**String**

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

str.append(str2); // "Writing "

**string::back** : Returns a reference to the last character of the string.

std::string str ("hello world.");

str.back() = '!';

**string::begin** : Returns an iterator pointing to the first character of the string.

**string::end :** Returns an iterator pointing to the *past-the-end* character of the string.

for ( std::string::iterator it=str.begin(); it!=str.end(); ++it)

std::cout << \*it;

**string::capacity :** Returns the size of the storage space currently allocated for the string, expressed in terms of bytes.This capacity is not necessarily equal to the 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.

std::cout << "capacity: " << str.capacity() << "\n";

**string::cbegin :** Returns a const\_iterator pointing to the first character of the string.

**string::cend :** 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. it cannot be used to modify the contents it points to, even if the [string](http://www.cplusplus.com/string) object is not itself const.

for (auto it=str.cbegin(); it!=str.cend(); ++it)

std::cout << \*it;

**string::clear :** 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).

str.clear();

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

if (str1.compare(str2) == 0)

std::cout << str1 << " is equal to " << str2 << '\n';

**string::copy (** size\_t copy (char\* s, size\_t len, size\_t pos = 0) const**)**: Copy sequence of characters from string. Copies a substring of the current value of the string object into the array pointed by s. This substring contains the len characters that start at position pos.

char buffer[20];

std::string str ("Test string...");

std::size\_t length = str.copy(buffer,6,5);

buffer[length]='\0';

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

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

for (auto rit=str.crbegin(); rit!=str.crend(); ++rit)

std::cout << \*rit;

**string::c\_str :** 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.

#include <cstring>

#include <string>

std::string str ("Please split this sentence into tokens");

char \* cstr = new char [str.length()+1];

std::strcpy (cstr, str.c\_str());

**string::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).

if(str.empty())

**string::erase :** Erases part of the [string](http://www.cplusplus.com/string), reducing its [length](http://www.cplusplus.com/string::length)

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."

**string::find :** 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*.

std::size\_t found = str.find(str2);

if (found!=std::string::npos)

std::cout << "first 'needle' found at: " << found << '\n';

found=str.find('.');

if (found!=std::string::npos)

std::cout << "Period found at: " << found << '\n';

**string::front** : Returns a reference to the first character of the string. Unlike member 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.

std::string str ("test string");

str.front() = 'T';

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

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 thequestion**

str.insert (str.end(),3,'.'); //to be, not to be: that is thequestion(...)

str.insert (it+2,str3.begin(),str3.begin()+3); // (or )

**string::length :** Returns the length of the string, in terms of bytes.

std::cout << "The size of str is " << str.length() << " bytes.\n";

**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.

std::cout << "max\_size: " << str.max\_size() << "\n";

max\_size: 4294967291

**string::operator+= : E**xtends the [string](http://www.cplusplus.com/string) by appending additional characters at the end of its current value:

std::string name ("John");

std::string family ("Smith");

name += " K. "; // c-string

name += family; // string

name += '\n'; // character

**string::operator= :** Assigns a new value to the string, replacing its current contents.

str1 = "Test string: "; // c-string

str2 = 'x'; // single character

str3 = str1 + str2; // string

**string::operator[] :** Returns a reference to the character at position *pos* in the [string](http://www.cplusplus.com/string).

for (int i=0; i<str.length(); ++i)

{

std::cout << str[i];

}

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

str.pop\_back();

**string::push\_back :** 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.

// string::push\_back

#include <iostream>

#include <fstream>

#include <string>

std::string str;

std::ifstream file ("test.txt",std::ios::in);

if (file) {

while (!file.eof()) str.push\_back(file.get());

}

**string::rbegin :** Return reverse iterator to reverse beginning

Returns a reverse iterator pointing to the last character of the string (i.e., its reverse beginning).

**string::rend :** 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).

for (std::string::reverse\_iterator rit=str.rbegin(); rit!=str.rend(); ++rit)

std::cout << \*rit;

**string::replace :** 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:

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)

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

std::size\_t found = str.rfind(key);

if (found!=std::string::npos)

str.replace (found,key.length(),"seventh");

**string::size :** 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).

std::cout << "The size of str is " << str.size() << " bytes.\n";

**string::substr :** 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;

std::string str="We think in generalities, but we live in details.";

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'; // think live in details.

**std::string::swap :** 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. Notice that a non-member function exists with the same name, [swap](http://www.cplusplus.com/string:swap), overloading that algorithm with an optimization that behaves like this member function.

std::string buyer ("money");

std::string seller ("goods");

seller.swap (buyer);

**string::npos :** static const size\_t npos = -1; Maximum value for size\_t

npos is a static member constant value with the greatest possible value for an element of type size\_t.

This value, when used as the value for a len (or sublen) parameter in 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 is an unsigned integral type, it is the largest possible representable value for this type.

**std::getline (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 as if its member push\_back was called. std::getline (std::cin,name);