C++ Memoire

[<iostream> 2](#_Toc74075360)

[printf() vs cout 2](#_Toc74075361)

[<istream> 3](#_Toc74075362)

[<string.h> 4](#_Toc74075363)

[definition 4](#_Toc74075364)

[member functions 4](#_Toc74075365)

[iterators 4](#_Toc74075366)

[capacity 4](#_Toc74075367)

[element access 4](#_Toc74075368)

[modifiers 5](#_Toc74075369)

[string operations 5](#_Toc74075370)

[constants 6](#_Toc74075371)

[overloads 6](#_Toc74075372)

[member types 6](#_Toc74075373)

[<regex> 7](#_Toc74075374)

[Templates 9](#_Toc74075375)

# <iostream>

Standard I/O stream objects header. Mostly defined in *<istream>*.

cin

Object of class *istream*, representing *stdin*, using *char* (narrow, single-byte) characters, providing input from a file, pipe, or keyboard (see: *istream* for operations supported by *cin*). *cin* is tied to *cout*, meaning the *cout* buffer is flushed before each *cin* operation. A program should mix operations on *cin* with operations on *wcin* (or other wide-oriented inputs – when an input operation is performed on a stream, it becomes either *narrow*, or *wide*, unless changed by call to *freopen()*.

cout

cerr

clog

wcin

wcout

wcerr

wclog

## printf() vs cout

<https://stackoverflow.com/questions/2872543/printf-vs-cout-in-c/20238349>

*printf()* is considerably faster than *cout*.

*printf()* cannot (easily) be used to print the null byte.

*cout* provides simpler handling of type.

*cout* (being a class) can be extended.

*cout* provides a ‘simpler’ syntax.

*printf()* provides precise (yet relatively simple) formatting

# [<istream>](http://www.cplusplus.com/reference/istream/istream/)

# [<string.h>](http://www.cplusplus.com/reference/string/string/)

*Use <cstring> for null terminated C string library in C++.*

## definition

typdef basic\_string<char> string

The C++ string is an object that represents a sequence of characters, similar to that of a standard container of bytes, but with added features specifically for single-byte characters. Specifically, *string* is an instantiation of the *basic\_string* class template that uses *char* as its character type, with its default *char\_traits* and allocator types (see: *basic\_string*). Note: the string is stored/processed/iterated in terms of bytes, regardless of text encoding being used.

## member functions

(constructor)

(destructor)

operator=

## iterators

begin()

end()

rbegin()

rend()

cbegin()

cend()

crbegin()

crend()

## capacity

size()

length()

max\_size()

resize()

capacity()

reserve()

clear()

empty()

shrink\_to\_fit()

## element access

operator[]

append()

push\_back()

assign()

insert()

erase()

replace()

swap()

pop\_back()

## modifiers

operator+=

append()

push\_back()

assign()

insert()

erase()

replace()

swap()

pop\_back()

## string operations

c\_str()

data()

get\_allocator()

copy()

find()

rfind()

find\_first\_of()

find\_last\_of()

find\_first\_not\_of()

find\_last\_not\_of()

substr()

compare()

## constants

npos

## overloads

operator+

relational operators

swap()

operator>>

operator<<

getline()

## 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) |

# <regex>

# Templates

Function overloading

cannot be overloaded

1. Function declarations that differ only in return type
2. Member functions if any of them is a static member functions
3. Functions with argument declarations that differ only in pointer/array (since an array as function argument in C/C++ is converted to a pointer as argument)
4. Functions with argument declarations that differ only in that one is a function type, and the other is a pointer to the same function type (that is: *‘void h(int ())’* vs *‘**void h(int (\*)())’*
5. Functions with argument declarations that differ only in present or absence of *const* / *volatile* specifiers.
6. Functions with argument declarations that differ only in their default values

d