C++ Standard Library headers

Standard Library headers

The interface of C++ standard library is defined by the following collection of headers.

Concepts library

<concepts> (C++20)	Fundamental library concepts

Coroutines library

<coroutine> (C++20)	Coroutine support library

Utilities library

<cstdlib>	General purpose utilities: program control, dynamic memory allocation, random numbers, so
<csignal>	Functions and macro constants for signal management
<csetjmp>	Macro (and function) that saves (and jumps) to an execution context
<cstdarg>	Handling of variable length argument lists
<typeinfo>	Runtime type information utilities
<typeindex> (C++11)	std::type_index
<type_traits> (C++11)	Compile-time type information
<bitset>	std::bitset class template
<functional>	Function objects, Function invocations, Bind operations and Reference wrappers
<utility>	Various utility components
<ctime>	C-style time/date utilites
<chrono> (C++11)	C++ time utilites
<cstddef>	Standard macros and typedefs
<initializer_list> (C++11)	std::initializer_list class template
<tuple> (C++11)	std::tuple class template
<any> (C++17)	std::any class
<optional> (C++17)	std::optional class template
<variant> (C++17)	std::variant class template
<compare> (C++20)	Three-way comparison operator support
<version> (C++20)	Supplies implementation-dependent library information
<source_location>	Supplies means to obtain source code location

(C++20)	
Dynamic memory management	
<new>	Low-level memory management utilities
<memory>	High-level memory management utilities
<scoped_allocator> (C++11)	Nested allocator class
<memory_resource> (C++17)	Polymorphic allocators and memory resources
Numeric limits	
<climits>	Limits of integral types
<cfloat>	Limits of floating-point types
<cstdint> (C++11)	Fixed-width integer types and limits of other types
<cinttypes> (C++11)	Formatting macros, `intmax_t` and `uintmax_t` math and conversions
<limits>	Uniform way to query properties of arithmetic types
Error handling	
<exception>	Exception handling utilities
<stdexcept>	Standard exception objects
<cassert>	Conditionally compiled macro that compares its argument to zero
<system_error> (C++11)	Defines std::error_code, a platform-dependent error code
<cerrno>	Macro containing the last error number
Strings library	
<cctype>	Functions to determine the category of narrow characters
<cwctype>	Functions to determine the catagory of wide characters
<cstring>	Various narrow character string handling functions
<cwchar>	Various wide and multibyte string handling functions
<cuchar> (C++11)	C-style Unicode character conversion functions
<string>	std::basic_string class template
<string_view> (C++17)	std::basic_string_view class template
<charconv> (C++17)	std::to_chars and std::from_chars
<format> (C++20)	Formatting library including std::format
Containers library	
<array> (C++11)	std::array container

<vector>	std::vector container
<deque>	std::deque container
<list>	std::list container
<forward_list> (C++11)	std::forward_list container
<set>	std::set and std::multiset associative containers
<map>	std::map and std::multimap associative containers
<unordered_set> (C++11)	std::unordered_set and std::unordered_multiset unordered associative containers
<unordered_map> (C++11)	std::unordered_map and std::unordered_multimap unordered associative containers
<stack>	std::stack container adaptor
<queue>	std::queue and std::priority_queue container adaptors
**** (C++20)	std::span view
Iterators library	
<iterator>	Range iterators
Ranges library	
<ranges> (C++20)	Range access, primitives, requirements, utilities and adaptors
Algorithms library	
<algorithm>	Algorithms that operate on ranges
<execution> (C++17)	Predefined execution policies for parallel versions of the algorithms
Numerics library	
<cmath>	Common mathematics functions
<complex>	Complex number type
<valarray>	Class for representing and manipulating arrays of values
<random> (C++11)	Random number generators and distributions
<numeric>	Numeric operations on values in ranges
<ratio> (C++11)	Compile-time rational arithmetic
<cfenv> (C++11)	Floating-point environment access functions
<bit> (C++20)	Bit manipulation functions
<numbers> (C++20)	Math constants
Localization library	

<locale>	Localization utilities
<clocale>	C localization utilities
<codecvt> (C++11)(deprecated in C++17)	Unicode conversion facilities

Input/output library

<iosfwd>	Forward declarations of all classes in the input/output library
<ios>	std::ios_base class, std::basic_ios class template and several typedefs
<istream>	std::basic_istream class template and several typedefs
<ostream>	std::basic_ostream, std::basic_iostream class templates and several typedefs
<iostream>	Several standard stream objects
<fstream>	std::basic_fstream, std::basic_ifstream, std::basic_ofstream class templates and several typed
<sstream>	std::basic_stringstream, std::basic_istringstream, std::basic_ostringstream class templates an
<syncstream> (C++20)	std::basic_osyncstream, std::basic_syncbuf, and typedefs
<strstream> (deprecated in C++98)	std::strstream, std::istrstream, std::ostrstream
<iomanip>	Helper functions to control the format of input and output
<streambuf>	std::basic_streambuf class template
<cstdio>	C-style input-output functions

Filesystem library

<filesystem> (C++17)	std::path class and supporting functions

Regular Expressions library

<regex> (C++11)	Classes, algorithms and iterators to support regular expression processing

Atomic Operations library

<atomic> (C++11)	Atomic operations library

Thread support library

<thread> (C++11)	std::thread class and supporting functions
<stop_token> (C++20)	Stop tokens for std::jthread
<mutex> (C++11)	Mutual exclusion primitives
<shared_mutex> (C++14)	Shared mutual exclusion primitives
<future> (C++11)	Primitives for asynchronous computations
<condition_variable> (C++11)	Thread waiting conditions

<semaphore> (C++20)	Semaphores
<latch> (C++20)	Latches
<barrier> (C++20)	Barriers

C compatibility headers

For some of the C standard library headers of the form `xxx.h`, the C++ standard library both includes an identically-named header and another header of the form `cxxx` (all meaningful `cxxx` headers are listed above).

With the exception of `complex.h` , each `xxx.h` header included in the C++ standard library places in the global namespace each name that the corresponding `cxxx` header would have placed in the `std` namespace. These headers are allowed to also declare the same names in the `std` namespace, and the corresponding `cxxx` headers are allowed to also declare the same names in the global namespace: including <cstdlib> definitely provides `std::malloc` and may also provide `::malloc`. Including <stdlib.h> definitely provides `::malloc` and may also provide `std::malloc`. This applies even to functions and function overloads that are not part of C standard library.

<assert.h> (deprecated)	Behaves same as `<cassert>`
<ctype.h> (deprecated)	Behaves as if each name from `<cctype>` is placed in global namespace
<errno.h> (deprecated)	Behaves same as `<cerrno>`
<fenv.h> (C++11)(deprecated)	Behaves as if each name from `<cfenv>` is placed in global namespace
<float.h> (deprecated)	Behaves same as `<cfloat>`
<inttypes.h> (C++11)(deprecated)	Behaves as if each name from `<cinttypes>` is placed in global namespace
<limits.h> (deprecated)	Behaves same as `<climits>`
<locale.h> (deprecated)	Behaves as if each name from `<clocale>` is placed in global namespace
<math.h> (deprecated)	Behaves as if each name from `<cmath>` is placed in global namespace, except for names of mathematical special functions
<setjmp.h> (deprecated)	Behaves as if each name from `<csetjmp>` is placed in global namespace
<signal.h> (deprecated)	Behaves as if each name from `<csignal>` is placed in global namespace
<stdarg.h> (deprecated)	Behaves as if each name from `<cstdarg>` is placed in global namespace
<stddef.h> (deprecated)	Behaves as if each name from `<cstddef>` is placed in global namespace, except for names of `std::byte` and related functions
<stdint.h> (C++11)(deprecated)	Behaves as if each name from `<cstdint>` is placed in global namespace

<stdio.h> (deprecated)	Behaves as if each name from <cstdio> is placed in global namespace
<stdlib.h> (deprecated)	Behaves as if each name from <cstdlib> is placed in global namespace
<string.h> (deprecated)	Behaves as if each name from <cstring> is placed in global namespace
<time.h> (deprecated)	Behaves as if each name from <ctime> is placed in global namespace
<uchar.h> (C++11)(deprecated)	Behaves as if each name from <cuchar> is placed in global namespace
<wchar.h> (deprecated)	Behaves as if each name from <cwchar> is placed in global namespace
<wctype.h> (deprecated)	Behaves as if each name from <cwctype> is placed in global namespace

Empty C headers

The headers <complex.h>, <ccomplex>, <tgmath.h>, and <ctgmath> do not contain any content from the C standard library and instead merely include other headers from the C++ standard library. The use of all these headers is deprecated in C++.

<ccomplex> (C++11)(deprecated in C++17)(removed in C++20)	Simply includes the header <complex>
<complex.h> (C++11)(deprecated)	Simply includes the header <complex>
<ctgmath> (C++11)(deprecated in C++17)(removed in C++20)	Simply includes the headers <complex> and <cmath>: the overl header tgmath.h are already provided by those headers
<tgmath.h> (C++11)(deprecated)	Simply includes the headers <complex> and <cmath>

Meaningless C headers

The headers <ciso646>, <cstdalign>, and <cstdbool> are meaningless in C++ because the macros they provide in C are language keywords in C++.

<ciso646> (removed in C++20)	Empty header. The macros that appear in iso646.h in C are keyv
<iso646.h> (deprecated)	Has no effect
<cstdalign> (C++11)(deprecated in C++17)(removed in C++20)	Defines one compatibility macro constant
<stdalign.h> (C++11)(deprecated)	Defines one compatibility macro constant
<cstdbool> (C++11)(deprecated in C++17)(removed in C++20)	Defines one compatibility macro constant
<stdbool.h> (C++11)(deprecated)	Defines one compatibility macro constant