end(),3,'.'); *// to be, not to be: that is the question(...)*  str.insert (it+2,str3.begin(),str3.begin()+3); *// (or )*  std::cout << str << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| to be, or not to be: that is the question... |

**Complexity**

Unspecified, but generally up to linear in the new [string length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If s does not point to an array long enough, or if either *p* or the range specified by [first,last) is not valid, it causes *undefined behavior*.  
  
If *pos* is greater than the [string length](http://www.cplusplus.com/basic_string::length), or if *subpos* is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::erase**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **sequence (1)** | basic\_string& erase (size\_type pos = 0, size\_type len = npos); |
| **character (2)** | iterator erase (iterator p); |
| **range (3)** | iterator erase (iterator first, iterator last); |

Erase characters from string

Erases part of the [basic\_string](http://www.cplusplus.com/basic_string), reducing its [length](http://www.cplusplus.com/basic_string::length):

(1) sequence

Erases the portion of the string value that begins at the character position *pos* and spans *len* characters (or until the *end of the string*, if either the content is too short or if *len* is [basic\_string::npos](http://www.cplusplus.com/basic_string::npos).  
Notice that the default argument erases all characters in the string (like member function [clear](http://www.cplusplus.com/basic_string::clear)).

(2) character

Erases the character pointed by *p*.

(3) range

Erases the sequence of characters in the range [first,last).

**Parameters**

pos

Position of the first character to be erased.  
If this is greater than the [string length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character in *str* is denoted by a value of 0 (not 1).

len

Number of characters to erase (if the string is shorter, as many characters as possible are erased).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of the string.

p

Iterator to the character to be removed.

first, last

Iterators specifying a range within the [basic\_string](http://www.cplusplus.com/basic_string)] to be removed: [first,last). i.e., the range includes all the characters between *first* and *last*, including the character pointed by *first* but not the one pointed by *last*.

Member type size\_type is an unsigned integral type.  
Member types iterator and const\_iterator are [random access iterator](http://www.cplusplus.com/RandomAccessIterator) types that point to characters of the [basic\_string](http://www.cplusplus.com/basic_string).

**Return value**

The *sequence version (1)* returns \*this.  
The others return an iterator referring to the character that now occupies the position of the first character erased, or [basic\_string::end](http://www.cplusplus.com/basic_string::end) if no such character exists.  
  
Member type iterator is a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) type that points to characters of the [basic\_string](http://www.cplusplus.com/basic_string).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | *// string::erase*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("This is an example sentence.");  std::cout << str << '\n';  *// "This is an example sentence."*  str.erase (10,8); *// ^^^^^^^^*  std::cout << str << '\n';  *// "This is an sentence."*  str.erase (str.begin()+9); *// ^*  std::cout << str << '\n';  *// "This is a sentence."*  str.erase (str.begin()+5, str.end()-9); *// ^^^^^*  std::cout << str << '\n';  *// "This sentence."*  *return* 0;  } |

Output:

|  |
| --- |
| This is an example sentence.  This is an sentence.  This is a sentence.  This sentence. |

**Complexity**

Unspecified, but generally up to linear in the new [string length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
An invalid *p* (or range) causes *undefined behavior*.  
  
If *pos* is greater than the [string length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::replace**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | basic\_string& replace (size\_type pos, size\_type len, const basic\_string& str);  basic\_string& replace (iterator i1, iterator i2, const basic\_string& str); |
| **substring (2)** | basic\_string& replace (size\_type pos, size\_type len, const basic\_string& str,  size\_type subpos, size\_type sublen); |
| **c-string (3)** | basic\_string& replace (size\_type pos, size\_type len, const charT\* s);  basic\_string& replace (iterator i1, iterator i2, const charT\* s); |
| **buffer (4)** | basic\_string& replace (size\_type pos, size\_type len, const charT\* s, size\_type n);  basic\_string& replace (iterator i1, iterator i2, const charT\* s, size\_type n); |
| **fill (5)** | basic\_string& replace (size\_type pos, size\_type len, size\_type n, charT c);  basic\_string& replace (iterator i1, iterator i2, size\_type n, charT c); |
| **range (6)** | template <class InputIterator>  basic\_string& replace (iterator i1, iterator i2,  InputIterator first, InputIterator last); |

Replace portion of string

Replaces the portion of the string that begins at character *pos* and spans *len* characters (or the part of the string in the range between [i1,i2)) by new contents:

(1) string

Copies *str*.

(2) substring

Copies the portion of *str* that begins at the character position *subpos* and spans *sublen* characters (or until the end of *str*, if either *str* is too short or if *sublen* is [basic\_string::npos](http://www.cplusplus.com/basic_string::npos)).

(3) c-string

Copies the null-terminated character sequence (C-string) pointed by *s*.  
The [length](http://www.cplusplus.com/string::length) is determined by calling [traits.length](http://www.cplusplus.com/char_traits::length)(s).

(4) buffer

Copies the first *n* characters from the array of characters pointed by *s*.

(5) fill

Replaces the portion of the string by *n* consecutive copies of character *c*.

(6) range

Copies the sequence of characters in the range [first,last), in the same order.

(7) initializer list

Copies each of the characters in *il*, in the same order.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc), whose value is copied.

pos

Position of the first character to be replaced.  
If this is greater than the [string length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).

len

Number of characters to replace (if the string is shorter, as many characters as possible are replaced).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of the string.

subpos

Position of the first character in *str* that is copied to the object as replacement.  
If this is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).

sublen

Length of the substring to be copied (if the string is shorter, as many characters as possible are copied).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of *str*.

s

Pointer to an array of characters (such as a *c-string*).

n

Number of characters to copy.

c

Character value, repeated *n* times.

first, last

[Input iterators](http://www.cplusplus.com/InputIterator) to the initial and final positions in a range. The range used is [first,last), which includes all the characters between *first* and *last*, including the character pointed by *first* but not the character pointed by *last*.  
The function template argument InputIterator shall be an [input iterator](http://www.cplusplus.com/InputIterator) type that points to elements of a type convertible to charT.

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
These objects are automatically constructed from *initializer list* declarators.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

\*this

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 | *// replacing in a string*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string base="this is a test string.";  std::string str2="n example";  std::string str3="sample phrase";  std::string str4="useful.";  *// replace signatures used in the same order as described above:*  *// Using positions: 0123456789\*123456789\*12345*  std::string str=base; *// "this is a test string."*  str.replace(9,5,str2); *// "this is an example string." (1)*  str.replace(19,6,str3,7,6); *// "this is an example phrase." (2)*  str.replace(8,10,"just a"); *// "this is just a phrase." (3)*  str.replace(8,6,"a shorty",7); *// "this is a short phrase." (4)*  str.replace(22,1,3,'!'); *// "this is a short phrase!!!" (5)*  *// Using iterators: 0123456789\*123456789\**  str.replace(str.begin(),str.end()-3,str3); *// "sample phrase!!!" (1)*  str.replace(str.begin(),str.begin()+6,"replace"); *// "replace phrase!!!" (3)*  str.replace(str.begin()+8,str.begin()+14,"is coolness",7); *// "replace is cool!!!" (4)*  str.replace(str.begin()+12,str.end()-4,4,'o'); *// "replace is cooool!!!" (5)*  str.replace(str.begin()+11,str.end(),str4.begin(),str4.end());*// "replace is useful." (6)*  std::cout << str << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| replace is useful. |

**Complexity**

Unspecified, but generally up to linear in the new [string length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If s does not point to an array long enough, or if the range specified by [first,last) is not valid, it causes *undefined behavior*.  
  
If *pos* is greater than the [string length](http://www.cplusplus.com/basic_string::length), or if *subpos* is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::swap**

void swap (basic\_string& str);

Swap string values

Exchanges the content of the container by the content of *str*, which is another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type. [Lengths](http://www.cplusplus.com/basic_string::length) may differ.  
  
After the call to this member function, the value of this object is the value *str* had before the call, and the value of *str* is the value this object had before the call.  
  
Notice that a non-member function exists with the same name, [swap](http://www.cplusplus.com/basic_string:swap), overloading that algorithm with an optimization that behaves like this member function.

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

No specifics on [allocators](http://www.cplusplus.com/basic_string::get_allocator).

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (i.e., instantiated with the same template parameters, charT, traits and Alloc), whose value is swapped with that of this [basic\_string](http://www.cplusplus.com/basic_string).

**Return value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | *// swap strings*  *#include <iostream>*  *#include <string>*  main ()  {  std::string buyer ("money");  std::string seller ("goods");  std::cout << "Before the swap, buyer has " << buyer;  std::cout << " and seller has " << seller << '\n';  seller.swap (buyer);  std::cout << " After the swap, buyer has " << buyer;  std::cout << " and seller has " << seller << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Before the swap, buyer has money and seller has goods  After the swap, buyer has goods and seller has money |

**Complexity**

Constant.

**Iterator validity**

Any iterators, pointers and references related to this object and to *str* may be invalidated.

**Data races**

Both the object and *str* are modified.

**Exception safety**

If the allocators in both [strings](http://www.cplusplus.com/basic_string) compare equal, or if their [allocator traits](http://www.cplusplus.com/allocator_traits) indicate that the allocators shall [propagate](http://www.cplusplus.com/allocator_traits#types), the function never throws exceptions (no-throw guarantee).  
Otherwise, it causes *undefined behavior*.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::pop\_back**

void pop\_back();

Delete last character

Erases the last character of the [basic\_string](http://www.cplusplus.com/basic_string), effectively reducing its [length](http://www.cplusplus.com/basic_string::length) by one.

**Parameters**

none

**Return value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 | *// string::pop\_back*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("hello world!");  str.pop\_back();  std::cout << str << '\n';  *return* 0;  } |

|  |
| --- |
| hello world |

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

Any iterators, pointers and references related to this object may be invalidated.

**Data races**

The object is modified.

**Exception safety**

If the [basic\_string](http://www.cplusplus.com/basic_string) is [empty](http://www.cplusplus.com/basic_string::empty), it causes *undefined behavior*.  
Otherwise, the function never throws exceptions (no-throw guarantee).

You are using a version without Ads of this website. *Please, consider donating:*

Top of Form



Bottom of Form

[[hide](javascript:donate.hide())]

**[basic\_string](http://www.cplusplus.com/reference/string/basic_string/)::c\_str**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

const charT\* c\_str() const;

Get C-string equivalent

Returns a pointer to an array that contains a null-terminated sequence of characters (i.e., a C-string) representing the current value of the [basic\_string](http://www.cplusplus.com/basic_string) object.  
  
This array includes the same sequence of characters that make up the value of the [basic\_string](http://www.cplusplus.com/basic_string) object plus an additional terminating null-character (charT()) at the end.

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

A program shall not alter any of the characters in this sequence.

The pointer returned may be invalidated by further calls to other member functions that modify the object.

**Parameters**

none

**Return Value**

A pointer to the c-string representation of the [basic\_string](http://www.cplusplus.com/basic_string) object's value.  
  
charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | *// strings and c-strings*  *#include <iostream>*  *#include <cstring>*  *#include <string>*  *int* main ()  {  std::string str ("Please split this sentence into tokens");  *char* \* cstr = *new* *char* [str.length()+1];  std::strcpy (cstr, str.c\_str());  *// cstr now contains a c-string copy of str*  *char* \* p = std::strtok (cstr," ");  *while* (p!=0)  {  std::cout << p << '\n';  p = strtok(NULL," ");  }  *delete*[] cstr;  *return* 0;  } |

Output:

|  |
| --- |
| Please  split  this  sentence  into  tokens |

* [C++98](javascript:switch3.select(1))
* [C++11](javascript:switch3.select(2))

**Complexity, iterator, access, exceptions**

Unspecified or contradictory specifications.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::data**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

const charT\* data() const;

Get string data

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Returns a pointer to an array that contains the same sequence of characters as the characters that make up the value of the [basic\_string](http://www.cplusplus.com/basic_string) object.  
  
Accessing the value at data()+size() produces *undefined behavior*: There are no guarantees that a null character terminates the character sequence pointed by the value returned by this function. See [basic\_string::c\_str](http://www.cplusplus.com/basic_string::c_str) for a function that provides such guarantee.  
  
A program shall not alter any of the characters in this sequence.

The pointer returned may be invalidated by further calls to other member functions that modify the object.

**Parameters**

none

**Return Value**

A pointer to the c-string representation of the [basic\_string](http://www.cplusplus.com/basic_string) object's value.  
  
charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 | *// string::data*  *#include <iostream>*  *#include <string>*  *#include <cstring>*  *int* main ()  {  *int* length;  std::string str = "Test string";  *char*\* cstr = "Test string";  *if* ( str.length() == std::strlen(cstr) )  {  std::cout << "str and cstr have the same length.\n";  *if* ( memcmp (cstr, str.data(), str.length() ) == 0 )  std::cout << "str and cstr have the same content.\n";  }  *return* 0;  } |

Output:

|  |
| --- |
| str and cstr have the same length.  str and cstr have the same content. |

* [C++98](javascript:switch3.select(1))
* [C++11](javascript:switch3.select(2))

**Complexity, iterator, access, exceptions**

Unspecified or contradictory specifications.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::get\_allocator**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

allocator\_type get\_allocator() const;

Get allocator

Returns a copy of the allocator object associated with the [basic\_string](http://www.cplusplus.com/basic_string).

**Parameters**

none

**Return Value**

The allocator.  
  
Member type allocator\_type is the type of the allocator used by the container, defined in [basic\_allocator](http://www.cplusplus.com/basic_allocator) as an alias of its third template parameter (Alloc).

**Complexity**

Unspecified, but generally constant.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
Copying any instantiation of the [default allocator](http://www.cplusplus.com/allocator) is also guaranteed to never throw.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::copy**

size\_type copy (charT\* s, size\_type len, size\_type pos = 0) const;

Copy sequence of characters from string

Copies a substring of the current value of the [basic\_string](http://www.cplusplus.com/basic_string) object into the array pointed by *s*. This substring contains the *len* characters that start at position *pos*.  
  
The function does not append a null character at the end of the copied content.

**Parameters**

s

Pointer to an array of characters.  
The array shall contain enough storage for the copied characters.

len

Number of characters to copy (if the string is shorter, as many characters as possible are copied).

pos

Position of the first character to be copied.  
If this is greater than the [string length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character in the [basic\_string](http://www.cplusplus.com/basic_string) is denoted by a value of 0 (not 1).

**Return value**

The number of characters copied to the array pointed by *s*. This may be equal to *len* or to [length()](http://www.cplusplus.com/basic_string::length)-pos (if the string value is shorter than pos+len).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 | *// string::copy*  *#include <iostream>*  *#include <string>*  *int* main ()  {  *char* buffer[20];  std::string str ("Test string...");  std::size\_t length = str.copy(buffer,6,5);  buffer[length]='\0';  std::cout << "buffer contains: " << buffer << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| buffer contains: string |

**Complexity**

Linear in the number of characters copied.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If *s* does not point to an array long enough, it causes *undefined behavior*.  
  
If *pos* is greater than the [string length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::find**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | size\_type find (const basic\_string& str, size\_type pos = 0) const; |
| **c-string (2)** | size\_type find (const charT\* s, size\_type pos = 0) const; |
| **buffer (3)** | size\_type find (const charT\* s, size\_type pos, size\_type n) const; |
| **character (4)** | size\_type find (charT c, size\_type pos = 0) const; |

Find first occurrence in string

Searches the [basic\_string](http://www.cplusplus.com/basic_string) for the first occurrence of the sequence specified by its arguments.  
  
When *pos* is specified, the search only includes characters at or after position *pos*, ignoring any possible occurrences that include characters before *pos*.  
  
The function uses [traits\_type::eq](http://www.cplusplus.com/char_traits::eq) to determine character equivalences.  
  
Notice that unlike member [find\_first\_of](http://www.cplusplus.com/basic_string::find_first_of), whenever more than one character is being searched for, it is not enough that just one of these characters match, but the entire sequence must match.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) with the subject to search for.

pos

Position of the first character in the string to be considered in the search.  
If this is greater than the [string length](http://www.cplusplus.com/basic_string::length), the function never finds matches.  
Note: The first character is denoted by a value of 0 (not 1): A value of 0 means that the entire string is searched.

s

Pointer to an array of characters.  
If argument *n* is specified *(3)*, the sequence to match are the first *n* characters in the array.  
Otherwise *(2)*, a null-terminated sequence is expected: the length of the sequence to match is determined by the first occurrence of a null character.

n

Length of sequence of characters to match.

c

Individual character to be searched for.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

The position of the first character of the first match.  
If no matches were found, the function returns [basic\_string::npos](http://www.cplusplus.com/basic_string::npos).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 | *// string::find*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("There are two needles in this haystack with needles.");  std::string str2 ("needle");  *// different member versions of find in the same order as above:*  std::string::size\_type found = str.find(str2);  *if* (found!=std::string::npos)  std::cout << "first 'needle' found at: " << found << '\n';  found=str.find("needles are small",found+1,6);  *if* (found!=std::string::npos)  std::cout << "second 'needle' found at: " << found << '\n';  found=str.find("haystack");  *if* (found!=std::string::npos)  std::cout << "'haystack' also found at: " << found << '\n';  found=str.find('.');  *if* (found!=std::string::npos)  std::cout << "Period found at: " << found << '\n';  *// let's replace the first needle:*  str.replace(str.find(str2),str2.length(),"preposition");  std::cout << str << '\n';  *return* 0;  } |

Notice how parameter *pos* is used to search for a second instance of the same search string. Output:

|  |
| --- |
| first 'needle' found at: 14  second 'needle' found at: 44  'haystack' also found at: 30  Period found at: 51  There are two prepositions in this haystack with needles. |

**Complexity**

Unspecified, but generally up to linear in [length()](http://www.cplusplus.com/basic_string::length)-pos times the length of the sequence to match (worst case).

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

If s does not point to an array long enough, it causes *undefined behavior*.  
Otherwise, the function never throws exceptions (no-throw guarantee).

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::rfind**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | size\_type rfind (const basic\_string& str, size\_type pos = npos) const; |
| **c-string (2)** | size\_type rfind (const charT\* s, size\_type pos = npos) const; |
| **buffer (3)** | size\_type rfind (const charT\* s, size\_type pos, size\_type n) const; |
| **character (4)** | size\_type rfind (charT c, size\_type pos = npos) const; |

Find last occurrence in string

Searches the [basic\_string](http://www.cplusplus.com/basic_string) for the last occurrence of the sequence specified by its arguments.  
  
When *pos* is specified, the search only includes sequences of characters that begin at or before position *pos*, ignoring any possible match beginning after *pos*.  
  
The function uses [traits\_type::eq](http://www.cplusplus.com/char_traits::eq) to determine character equivalences.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) with the subject to search for.

pos

Position of the last character in the string to be considered as the beginning of a match.  
Any value greater or equal than the [string length](http://www.cplusplus.com/basic_string::length) (including [basic\_string::npos](http://www.cplusplus.com/basic_string::npos)) means that the entire string is searched.  
Note: The first character is denoted by a value of 0 (not 1).

s

Pointer to an array of characters.  
If argument *n* is specified *(3)*, the sequence to match are the first *n* characters in the array.  
Otherwise *(2)*, a null-terminated sequence is expected: the length of the sequence to match is determined by the first occurrence of a null character.

n

Length of sequence of characters to match.

c

Individual character to be searched for.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

The position of the first character of the last match.  
If no matches were found, the function returns [basic\_string::npos](http://www.cplusplus.com/basic_string::npos).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | *// string::rfind*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("The sixth sick sheik's sixth sheep's sick.");  std::string key ("sixth");  std::string::size\_type found = str.rfind(key);  *if* (found!=std::string::npos)  str.replace (found,key.length(),"seventh");  std::cout << str << '\n';  *return* 0;  } |

|  |
| --- |
| The sixth sick sheik's seventh sheep's sick. |

**Complexity**

Unspecified, but generally up to linear in the [string length](http://www.cplusplus.com/basic_string::length) (or *pos*) times the number of characters to match (worst case).

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

If s does not point to an array long enough, it causes *undefined behavior*.  
Otherwise, the function never throws exceptions (no-throw guarantee).

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::find\_first\_of**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | size\_type find\_first\_of (const basic\_string& str, size\_type pos = 0) const; |
| **c-string (2)** | size\_type find\_first\_of (const charT\* s, size\_type pos = 0) const; |
| **buffer (3)** | size\_type find\_first\_of (const charT\* s, size\_type pos, size\_type n) const; |
| **character (4)** | size\_type find\_first\_of (charT c, size\_type pos = 0) const; |

Find character in string

Searches the [basic\_string](http://www.cplusplus.com/basic_string) for the first character that matches **any** of the characters specified in its arguments.  
  
When *pos* is specified, the search only includes characters at or after position *pos*, ignoring any possible occurrences before *pos*.  
  
Notice that it is enough for one single character of the sequence to match (not all of them). See [basic\_string::find](http://www.cplusplus.com/basic_string::find) for a function that matches entire sequences.  
  
The function uses [traits\_type::eq](http://www.cplusplus.com/char_traits::eq) to determine character equivalences.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) with the characters to search for.

pos

Position of the first character in the string to be considered in the search.  
If this is greater than the [string length](http://www.cplusplus.com/basic_string::length), the function never finds matches.  
Note: The first character is denoted by a value of 0 (not 1): A value of 0 means that the entire string is searched.

s

Pointer to an array of characters.  
If argument *n* is specified *(3)*, the first *n* characters in the array are searched for.  
Otherwise *(2)*, a null-terminated sequence is expected: the length of the sequence with the characters to match is determined by the first occurrence of a null character.

n

Number of character values to search for.

c

Individual character to be searched for.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

The position of the first character that matches.  
If no matches are found, the function returns [basic\_string::npos](http://www.cplusplus.com/basic_string::npos).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | *// string::find\_first\_of*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("PLease, replace the vowels in this sentence by asterisks.");  std::string::size\_type found = str.find\_first\_of("aeiou");  *while* (found!=std::string::npos)  {  str[found]='\*';  found=str.find\_first\_of("aeiou",found+1);  }  std::cout << str << '\n';  *return* 0;  } |

|  |
| --- |
| Pl\*\*s\*, \*pl\*c\* th\* v\*w\*ls \*n th\*s s\*nt\*nc\* by \*st\*r\*sks. |

**Complexity**

Unspecified, but generally up to linear in [length()](http://www.cplusplus.com/basic_string::length)-pos times the number of characters to match (worst case).

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

If s does not point to an array long enough, it causes *undefined behavior*.  
Otherwise, the function never throws exceptions (no-throw guarantee).

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::find\_last\_of**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | size\_type find\_last\_of (const basic\_string& str, size\_type pos = npos) const; |
| **c-string (2)** | size\_type find\_last\_of (const charT\* s, size\_type pos = npos) const; |
| **buffer (3)** | size\_type find\_last\_of (const charT\* s, size\_type pos, size\_type n) const; |
| **character (4)** | size\_type find\_last\_of (charT c, size\_type pos = npos) const; |

Find character in string from the end

Searches the [basic\_string](http://www.cplusplus.com/basic_string) for the last character that matches **any** of the characters specified in its arguments.  
  
When *pos* is specified, the search only includes characters at or before position *pos*, ignoring any possible occurrences after *pos*.  
  
The function uses [traits\_type::eq](http://www.cplusplus.com/char_traits::eq) to determine character equivalences.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) with the characters to search for.

pos

Position of the last character in the string to be considered in the search.  
Any value greater or equal than the [string length](http://www.cplusplus.com/basic_string::length) (including [basic\_string::npos](http://www.cplusplus.com/basic_string::npos)) means that the entire string is searched.  
Note: The first character is denoted by a value of 0 (not 1).

s

Pointer to an array of characters.  
If argument *n* is specified *(3)*, the first *n* characters in the array are searched for.  
Otherwise *(2)*, a null-terminated sequence is expected: the length of the sequence with the characters to match is determined by the first occurrence of a null character.

n

Number of character values to search for.

c

Individual character to be searched for.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

The position of the last character that matches.  
If no matches are found, the function returns [basic\_string::npos](http://www.cplusplus.com/basic_string::npos).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | *// string::find\_last\_of*  *#include <iostream>*  *#include <string>*  *void* SplitFilename (*const* std::string& str)  {  std::cout << "Splitting: " << str << '\n';  std::string::size\_type found = str.find\_last\_of("/\\");  std::cout << " path: " << str.substr(0,found) << '\n';  std::cout << " file: " << str.substr(found+1) << '\n';  }  *int* main ()  {  std::string str1 ("/usr/bin/man");  std::string str2 ("c:\\windows\\winhelp.exe");  SplitFilename (str1);  SplitFilename (str2);  *return* 0;  } |

|  |
| --- |
| Splitting: /usr/bin/man  path: /usr/bin  file: man  Splitting: c:\windows\winhelp.exe  path: c:\windows  file: winhelp.exe |

**Complexity**

Unspecified, but generally up to linear in the [string length](http://www.cplusplus.com/basic_string::length) (or *pos*) times the number of characters to match (worst case).

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

If s does not point to an array long enough, it causes *undefined behavior*.  
Otherwise, the function never throws exceptions (no-throw guarantee).

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::find\_first\_not\_of**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | size\_type find\_first\_not\_of (const basic\_string& str, size\_type pos = 0) const; |
| **c-string (2)** | size\_type find\_first\_not\_of (const charT\* s, size\_type pos = 0) const; |
| **buffer (3)** | size\_type find\_first\_not\_of (const charT\* s, size\_type pos, size\_type n) const; |
| **character (4)** | size\_type find\_first\_not\_of (charT c, size\_type pos = 0) const; |

Find non-matching character in string

Searches the [basic\_string](http://www.cplusplus.com/basic_string) for the first character that does not match any of the characters specified in its arguments.  
  
When *pos* is specified, the search only includes characters at or after position *pos*, ignoring any possible occurrences before that character.  
  
The function uses [traits\_type::eq](http://www.cplusplus.com/char_traits::eq) to determine character equivalences.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) with the set of characters to be used in the search.

pos

Position of the first character in the string to be considered in the search.  
If this is greater than the [string length](http://www.cplusplus.com/basic_string::length), the function never finds matches.  
Note: The first character is denoted by a value of 0 (not 1): A value of 0 means that the entire string is searched.

s

Pointer to an array of characters.  
If argument *n* is specified *(3)*, the first *n* characters in the array are used in the search.  
Otherwise *(2)*, a null-terminated sequence is expected: the length of the sequence with the characters used in the search is determined by the first occurrence of a null character.

n

Number of character values to search for.

c

Individual character to be searched for.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

The position of the first character that does not match.  
If no such characters are found, the function returns [basic\_string::npos](http://www.cplusplus.com/basic_string::npos).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | *// string::find\_first\_not\_of*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("look for non-alphabetic characters...");  std::string::size\_type found = str.find\_first\_not\_of("abcdefghijklmnopqrstuvwxyz ");  *if* (found!=std::string::npos)  {  std::cout << "The first non-alphabetic character is " << str[found];  std::cout << " at position " << found << '\n';  }  *return* 0;  } |

|  |
| --- |
| The first non-alphabetic character is - at position 12 |

**Complexity**

Unspecified, but generally up to linear in [length()](http://www.cplusplus.com/basic_string::length)-pos times the number of characters to match (worst case).

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

If s does not point to an array long enough, it causes *undefined behavior*.  
Otherwise, the function never throws exceptions (no-throw guarantee).

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::find\_last\_not\_of**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | size\_type find\_last\_not\_of (const basic\_string& str, size\_type pos = npos) const; |
| **c-string (2)** | size\_type find\_last\_not\_of (const charT\* s, size\_type pos = npos) const; |
| **buffer (3)** | size\_type find\_last\_not\_of (const charT\* s, size\_type pos, size\_type n) const; |
| **character (4)** | size\_type find\_last\_not\_of (charT c, size\_type pos = npos) const; |

Find non-matching character in string from the end

Searches the [basic\_string](http://www.cplusplus.com/basic_string) for the last character that does not match any of the characters specified in its arguments.  
  
When *pos* is specified, the search only includes characters at or before position *pos*, ignoring any possible occurrences after *pos*.  
  
The function uses [traits\_type::eq](http://www.cplusplus.com/char_traits::eq) to determine character equivalences.

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) with the set of characters to be used in the search.

pos

Position of the last character in the string to be considered in the search.  
Any value greater or equal than the [string length](http://www.cplusplus.com/basic_string::length) (including [basic\_string::npos](http://www.cplusplus.com/basic_string::npos)) means that the entire string is searched.  
Note: The first character is denoted by a value of 0 (not 1).

s

Pointer to an array of characters.  
If argument *n* is specified *(3)*, the first *n* characters in the array are used in the search.  
Otherwise *(2)*, a null-terminated sequence is expected: the length of the sequence with the characters used in the search is determined by the first occurrence of a null character.

n

Number of character values to search for.

c

Individual character to be searched for.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

The position of the first character that does not match.  
If no such characters are found, the function returns [basic\_string::npos](http://www.cplusplus.com/basic_string::npos).  
  
Member type size\_type is an unsigned integral type.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | *// string::find\_last\_not\_of*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str ("Please, erase trailing white-spaces \n");  std::string whitespaces (" \t\f\v\n\r");  std::string::size\_type found = str.find\_last\_not\_of(whitespaces);  *if* (found!=std::string::npos)  str.erase(found+1);  *else*  str.clear(); *// str is all whitespace*  std::cout << '[' << str << "]\n";  *return* 0;  } |

|  |
| --- |
| [Please, erase trailing white-spaces] |

**Complexity**

Unspecified, but generally up to linear in the [string length](http://www.cplusplus.com/basic_string::length) (or *pos*) times the number of characters to match (worst case).

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

If s does not point to an array long enough, it causes *undefined behavior*.  
Otherwise, the function never throws exceptions (no-throw guarantee).

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::substr**

basic\_string substr (size\_type pos = 0, size\_type len = npos) const;

Generate substring

Returns a newly constructed [basic\_string](http://www.cplusplus.com/basic_string) object with its value initialized to a copy of a substring of this object.  
  
The substring is the portion of the object that starts at character position *pos* and spans *len* characters (or until the end of the string, whichever comes first).

**Parameters**

pos

Position of the first character to be copied as a substring.  
If this is equal to the [*string length*](http://www.cplusplus.com/basic_string::length), the function returns an empty string.  
If this is greater than the [*string length*](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character is denoted by a value of 0 (not 1).

len

Number of characters to include in the substring (if the string is shorter, as many characters as possible are used).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of the string.

Member type size\_type is an unsigned integral type.

**Return Value**

A [basic\_string](http://www.cplusplus.com/basic_string) object with a substring of this object.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | *// string::substr*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str="We think in generalities, but we live in details.";  *// (quoting Alfred N. Whitehead)*  std::string str2 = str.substr (12,12); *// "generalities"*  std::string::size\_type pos = str.find("live"); *// position of "live" in str*  std::string str3 = str.substr (pos); *// get from "live" to the end*  std::cout << str2 << ' ' << str3 << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| generalities live in details. |

**Complexity**

Unspecified, but generally linear in the [length](http://www.cplusplus.com/basic_string::length) of the returned object.

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string).  
  
If *pos* is greater than the [string length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

public member function

<string>

**std::**[**basic\_string**](http://www.cplusplus.com/reference/string/basic_string/)**::compare**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | int compare (const basic\_string& str) const; |
| **substrings (2)** | int compare (size\_type pos, size\_type len, const basic\_string& str) const;  int compare (size\_type pos, size\_type len, const basic\_string& str,  size\_type subpos, size\_type sublen) const; |
| **c-string (3)** | int compare (const charT\* s) const;  int compare (size\_type pos, size\_type len, const charT\* s) const; |
| **buffer (4)** | int compare (size\_type pos, size\_type len, const charT\* s, size\_type n) const; |

Compare strings

Compares the value of the [basic\_string](http://www.cplusplus.com/basic_string) object (or a substring) to the sequence of characters specified by its arguments.  
  
The *compared string* is the value of the [basic\_string](http://www.cplusplus.com/basic_string) object or -if the signature used has a *pos* and a *len* parameters- the substring that begins at its character in position *pos* and spans *len* characters.  
  
This string is compared to a *comparing string*, which is determined by the other arguments passed to the function.  
  
The sequences are compared using [traits\_type::compare](http://www.cplusplus.com/char_traits::compare).

**Parameters**

str

Another [basic\_string](http://www.cplusplus.com/basic_string) object of the same type (with the same class template arguments charT, traits and Alloc), used entirely (or partially) as the *comparing string*.

pos

Position of the first character in the *compared string*.  
If this is greater than the [string length](http://www.cplusplus.com/basic_string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character is denoted by a value of 0 (not 1).

len

Length of *compared string* (if the string is shorter, as many characters as possible).  
A value of [basic\_string::npos](http://www.cplusplus.com/basic_string::npos) indicates all characters until the end of the string.

subpos, sublen

Same as *pos* and *len* above, but for the *comparing string*.

s

Pointer to an array of characters.  
If argument *n* is specified *(4)*, the first *n* characters in the array are used as the *comparing string*.  
Otherwise *(3)*, a null-terminated sequence is expected: the length of the sequence with the characters to use as *comparing string* is determined by the first occurrence of a null character.

n

Number of characters to compare.

charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).  
Member type size\_type is an unsigned integral type.

**Return Value**

Returns a signed integral indicating the relation between the strings:

|  |  |
| --- | --- |
| **value** | **relation between *compared string* and *comparing string*** |
| 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. |

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | *// comparing apples with apples*  *#include <iostream>*  *#include <string>*  *int* main ()  {  std::string str1 ("green apple");  std::string str2 ("red apple");  *if* (str1.compare(str2) != 0)  std::cout << str1 << " is not " << str2 << '\n';  *if* (str1.compare(6,5,"apple") == 0)  std::cout << "still, " << str1 << " is an apple\n";  *if* (str2.compare(str2.size()-5,5,"apple") == 0)  std::cout << "and " << str2 << " is also an apple\n";  *if* (str1.compare(6,5,str2,4,5) == 0)  std::cout << "therefore, both are apples\n";  *return* 0;  } |

Output:

|  |
| --- |
| green apple is not red apple  still, green apple is an apple  and red apple is also an apple  therefore, both are apples |

**Complexity**

Unspecified, but generally up to linear in both the *compared* and *comparing string*'s [lengths](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

No changes.

**Data races**

The object is accessed.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string) (except *(1)*, which is guaranteed to not throw).  
  
If s does not point to an array long enough, it causes *undefined behavior*.  
  
If *pos* is greater than the [string length](http://www.cplusplus.com/basic_string::length), or if *subpos* is greater than *str*'s [length](http://www.cplusplus.com/basic_string::length), an [out\_of\_range](http://www.cplusplus.com/out_of_range) exception is thrown.  
function template

<string>

**std::operator+ (basic\_string)**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **string (1)** | template <class charT, class traits, class Alloc>  basic\_string operator+ (const basic\_string<charT,traits,Alloc>& lhs,  const basic\_string<charT,traits,Alloc>& rhs); |
| **c-string (2)** | template <class charT, class traits, class Alloc>  basic\_string operator+ (const basic\_string<charT,traits,Alloc>& lhs, const charT\* rhs);  template <class charT, class traits, class Alloc>  basic\_string operator+ (const charT\* lhs, const basic\_string<charT,traits,Alloc>& rhs); |
| **character (3)** | template <class charT, class traits, class Alloc>  basic\_string operator+ (const basic\_string<charT,traits,Alloc>& lhs, charT rhs);  template <class charT, class traits, class Alloc>  basic\_string operator+ (charT lhs, const basic\_string<charT,traits,Alloc>& rhs); |

Concatenate strings

Returns a newly constructed [basic\_string](http://www.cplusplus.com/basic_string) object with its value being the concatenation of the characters in *lhs* followed by those of *rhs*.

* [C++11](javascript:switch2.select(1))

In the signatures taking at least one *rvalue reference* as argument, the returned object is *move-constructed* by passing this argument, which is left in an unspecified but valid state. If both arguments are *rvalue references*, only one of them is moved (it is unspecified which), with the other one preserving its value.

**Parameters**

lhs, rhs

Arguments to the left- and right-hand side of the operator, respectively.  
If of type charT\*, it shall point to a null-terminated character sequence.  
charT is [basic\_string](http://www.cplusplus.com/basic_string)'s character type (i.e., its first template parameter).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | *// concatenating strings*  *#include <iostream>*  *#include <string>*  main ()  {  std::string firstlevel ("com");  std::string secondlevel ("cplusplus");  std::string scheme ("http://");  std::string hostname;  std::string url;  hostname = "www." + secondlevel + '.' + firstlevel;  url = scheme + hostname;  std::cout << url << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| http://www.cplusplus.com |

**Return Value**

A [basic\_string](http://www.cplusplus.com/basic_string) object whose value is the concatenation of *lhs* and *rhs*.

**Complexity**

Unspecified, but generally linear in the resulting [string length](http://www.cplusplus.com/string::length) (and linear in the length of the non-moved argument for signatures with *rvalue references*).

**Iterator validity**

The signatures with *rvalue references* may invalidate iterators, pointers and references related to the moved [basic\_string](http://www.cplusplus.com/basic_string) object.

**Data races**

The signatures with *rvalue references* modify the moved [basic\_string](http://www.cplusplus.com/basic_string) object.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in either [basic\_string](http://www.cplusplus.com/basic_string) objects.  
  
If s is not a null-terminated character sequence, it causes *undefined behavior*.  
  
If the resulting [string length](http://www.cplusplus.com/basic_string::length) would exceed the [max\_size](http://www.cplusplus.com/basic_string::max_size), a [length\_error](http://www.cplusplus.com/length_error) exception is thrown.  
If the type uses the [default allocator](http://www.cplusplus.com/allocator), a [bad\_alloc](http://www.cplusplus.com/bad_alloc) exception is thrown if the function needs to allocate storage and fails.

function template

<string>

**std::relational operators (basic\_string)**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **(1)** | template <class charT, class traits, class Alloc>  bool operator== (const basic\_string<charT,traits,Alloc>& lhs,  const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator== (const charT\* lhs, const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator== (const basic\_string<charT,traits,Alloc>& lhs, const charT\* rhs); |
| **(2)** | template <class charT, class traits, class Alloc>  bool operator!= (const basic\_string<charT,traits,Alloc>& lhs,  const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator!= (const charT\* lhs, const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator!= (const basic\_string<charT,traits,Alloc>& lhs, const charT\* rhs); |
| **(3)** | template <class charT, class traits, class Alloc>  bool operator< (const basic\_string<charT,traits,Alloc>& lhs,  const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator< (const charT\* lhs, const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator< (const basic\_string<charT,traits,Alloc>& lhs, const charT\* rhs); |
| **(4)** | template <class charT, class traits, class Alloc>  bool operator<= (const basic\_string<charT,traits,Alloc>& lhs,  const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator<= (const charT\* lhs, const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator<= (const basic\_string<charT,traits,Alloc>& lhs, const charT\* rhs); |
| **(5)** | template <class charT, class traits, class Alloc>  bool operator> (const basic\_string<charT,traits,Alloc>& lhs,  const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator> (const charT\* lhs, const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator> (const basic\_string<charT,traits,Alloc>& lhs, const charT\* rhs); |
| **(6)** | template <class charT, class traits, class Alloc>  bool operator>= (const basic\_string<charT,traits,Alloc>& lhs,  const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator>= (const charT\* lhs, const basic\_string<charT,traits,Alloc>& rhs);  template <class charT, class traits, class Alloc>  bool operator>= (const basic\_string<charT,traits,Alloc>& lhs, const charT\* rhs); |

Relational operators for basic\_string

Performs the appropriate comparison operation between the [basic\_string](http://www.cplusplus.com/basic_string) objects *lhs* and *rhs*.  
  
The functions use [basic\_string::compare](http://www.cplusplus.com/basic_string::compare) for the comparison, which depends on the [compare](http://www.cplusplus.com/char_traits::compare) member of its [character traits](http://www.cplusplus.com/char_traits).  
  
These operators are overloaded in header [<string>](http://www.cplusplus.com/%3Cstring%3E).

**Parameters**

lhs, rhs

Arguments to the left- and right-hand side of the operator, respectively.  
If of type charT\*, it shall point to a null-terminated character sequence.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | *// string comparisons*  *#include <iostream>*  *#include <vector>*  *int* main ()  {  std::string foo = "alpha";  std::string bar = "beta";  *if* (foo==bar) std::cout << "foo and bar are equal\n";  *if* (foo!=bar) std::cout << "foo and bar are not equal\n";  *if* (foo< bar) std::cout << "foo is less than bar\n";  *if* (foo> bar) std::cout << "foo is greater than bar\n";  *if* (foo<=bar) std::cout << "foo is less than or equal to bar\n";  *if* (foo>=bar) std::cout << "foo is greater than or equal to bar\n";  *return* 0;  } |

Output:

|  |
| --- |
| foo and bar are not equal  foo is less than bar  foo is less than or equal to bar |

**Return Value**

true if the condition holds, and false otherwise.

**Complexity**

Unspecified, but generally up to linear in both *lhs* and *rhs*'s [lengths](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

No changes.

**Data races**

Both objects, *lhs* and *rhs*, are accessed.

**Exception safety**

**Strong guarantee:** if an exception is thrown, there are no changes in the [basic\_string](http://www.cplusplus.com/basic_string) (except operations between [basic\_string](http://www.cplusplus.com/basic_string) objects, which are guaranteed to not throw).  
  
If an argument of type charT\* does not point to null-terminated character sequence, it causes *undefined behavior*.

You are using a version without Ads of this website. *Please, consider donating:*

Top of Form



Bottom of Form

[[hide](javascript:donate.hide())]

function template

<string>

**std::swap (basic\_string)**

template <class charT, class traits, class Alloc>

void swap (basic\_string<charT,traits,Alloc>& x,

basic\_string<charT,traits,Alloc>& y);

Exchanges the values of two strings

Exchanges the values of [basic\_string](http://www.cplusplus.com/basic_string) objects *x* and *y*, such that after the call to this function, the value of *x* is the one which was on *y* before the call, and the value of *y* is that of *x*.  
  
This is an overload of the generic algorithm [swap](http://www.cplusplus.com/swap) that improves its performance by mutually transferring ownership over their internal data to the other object (i.e., the strings exchange references to their data, without actually copying the characters): It behaves as if x.[swap](http://www.cplusplus.com/basic_string::swap)(y) was called.

**Parameters**

x,y

[basic\_string](http://www.cplusplus.com/basic_string) objects of the same type (i.e., having both the same template parameters, charT, traits and Alloc).

**Return value**

none

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | *// swap strings*  *#include <iostream>*  *#include <string>*  main ()  {  std::string buyer ("money");  std::string seller ("goods");  std::cout << "Before the swap, buyer has " << buyer;  std::cout << " and seller has " << seller << '\n';  swap (buyer,seller);  std::cout << " After the swap, buyer has " << buyer;  std::cout << " and seller has " << seller << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Before the swap, buyer has money and seller has goods  After the swap, buyer has goods and seller has money |

**Complexity**

Constant.

**Iterator validity**

Any iterators, pointers and references related to both *x* and *y* may be invalidated.

**Data races**

Both objects, *x* and *y*, are modified.

**Exception safety**

If the allocators in both [strings](http://www.cplusplus.com/basic_string) compare equal, or if their [allocator traits](http://www.cplusplus.com/allocator_traits) indicate that the allocators shall [propagate](http://www.cplusplus.com/allocator_traits" \l "types), the function never throws exceptions (no-throw guarantee).  
Otherwise, it causes *undefined behavior*.

function template

<string>

**std::operator>> (basic\_string)**

template <class charT, class traits, class Alloc>

basic\_istream<charT,traits>& operator>> (basic\_istream<charT,traits>& is,

basic\_string<charT,traits,Alloc>& str);

Extract string from stream

Extracts a string from the input stream *is*, storing the sequence in *str*, which is overwritten (the previous value of *str* is replaced).  
  
This function overloads operator>> to behave as described in [basic\_istream::operator>>](http://www.cplusplus.com/basic_istream::operator%3E%3E) for c-strings, but applied to [basic\_string](http://www.cplusplus.com/basic_string) objects.  
  
Each extracted character is appended to the [basic\_string](http://www.cplusplus.com/basic_string) as if its member [push\_back](http://www.cplusplus.com/basic_string::push_back) was called.  
  
Notice that the [basic\_istream](http://www.cplusplus.com/basic_istream) extraction operations use whitespaces as separators; Therefore, this operation will only extract what can be considered a word from the stream. To extract entire lines of text, see the [basic\_string](http://www.cplusplus.com/basic_string) overload of global function [getline](http://www.cplusplus.com/basic_string:getline).

**Parameters**

is

[basic\_istream](http://www.cplusplus.com/basic_istream) object from which characters are extracted.

str

[basic\_string](http://www.cplusplus.com/basic_string) object where the extracted content is stored.

**Return Value**

The same as parameter *is*.  
  
A call to this function may set any of the internal state flags of *is* if:

|  |  |
| --- | --- |
| **flag** | **error** |
| eofbit | The end of the source of characters is reached during its operations. |
| failbit | The input obtained could not be interpreted as a valid textual representation of an object of this type. In this case, *distr* preserves the parameters and internal data it had before the call. Notice that some eofbit cases will also set failbit. |
| badbit | An error other than the above happened. |

(see [ios\_base::iostate](http://www.cplusplus.com/ios_base::iostate) for more info on these)  
  
Additionally, in any of these cases, if the appropriate flag has been set with *is*'s member function [basic\_ios::exceptions](http://www.cplusplus.com/basic_ios::exceptions), an exception of type [ios\_base::failure](http://www.cplusplus.com/ios_base::failure) is thrown.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | *// extract to string*  *#include <iostream>*  *#include <string>*  main ()  {  std::string name;  std::cout << "Please, enter your name: ";  std::cin >> name;  std::cout << "Hello, " << name << "!\n";  *return* 0;  } |

**Complexity**

Unspecified, but generally linear in the resulting [length](http://www.cplusplus.com/basic_string::length) of *str*.

**Iterator validity**

Any iterators, pointers and references related to *str* may be invalidated.

**Data races**

Both objects, *is* and *str*, are modified.

**Exception safety**

**Basic guarantee:** if an exception is thrown, both *is* and *str* end up in a valid state.

function template

<string>

**std::operator<< (basic\_string)**

template <class charT, class traits, class Alloc>

basic\_ostream<charT,traits>& operator<< (basic\_ostream<charT,traits>& os,

const basic\_string<charT,traits,Alloc>& str);

Insert string into stream

Inserts the sequence of characters that conforms value of *str* into *os*.  
  
This function overloads operator<< to behave as described in [basic\_ostream::operator<<](http://www.cplusplus.com/basic_ostream::operator%3C%3C) for c-strings, but applied to [basic\_string](http://www.cplusplus.com/basic_string) objects.

**Parameters**

os

[basic\_ostream](http://www.cplusplus.com/basic_ostream) object where characters are inserted.

str

[basic\_string](http://www.cplusplus.com/basic_string) object with the content to insert.

**Return Value**

The same as parameter *os*.  
  
If some error happens during the output operation, the stream's *badbit* flag is set, and if the appropriate flag has been set with [basic\_ios::exceptions](http://www.cplusplus.com/basic_ios::exceptions), an exception is thrown.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 | *// inserting strings into output streams*  *#include <iostream>*  *#include <string>*  main ()  {  std::string str = "Hello world!";  std::cout << str << '\n';  *return* 0;  } |

**Complexity**

Unspecified, but generally linear in *str*'s [length](http://www.cplusplus.com/basic_string::length).

**Iterator validity**

No changes.

**Data races**

Objects *os* is modified.

**Exception safety**

**Basic guarantee:** if an exception is thrown, both *is* and *str* end up in a valid state.

function template

<string>

**std::getline (basic\_string)**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **(1)** | template <class charT, class traits, class Alloc>  basic\_istream<charT,traits>& getline (basic\_istream<charT,traits>& is,  basic\_string<charT,traits,Alloc>& str, charT delim); |
| **(2)** | template <class charT, class traits, class Alloc>  basic\_istream<charT,traits>& getline (basic\_istream<charT,traits>& is,  basic\_string<charT,traits,Alloc>& str); |

Get line from stream into string

Extracts characters from *is* and stores them into *str* until the delimitation character *delim* is found (or the newline character, for *(2)*).  
  
The extraction also stops if the end of file is reached in *is* or if some other error occurs during the input operation.  
  
If the delimiter is found, it is extracted and discarded, i.e. it is not stored and the next input operation will begin after it.  
  
Each extracted character is appended to the [basic\_string](http://www.cplusplus.com/basic_string) as if its member [push\_back](http://www.cplusplus.com/basic_string::push_back) was called.

**Parameters**

is

[basic\_istream](http://www.cplusplus.com/basic_istream) object from which characters are extracted.

str

[basic\_string](http://www.cplusplus.com/basic_string) object where the extracted line is stored.

**Return Value**

The same as parameter *is*.  
  
A call to this function may set any of the internal state flags of *is* if:

|  |  |
| --- | --- |
| **flag** | **error** |
| eofbit | The end of the source of characters is reached during its operations. |
| failbit | The input obtained could not be interpreted as a valid textual representation of an object of this type. In this case, *distr* preserves the parameters and internal data it had before the call. Notice that some eofbit cases will also set failbit. |
| badbit | An error other than the above happened. |

(see [ios\_base::iostate](http://www.cplusplus.com/ios_base::iostate) for more info on these)  
  
Additionally, in any of these cases, if the appropriate flag has been set with *is*'s member function [basic\_ios::exceptions](http://www.cplusplus.com/basic_ios::exceptions), an exception of type [ios\_base::failure](http://www.cplusplus.com/ios_base::failure) is thrown.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 | *// extract to string*  *#include <iostream>*  *#include <string>*  main ()  {  std::string name;  std::cout << "Please, enter your full name: ";  std::getline (std::cin,name);  std::cout << "Hello, " << name << "!\n";  *return* 0;  } |

**Complexity**

Unspecified, but generally linear in the resulting [length](http://www.cplusplus.com/basic_string::length) of *str*.

**Iterator validity**

Any iterators, pointers and references related to *str* may be invalidated.

**Data races**

Both objects, *is* and *str*, are modified.

**Exception safety**

**Basic guarantee:** if an exception is thrown, both *is* and *str* end up in a valid state.