C++ Programming/Code/IO/Streams/string

< [C++ Programming](https://en.wikibooks.org/wiki/C%2B%2B_Programming)‎ | [Code/IO](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO)

[Jump to navigation](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#mw-head)[Jump to search](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#p-search)



**Contents**

* [1The string class](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#The_string_class)
  + [1.1Basic usage](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Basic_usage)
  + [1.2Text I/O](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Text_I/O)
    - [1.2.1Getting user input](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Getting_user_input)
  + [1.3More advanced string manipulation](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#More_advanced_string_manipulation)
    - [1.3.1Size](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Size)
    - [1.3.2I/O](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#I/O)
    - [1.3.3Operators](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Operators)
    - [1.3.4Searching strings](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Searching_strings)
    - [1.3.5Inserting/erasing](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Inserting/erasing)
    - [1.3.6Backwards compatibility](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Backwards_compatibility)
  + [1.4String Formatting](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#String_Formatting)
    - [1.4.1Example](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Example)
  + [1.5Advanced use](https://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/IO/Streams/string#Advanced_use)

The string class[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=1" \o "Edit section: The string class)]

The string class is a part of the C++ standard library, used for convenient manipulation of sequences of characters, to replace the static, unsafe C method of handling strings. To use the string class in a program, the <string> header must be included. The standard library string class can be accessed through the ***std namespace***.

The basic template class is basic\_string<> and its standard specializations are string and wstring.

**Basic usage**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=2" \o "Edit section: Basic usage)]

Declaring a *std* string is done by using one of these two methods:

**using** **namespace** std;

string std\_string;

or

std::string std\_string;

**Text I/O**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=3" \o "Edit section: Text I/O)]

This section will deal only with keyboard and text input. There are many other inputs that can be read (mouse movements and button clicks, etc), but these will not be covered in this section, even reading the special keys of the keyboard will be excluded.

Perhaps the most basic use of the string class is for reading text from the user and writing it to the screen. In the header file iostream, C++ defines an object named cin that handles input in much the same way that cout handles output.

*// snipped designed to get an integer value from the user*

int x;

std::cin >> x;

The >> operator will cause the execution to stop and will wait for the user to type something. If the user types a valid integer, it will be converted into an integer value and stored in x.

If the user types something other than an integer, the compiler will not report an error. Instead, it leaves the old content (a "random" meaningless value) in x and continues.

This can then be extended into the following program:

#include *<iostream>*

#include *<string>*

int main(){

std::string name;

std::cout << "Please enter your first name: ";

std::cin >> name;

std::cout << "Welcome " << name << "!" << std::endl;

**return** 0;

}

Although a string may hold a sequence containing any character—including spaces and nulls—when reading into a string using cin and the extraction operator (>>) only the characters before the first space will be stored. Alternatively, if an entire line of text is desired, the *getline* function may be used:

std::getline(std::cin, name);

**Getting user input**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=4" \o "Edit section: Getting user input)]

Fortunately, there is a way to check and see if an input statement succeeds. We can invoke the good function on cin to check what is called the stream state. good returns a bool: if true, then the last input statement succeeded. If not, we know that some previous operation failed, and also that the next operation will fail.

Thus, getting input from the user might look like this:

#include *<iostream>*

**using** **namespace** std;

int main ()

{

int x;

*// prompt the user for input*

cout << "Enter an integer: ";

*// get input*

cin >> x;

*// check and see if the input statement succeeded*

**if** (cin.good() == false) {

cout << "That was not an integer." << endl;

**return** -1;

}

*// print the value we got from the user*

cout << x << endl;

**return** 0;

}

cin can also be used to input a string:

string name;

cout << "What is your name? ";

cin >> name;

cout << name << endl;

As with the scanf() function from the Standard C Library, this statement only takes the first word of input, and leaves the rest for the next input statement. So, if you run this program and type your full name, it will only output your first name.

You may also notice the >> operator doesn't handle errors as expected (for example, if you accidentally typed your name in a prompt for a number.) Because of these issues, it may be more suitable to read a line of text, and using the line for input — this is performed using the function called getline.

string name;

cout << "What is your name? ";

getline (cin, name);

cout << name << endl;

The first argument to getline is cin, which is where the input is coming from. The second argument is the name of the string variable where you want the result to be stored.

getline reads the entire line until the user hits Return or Enter. This is useful for inputting strings that contain spaces.

In fact, getline is generally useful for getting input of any kind. For example, if you wanted the user to type an integer, you could input a string and then check to see if it is a valid integer. If so, you can convert it to an integer value. If not, you can print an error message and ask the user to try again.

To convert a string to an integer you can use the strtol function defined in the header file cstdlib. (Note that the older function atoi is less safe than strtol, as well as being less capable.)

If you still need the features of the >> operator, you will need to create a string stream as available from <sstream>. The use of this stream will be discussed in a later chapter.

**More advanced string manipulation**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=5" \o "Edit section: More advanced string manipulation)]

|  |  |
| --- | --- |
| Clipboard | **To do:** Detail the commonly used std::string member functions(partially done) |

We will be using this dummy string for some of our examples.

string str("Hello World!");

This invokes the default constructor with a const char\* argument. Default constructor creates a string which contains nothing, i.e. no characters, not even a '\0' (however std::string is not null terminated).

string str2(str);

Will trigger the copy constructor. std::string knows enough to make a deep copy of the characters it stores.

string str2 = str;

This will copy strings using assignment operator. Effect of this code is same as using copy constructor in example above.

**Size**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=6" \o "Edit section: Size)]

string::size\_type string::size() **const**;

string::size\_type string::length() **const**;

So for example one might do:

string::size\_type strSize = str.size();

string::size\_type strSize2 = str2.length();

The methods size() and length() both return the size of the string object. There is no apparent difference. Remember that the last character in the string is size() - 1 and not size(). Like in C-style strings, and arrays in general, std::string starts counting from 0.

**I/O**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=7" \o "Edit section: I/O)]

ostream& **operator**<<(ostream &out, string &str);

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

The shift operators (>> and <<) have been overloaded so you can perform I/O operations on istream and ostream objects, most notably cout, cin, and filestreams. Thus you could just do console I/O like this:

std::cout << str << endl;

std::cin >> str;

istream& getline (istream& in, string& str, char delim = '\n');

Alternatively, if you want to read entire lines at a time, use getline(). Note that this is not a member function. getline() will retrieve characters from input stream in and assign them to str until EOF is reached or delim is encountered. getline will reset the input string before appending data to it. delim can be set to any char value and acts as a general delimiter. Here is some example usage:

#include *<fstream>*

*//open a file*

std::ifstream file("somefile.cpp");

std::string data, temp;

**while**( getline(file, temp, '#')) *//while data left in file*

{

*//append data*

data += temp;

}

std::cout << data;

Because of the way getline works (i.e. it returns the input stream), you can nest multiple getline() calls to get multiple strings; however this may significantly reduce readability.

**Operators**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=8" \o "Edit section: Operators)]

char& string::**operator**[](string::size\_type pos);

Chars in strings can be accessed directly using the overloaded subscript ([]) operator, like in char arrays:

std::cout << str[0] << str[2];

prints "Hl".

std::string supports casting from the older C string type const char\*. You can also assign or append a simple char to a string. Assigning a char\* to a string is as simple as

str = "Hello World!";

If you want to do it character by character, you can also use

str = 'H';

Not surprisingly, operator+ and operator+= are also defined! You can append another string, a const char\* or a char to any string.

The comparison operators >, <, ==, >=, <=, != all perform comparison operations on strings, similar to the C strcmp() function. These return a true/false value.

**if**(str == "Hello World!")

{

std::cout << "Strings are equal!";

}

**Searching strings**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=9" \o "Edit section: Searching strings)]

string::size\_type string::find(string needle, string::size\_type pos = 0) **const**;

You can use the find() member function to find the first occurrence of a string inside another. find() will look for needle inside this starting from position pos and return the position of the first occurrence of the needle. For example:

std::string haystack = "Hello World!";

std::string needle = "o";

std::cout << haystack.find(needle);

Will simply print "4" which is the index of the first occurrence of "o" in str. If we want the "o" in "World", we need to modify pos to point past the first occurrence. str.find(find, 4) would return 4, while str.find(find, 5) would give 7. If the substring isn't found, find() returns std::string::npos.This simple code searches a string for all occurrences of "wiki" and prints their positions:

std::string wikistr = "wikipedia is full of wikis (wiki-wiki means fast)";

**for**(string::size\_type i = 0, tfind; (tfind = wikistr.find("wiki", i)) != string::npos; i = tfind + 1)

{

std::cout << "Found occurrence of 'wiki' at position " << tfind << std::endl;

}

string::size\_type string::rfind(string needle, string::size\_type pos = string::npos) **const**;

The function rfind() works similarly, except it returns the *last* occurrence of the passed string.

**Inserting/erasing**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=10" \o "Edit section: Inserting/erasing)]

string& string::insert(size\_type pos, **const** string& str);

You can use the insert() member function to insert another string into a string. For example:

string newstr = " Human";

str.insert (5,newstr);

Would return *Hello Human World!*

string& string::erase(size\_type pos, size\_type n);

You can use erase() to remove a substring from a string. For example:

str.erase (6,11);

Would return *Hello!*

string& string::substr(size\_type pos, size\_type n);

You can use substr() to extract a substring from a string. For example:

string str = "Hello World!";

string part = str.substr(6,5);

Would return *World*.

**Backwards compatibility**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=11" \o "Edit section: Backwards compatibility)]

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

**const** char\* string::data() **const**;

For backwards compatibility with C/C++ functions which only accept char\* parameters, you can use the member functions string::c\_str() and string::data() to return a temporary const char\* string you can pass to a function. The difference between these two functions is that c\_str() returns a null-terminated string while data() does not necessarily return a null-terminated string. So, if your legacy function requires a null-terminated string, use c\_str(), otherwise use data() (and presumably pass the length of the string in as well).

**String Formatting**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=12" \o "Edit section: String Formatting)]

Strings can only be appended to other strings, but not to numbers or other datatypes, so something like std::string("Foo") + 5 would not result in a string with the content "Foo5". To convert other datatypes into string there exist the class std::ostringstream, found in the include file <sstream>. std::ostringstream acts exactly like std::cout, the only difference is that the output doesn't go to the current standard output as provided by the operating system, but into an internal buffer, that buffer can be converted into a std::string via the std::ostringstream::str() method.

**Example**[[edit](https://en.wikibooks.org/w/index.php?title=C%2B%2B_Programming/Code/IO/Streams/string&action=edit&section=13" \o "Edit section: Example)]

#include *<iostream>*

#include *<sstream>*

int main()

{

std::ostringstream buffer;

*// Use the std::ostringstream just like std::cout or other iostreams*

buffer << "You have: " << 5 << " Helloworlds in your inbox";

*// Convert the std::ostringstream to a normal string*

std::string text = buffer.str();

std::cout << text << std::endl;

**return** 0;

}