# Source:

## <http://www.yolinux.com/TUTORIALS/LinuxTutorialC++StringClass.html>

## <http://www.cplusplus.com/reference/string/string/>

**string**

std::string

**String class**

Strings are objects that represent sequences of characters.  
  
The standard string class provides support for such objects with an interface similar to that of a [standard container](http://www.cplusplus.com/stl) of bytes, but adding features specifically designed to operate with strings of single-byte characters.  
  
The string class is an instantiation of the [basic\_string](http://www.cplusplus.com/basic_string) class template that uses char (i.e., bytes) as its *character type*, with its default [char\_traits](http://www.cplusplus.com/char_traits) and [allocator](http://www.cplusplus.com/allocator) types (see [basic\_string](http://www.cplusplus.com/basic_string) for more info on the template).  
  
Note that this class handles bytes independently of the encoding used: If used to handle sequences of multi-byte or variable-length characters (such as UTF-8), all members of this class (such as [length](http://www.cplusplus.com/string::length) or [size](http://www.cplusplus.com/string::size)), as well as its iterators, will still operate in terms of bytes (not actual encoded characters).

### Member types

|  |  |
| --- | --- |
| **member type** | **definition** |
| value\_type | char |
| traits\_type | [char\_traits](http://www.cplusplus.com/char_traits)<char> |
| allocator\_type | [allocator](http://www.cplusplus.com/allocator)<char> |
| reference | char& |
| const\_reference | const char& |
| pointer | char\* |
| const\_pointer | const char\* |
| iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to char (convertible to const\_iterator) |
| const\_iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to const char |
| 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 | [ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) |
| size\_type | [size\_t](http://www.cplusplus.com/size_t) |

### Member functions

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

Construct string object (public member function )

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

String destructor (public member function )

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

String assignment (public member function )

**Iterators**:

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

Return iterator to beginning (public member function )

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

Return iterator to end (public member function )

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

Return reverse iterator to reverse beginning (public member function )

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

Return reverse iterator to reverse end (public member function )

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

Return const\_iterator to beginning (public member function )

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

Return const\_iterator to end (public member function )

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

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

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

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

**Capacity**:

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

Return length of string (public member function )

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

Return length of string (public member function )

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

Return maximum size of string (public member function )

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

Resize string (public member function )

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

Return size of allocated storage (public member function )

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

Request a change in capacity (public member function )

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

Clear string (public member function )

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

Test if string is empty (public member function )

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

Shrink to fit (public member function )

**Element access**:

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

Get character of string (public member function )

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

Get character in string (public member function )

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

Access last character (public member function )

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

Access first character (public member function )

**Modifiers**:

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

Append to string (public member function )

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

Append to string (public member function )

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

Append character to string (public member function )

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

Assign content to string (public member function )

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

Insert into string (public member function )

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

Erase characters from string (public member function )

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

Replace portion of string (public member function )

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

Swap string values (public member function )

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

Delete last character (public member function )

**String operations**:

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

Get C string equivalent (public member function )

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

Get string data (public member function )

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

Get allocator (public member function )

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

Copy sequence of characters from string (public member function )

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

Find content in string (public member function )

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

Find last occurrence of content in string (public member function )

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

Find character in string (public member function )

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

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

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

Find absence of character in string (public member function )

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

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

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

Generate substring (public member function )

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

Compare strings (public member function )

### Member constants

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

Maximum value for size\_t (public static member constant )

### Non-member function overloads

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

Concatenate strings (function )

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

Relational operators for string (function )

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

Exchanges the values of two strings (function )

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

Extract string from stream (function )

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

Insert string into stream (function )

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

Get line from stream into string (function )

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

**Construct string object**

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

|  |  |
| --- | --- |
| ***default (1)*** | **string();** |
| ***copy (2)*** | **string (const string& str);** |
| ***substring (3)*** | **string (const string& str, size\_t pos, size\_t len = npos);** |
| ***from c-string (4)*** | **string (const char\* s);** |
| ***from buffer (5)*** | **string (const char\* s, size\_t n);** |
| ***fill (6)*** | **string (size\_t n, char c);** |
| ***range (7)*** | **template <class InputIterator>**  **string (InputIterator first, InputIterator last);** |
| ***initializer list (8)*** | **string (initializer\_list<char> il);** |
| ***move (9)*** | **string (string&& str) noexcept;** |

***(1) empty string constructor (default constructor)***

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

***(2) copy constructor***

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

***(4) from c-string***

Copies the null-terminated character sequence (C-string) pointed by *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 constructor***

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

### 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");  std::string s5 ("Another character sequence", 12);  std::string s6a (10, 'x');  std::string s6b (10, 42); // 42 is the ASCII code for '\*'  std::string s7 (s0.begin(), s0.begin()+7);  std::cout << "s1: " << s1 << "\ns2: " << s2 << "\ns3: " << s3;  std::cout << "\ns4: " << s4 << "\ns5: " << s5 << "\ns6a: " << s6a;  std::cout << "\ns6b: " << s6b << "\ns7: " << s7 << '\n';  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/string/) |

Output:

|  |
| --- |
| s1:  s2: Initial string  s3: str  s4: A character sequence  s5: Another char  s6a: xxxxxxxxxx  s6b: \*\*\*\*\*\*\*\*\*\*  s7: Initial |

# std::[string](http://www.cplusplus.com/reference/string/string/)::operator=

**String assignment**

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

|  |  |
| --- | --- |
| ***string (1)*** | **string& operator= (const string& str);** |
| ***c-string (2)*** | **string& operator= (const char\* s);** |
| ***character (3)*** | **string& operator= (char c);** |
| ***initializer list (4)*** | **string& operator= (initializer\_list<char> il);** |
| ***move (5)*** | **string& operator= (string&& str) noexcept;** |

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/operator=/) |

Output:

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

# std::string::iterator

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/begin/) |

Output:

|  |
| --- |
| Test string |

# std::[string](http://www.cplusplus.com/reference/string/string/)::size

**Return length of string**

Returns the length of the string, in terms of bytes.  
  
This is the number of actual bytes that conform the contents of the [string](http://www.cplusplus.com/string), which is not necessarily equal to its storage [capacity](http://www.cplusplus.com/string::capacity).

Both **string::size** and [**string::length**](http://www.cplusplus.com/string::length)are synonyms and return the same value.

**size\_t size() const noexcept;**

### Return Value

The number of bytes in the string.

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### 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() << " bytes.\n";  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/size/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::length

**Return length of string**

Returns the length of the string, in terms of bytes.  
  
This is the number of actual bytes that conform the contents of the [string](http://www.cplusplus.com/string), which is not necessarily equal to its storage [capacity](http://www.cplusplus.com/string::capacity).

Both [string::size](http://www.cplusplus.com/string::size) and string::length are synonyms and return the exact same value.

### 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() << " bytes.\n";  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/length/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::max\_size

**Return maximum size of string**

Returns the maximum length the [string](http://www.cplusplus.com/string) can reach.  
  
This is the maximum potential [length](http://www.cplusplus.com/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.

**size\_t max\_size() const noexcept;**

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/max_size/) |

A possible output for this program could be:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::capacity

**Return size of allocated storage**

Returns the size of the storage space currently allocated for the [string](http://www.cplusplus.com/string), expressed in terms of bytes.  
  
This *capacity* is not necessarily equal to the [string length](http://www.cplusplus.com/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 [string](http://www.cplusplus.com/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/string::length) of a [string](http://www.cplusplus.com/string) is given by member [max\_size](http://www.cplusplus.com/string::max_size).

The *capacity* of a [string](http://www.cplusplus.com/string) can be explicitly altered by calling member [reserve](http://www.cplusplus.com/string::reserve).

# std::[string](http://www.cplusplus.com/reference/string/string/)::reserve

**Request a change in capacity**

Requests that the [string capacity](http://www.cplusplus.com/string::capacity) be adapted to a planned change in [size](http://www.cplusplus.com/string::size) to a [length](http://www.cplusplus.com/string::length) of up to *n* characters.  
  
If *n* is greater than the current [string capacity](http://www.cplusplus.com/string::capacity), the function causes the container to increase its [capacity](http://www.cplusplus.com/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/string::capacity): the container implementation is free to optimize otherwise and leave the [string](http://www.cplusplus.com/string) with a [capacity](http://www.cplusplus.com/string::capacity) greater than *n*.  
  
This function has no effect on the [string length](http://www.cplusplus.com/string::length) and cannot alter its content.

### 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 [string](http://www.cplusplus.com/string) object to store an entire file, which is then read character by character. By reserving a [capacity](http://www.cplusplus.com/string::capacity) for the [string](http://www.cplusplus.com/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/string::length) surpass its [capacity](http://www.cplusplus.com/string::capacity).

# std::[string](http://www.cplusplus.com/reference/string/string/)::clear

**Clear string**

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

**void clear() noexcept;**

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/clear/) |

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::empty

**Test if string is empty**

Returns whether the [string](http://www.cplusplus.com/string) is empty (i.e. whether its [length](http://www.cplusplus.com/string::length) is 0).  
This function does not modify the value of the string in any way.

**bool empty() const noexcept;**

### Return Value

true if the [string length](http://www.cplusplus.com/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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/empty/) |

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::shrink\_to\_fit

**Shrink to fit**

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

**void shrink\_to\_fit();**

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/shrink_to_fit/) |

Possible output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::resize

**Resize string**

Resizes the string to a [length](http://www.cplusplus.com/string::length) of *n* characters.  
  
If *n* is smaller than the current [string length](http://www.cplusplus.com/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/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).

**void resize (size\_t n);**

**void resize (size\_t n, char c);**

### Parameters

n

New [string length](http://www.cplusplus.com/string::length), expressed in number of characters.  
[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

c

Character used to fill the new character space added to the string (in case the string is expanded).

### 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';  unsigned sz = str.size();  str.resize (sz+2,'+');  std::cout << str << '\n';  str.resize (14);  std::cout << str << '\n';  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/resize/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::back [C++11]

**Access last character**

Returns a reference to the last character of the [string](http://www.cplusplus.com/string).  
  
This function shall not be called on [empty strings](http://www.cplusplus.com/string::empty).  
 **char& back();**

**const char& back() const;**

### Parameters

none

### Return value

A reference to the last character in the [string](http://www.cplusplus.com/string).  
  
If the [string](http://www.cplusplus.com/string) object is const-qualified, the function returns a const char&. Otherwise, it returns a char&.

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/back/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::front

char& front();

const char& front() const;

**Access first character**

Returns a reference to the first character of the [string](http://www.cplusplus.com/string).  
  
Unlike member [string::begin](http://www.cplusplus.com/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/string::empty).

### Parameters

none

### Return value

A reference to the first character in the [string](http://www.cplusplus.com/string).  
  
If the [string](http://www.cplusplus.com/string) object is const-qualified, the function returns a const char&. Otherwise, it returns a char&.

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/front/) |

Output:

|  |
| --- |
| Test string |

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

**Append to string**

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

|  |  |
| --- | --- |
| ***string (1)*** | **string& operator+= (const string& str);** |
| ***c-string (2)*** | **string& operator+= (const char\* s);** |
| ***character (3)*** | **string& operator+= (char c);** |
| ***initializer list (4)*** | **string& operator+= (initializer\_list<char> il);** |

### 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;  name += {'a','b','c'}; // Initializer list  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/operator+=/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::append

**Append to string**

Extends the [string](http://www.cplusplus.com/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 [string::npos](http://www.cplusplus.com/string::npos)).

***(3) c-string***

Appends a copy of the string formed by the null-terminated character sequence (C-string) pointed by *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.

|  |  |
| --- | --- |
| ***string (1)*** | **string& append (const string& str);** |
| ***substring (2)*** | **string& append (const string& str, size\_t subpos, size\_t sublen);** |
| ***c-string (3)*** | **string& append (const char\* s);** |
| ***buffer (4)*** | **string& append (const char\* s, size\_t n);** |
| ***fill (5)*** | **string& append (size\_t n, char c);** |
| ***range (6)*** | **template <class InputIterator>**  **string& append (InputIterator first, InputIterator last);** |
| ***initializer list(7)*** | **string& append (initializer\_list<char> il);** |

### Parameters

str

Another [string](http://www.cplusplus.com/string) object, 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/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 [string::npos](http://www.cplusplus.com/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 char.  
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.

[size\_t](http://www.cplusplus.com/size_t) 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/append/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::push\_back

void push\_back (char c);

**Append character to string**

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

### Parameters

c

Character added to the [string](http://www.cplusplus.com/string).

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/push_back/) |

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::assign

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

***(3) c-string***

Copies the null-terminated character sequence (C-string) pointed by *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.

|  |  |
| --- | --- |
| ***string (1)*** | **string& assign (const string& str);** |
| ***substring (2)*** | **string& assign (const string& str, size\_t subpos, size\_t sublen);** |
| ***c-string (3)*** | **string& assign (const char\* s);** |
| ***buffer (4)*** | **string& assign (const char\* s, size\_t n);** |
| ***fill (5)*** | **string& assign (size\_t n, char c);** |
| ***range (6)*** | **template <class InputIterator>**  **string& assign (InputIterator first, InputIterator last);** |
| ***initializer list(7)*** | **string& assign (initializer\_list<char> il);** |
| ***move (8)*** | **string& assign (string&& str) noexcept;** |

### Parameters

str

Another [string](http://www.cplusplus.com/string) object, 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/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 [string::npos](http://www.cplusplus.com/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 char.  
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.

[size\_t](http://www.cplusplus.com/size_t) 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/assign/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::insert

**Insert into string**

Inserts additional characters into the [string](http://www.cplusplus.com/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/string::npos)).

***(3) c-string***

Inserts a copy of the string formed by the null-terminated character sequence (C-string) pointed by *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.

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

|  |  |
| --- | --- |
| ***string (1)*** | **string& insert (size\_t pos, const string& str);** |
| ***substring (2)*** | **string& insert (size\_t pos, const string& str, size\_t subpos, size\_t sublen);** |
| ***c-string (3)*** | **string& insert (size\_t pos, const char\* s);** |
| ***buffer (4)*** | **string& insert (size\_t pos, const char\* s, size\_t n);** |
| ***fill (5)*** | **string& insert (size\_t pos, size\_t n, char c);**  **iterator insert (const\_iterator p, size\_t n, char c);** |
| ***single character (6)*** | **iterator insert (const\_iterator p, char c);** |
| ***range (7)*** | **template <class InputIterator>**  **iterator insert (iterator p, InputIterator first, InputIterator last);** |
| ***initializer list (8)*** | **string& insert (const\_iterator p, initializer\_list<char> il);** |

### 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/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 [string](http://www.cplusplus.com/string) object.

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/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/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 [string](http://www.cplusplus.com/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 char.

il

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

### Return value

The signatures returning a reference to [string](http://www.cplusplus.com/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 [string](http://www.cplusplus.com/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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/insert/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::erase

**Erase characters from string**

Erases part of the [string](http://www.cplusplus.com/string), reducing its [length](http://www.cplusplus.com/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 [string::npos](http://www.cplusplus.com/string::npos).  
Notice that the default argument erases all characters in the string (like member function [clear](http://www.cplusplus.com/string::clear)).

***(2) character***

Erases the character pointed by *p*.

***(3) range***

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

|  |  |
| --- | --- |
| ***sequence (1)*** | **string& erase (size\_t pos = 0, size\_t len = npos);** |
| ***character (2)*** | **iterator erase (const\_iterator p);** |
| ***range (3)*** | **iterator erase (const\_iterator first, const\_iterator last);** |

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/erase/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::replace

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

***(3) c-string***

Copies the null-terminated character sequence (C-string) pointed by *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.

|  |  |
| --- | --- |
| *string (1)* | string& replace (size\_t pos, size\_t len, const string& str);  string& replace (const\_iterator i1, const\_iterator i2, const string& str); |
| *substring (2)* | string& replace (size\_t pos, size\_t len, const string& str,  size\_t subpos, size\_t sublen); |
| *c-string (3)* | string& replace (size\_t pos, size\_t len, const char\* s);  string& replace (const\_iterator i1, const\_iterator i2, const char\* s); |
| *buffer (4)* | string& replace (size\_t pos, size\_t len, const char\* s, size\_t n);  string& replace (const\_iterator i1, const\_iterator i2, const char\* s, size\_t n); |
| *fill (5)* | string& replace (size\_t pos, size\_t len, size\_t n, char c);  string& replace (const\_iterator i1, const\_iterator i2, size\_t n, char c); |
| *range (6)* | template <class InputIterator>  string& replace (const\_iterator i1, const\_iterator i2,  InputIterator first, InputIterator last); |
| *initializer list (7)* | string& replace (const\_iterator i1, const\_iterator i2, initializer\_list<char> il); |

### Parameters

str

Another [string](http://www.cplusplus.com/string) object, 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/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 [string::npos](http://www.cplusplus.com/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/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 [string::npos](http://www.cplusplus.com/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 char.

il

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

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/replace/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::swap

**Swap string values**

Exchanges the content of the container by the content of *str*, which is another [string](http://www.cplusplus.com/string) object. [Lengths](http://www.cplusplus.com/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.

**void swap (string& str);**

### Parameters

str

Another [string](http://www.cplusplus.com/string) object, whose value is swapped with that of this [string](http://www.cplusplus.com/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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/swap/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::pop\_back

**Delete last character**

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

**void pop\_back();**

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/pop_back/) |

|  |
| --- |
| hello world |

# std::[string](http://www.cplusplus.com/reference/string/string/)::c\_str

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

This array includes the same sequence of characters that make up the value of the [string](http://www.cplusplus.com/string) object plus an additional terminating null-character ('\0') at the end.

**const char\* c\_str() const;**

### 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 = std::strtok(NULL," ");  }  delete[] cstr;  return 0;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/c_str/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::copy

**Copy sequence of characters from string**

Copies a substring of the current value of the [string](http://www.cplusplus.com/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.

**size\_t copy (char\* s, size\_t len, size\_t pos = 0) const;**

### 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/string::length), it throws [out\_of\_range](http://www.cplusplus.com/out_of_range).  
Note: The first character in the [string](http://www.cplusplus.com/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/string::length)-pos (if the string value is shorter than pos+len).  
  
[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/copy/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::find

**Find content in string**

Searches the [string](http://www.cplusplus.com/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*.  
  
Notice that unlike member [find\_first\_of](http://www.cplusplus.com/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.

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

### Parameters

str

Another [string](http://www.cplusplus.com/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/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.

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### Return Value

The position of the first character of the first match.  
If no matches were found, the function returns [**string::npos**](http://www.cplusplus.com/string::npos)**.**  
  
[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### 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> // std::cout  #include <string> // std::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::size\_t 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/find/) |

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. |

# std::[string](http://www.cplusplus.com/reference/string/string/)::substr

**Generate substring**

Returns a newly constructed [string](http://www.cplusplus.com/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).

**string substr (size\_t pos = 0, size\_t len = npos) const;**

### 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/string::length), the function returns an empty string.  
If this is greater than the [*string length*](http://www.cplusplus.com/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 [string::npos](http://www.cplusplus.com/string::npos) indicates all characters until the end of the string.

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### Return Value

A [string](http://www.cplusplus.com/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 (3,5); // "think"  std::size\_t 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/substr/) |

Output:

|  |
| --- |
| think live in details. |

# std::[string](http://www.cplusplus.com/reference/string/string/)::find\_first\_of

**Find character in string**

Searches the [string](http://www.cplusplus.com/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 [string::find](http://www.cplusplus.com/string::find) for a function that matches entire sequences.

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

### Parameters

str

Another [string](http://www.cplusplus.com/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/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.

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### Return Value

The position of the first character that matches.  
If no matches are found, the function returns [string::npos](http://www.cplusplus.com/string::npos).  
  
[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### Example

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | // string::find\_first\_of  #include <iostream> // std::cout  #include <string> // std::string  #include <cstddef> // std::size\_t  int main ()  {  std::string str ("Please, replace the vowels in this sentence by asterisks.");  std::size\_t 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/find_first_of/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::find\_last\_of

### 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::find\_last\_of  #include <iostream> // std::cout  #include <string> // std::string  #include <cstddef> // std::size\_t  void SplitFilename (const std::string& str)  {  std::cout << "Splitting: " << str << '\n';  std::size\_t 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/find_last_of/) |

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::find\_first\_not\_of

**Find absence of character in string**

Searches the [string](http://www.cplusplus.com/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.

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

### Parameters

str

Another [string](http://www.cplusplus.com/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/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.

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### Return Value

The position of the first character that does not match.  
If no such characters are found, the function returns [string::npos](http://www.cplusplus.com/string::npos).

### Example

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | // string::find\_first\_not\_of  #include <iostream> // std::cout  #include <string> // std::string  #include <cstddef> // std::size\_t  int main ()  {  std::string str ("look for non-alphabetic characters...");  std::size\_t 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/find_first_not_of/) |

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::find\_last\_not\_of

### Example

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | // string::find\_last\_not\_of  #include <iostream> // std::cout  #include <string> // std::string  #include <cstddef> // std::size\_t  int main ()  {  std::string str ("Please, erase trailing white-spaces \n");  std::string whitespaces (" \t\f\v\n\r");  std::size\_t 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/find_last_not_of/) |

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::compare

**Compare strings**

Compares the value of the [string](http://www.cplusplus.com/string) object (or a substring) to the sequence of characters specified by its arguments.  
  
The *compared string* is the value of the [string](http://www.cplusplus.com/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.

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

### Parameters

str

Another [string](http://www.cplusplus.com/string) object, 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/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 [string::npos](http://www.cplusplus.com/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.

[size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type (the same as member type [string::size\_type](http://www.cplusplus.com/string#types)).

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/compare/) |

Output:

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

# std::[string](http://www.cplusplus.com/reference/string/string/)::npos

**Maximum value for size\_t**

npos is a static member constant value with the greatest possible value for an element of type [size\_t](http://www.cplusplus.com/size_t).  
  
This value, when used as the value for a *len* (or *sublen*) parameter in [string](http://www.cplusplus.com/string)'s member functions, means *"until the end of the string"*.  
  
As a return value, it is usually used to indicate no matches.  
  
This constant is defined with a value of -1, which because [size\_t](http://www.cplusplus.com/size_t) is an unsigned integral type, it is the largest possible representable value for this type.

**static const size\_t npos = -1;**

# Non-member function overloads

## operator+ (string)

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

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/operator+/) |

Output:

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

## relational operators (string)

**Relational operators for string**

Performs the appropriate comparison operation between the [string](http://www.cplusplus.com/string) objects *lhs* and *rhs*.  
  
The functions use [string::compare](http://www.cplusplus.com/string::compare) for the comparison.  
  
These operators are overloaded in header [<string>](http://www.cplusplus.com/%3Cstring%3E).

|  |  |
| --- | --- |
| *(1)* | bool operator== (const string& lhs, const string& rhs);  bool operator== (const char\* lhs, const string& rhs);  bool operator== (const string& lhs, const char\* rhs); |
| *(2)* | bool operator!= (const string& lhs, const string& rhs);  bool operator!= (const char\* lhs, const string& rhs);  bool operator!= (const string& lhs, const char\* rhs); |
| *(3)* | bool operator< (const string& lhs, const string& rhs);  bool operator< (const char\* lhs, const string& rhs);  bool operator< (const string& lhs, const char\* rhs); |
| *(4)* | bool operator<= (const string& lhs, const string& rhs);  bool operator<= (const char\* lhs, const string& rhs);  bool operator<= (const string& lhs, const char\* rhs); |
| *(5)* | bool operator> (const string& lhs, const string& rhs);  bool operator> (const char\* lhs, const string& rhs);  bool operator> (const string& lhs, const char\* rhs); |
| *(6)* | bool operator>= (const string& lhs, const string& rhs);  bool operator>= (const char\* lhs, const string& rhs);  bool operator>= (const string& lhs, const char\* rhs); |

### Example

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | // string comparisons  #include <iostream>  #include <string>  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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/operators/) |

Output:

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

## operator>> (string)

**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 [istream::operator>>](http://www.cplusplus.com/istream::operator%3E%3E) for c-strings, but applied to [string](http://www.cplusplus.com/string)objects.  
  
Each extracted character is appended to the [string](http://www.cplusplus.com/string) as if its member [push\_back](http://www.cplusplus.com/string::push_back) was called.  
  
Notice that the [istream](http://www.cplusplus.com/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 [string](http://www.cplusplus.com/string) overload of global function [getline](http://www.cplusplus.com/string:getline).

**istream& operator>> (istream& is, string& str);**

### Parameters

is

[istream](http://www.cplusplus.com/istream) object from which characters are extracted.

str

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

### 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;  } | [Edit & Run](http://www.cplusplus.com/reference/string/string/operator%3E%3E/) |

# operator<< (string)

**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 [ostream::operator<<](http://www.cplusplus.com/ostream::operator%3C%3C) for c-strings, but applied to [string](http://www.cplusplus.com/string) objects.

**ostream& operator<< (ostream& os, const string& str);**

### Parameters

os

[ostream](http://www.cplusplus.com/ostream) object where characters are inserted.

str

[string](http://www.cplusplus.com/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 [ios::exceptions](http://www.cplusplus.com/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;  } |

# getline (string)

**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, '\n', 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).  
  
Note that any content in *str* before the call is replaced by the newly extracted sequence.  
  
Each extracted character is appended to the [string](http://www.cplusplus.com/string) as if its member [push\_back](http://www.cplusplus.com/string::push_back) was called.

|  |  |
| --- | --- |
| ***(1)*** | **istream& getline (istream& is, string& str, char delim);** |
| ***(2)*** | **istream& getline (istream& is, string& str);** |

### Parameters

is

[istream](http://www.cplusplus.com/istream) object from which characters are extracted.

str

[string](http://www.cplusplus.com/string) object where the extracted line is stored.  
The contents in the string before the call (if any) are discarded and replaced by the extracted line.

### 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 [ios::exceptions](http://www.cplusplus.com/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>  int main ()  {  std::string name;  std::cout << "Please, enter your full name: ";  std::getline (std::cin,name);  std::cout << "Hello, " << name << "!\n";  return 0;  } |