

Utilities library

- `<cstdlib>` - General purpose utilities: program control, dynamic memory allocation, random numbers, sort and search
- `<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>` - (since C++11) `std::type_index`
- `<type_traits>` - (since C++11) Compile-time type information
- `<bitset>` - `std::bitset` class template
- `<functional>` - Function objects, designed for use with the standard algorithms
- `<utility>` - Various utility components
- `<ctime>` - C-style time/date utilities
- `<chrono>` - (since C++11) C++ time utilities
- `<cstdint>` - typedefs for types such as `size_t`, `NULL` and others
- `<initializer_list>` - (since C++11) `std::initializer_list` class template
- `<tuple>` - (since C++11) `std::tuple` class template
- `<any>` - (since C++17) `std::any` class template
- `<optional>` - (since C++17) `std::optional` class template
- `<variant>` - (since C++17) `std::variant` class template

Dynamic memory management

- `<new>` - Low-level memory management utilities
- `<memory>` - Higher level memory management utilities
- `<scoped_allocator>` - (since C++11) Nested allocator class
- `<memory_resource>` - (since C++17) Polymorphic allocators and memory resources

Numeric limits

- `<climits>` - limits of integral types
- `<cfloat>` - limits of float types
- `<cstdint>` - (since C++11) fixed-size types and limits of other types
- `<cstdint>` - (since C++11) formatting macros, `intmax_t` and `uintmax_t` math and conversions

- `<limits>` - standardized 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>` - (since 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 type contained in character data
- `<cwctype>` - functions for determining the type of wide character data
- `<cstring>` - various narrow character string handling functions
- `<cwchar>` - various wide and multibyte string handling functions
- `<cuchar>` - (since C++11) C-style Unicode character conversion functions
- `<string>` - `std::basic_string` class template
- `<string_view>` - (since C++17) `std::basic_string_view` class template

Containers library

- `<array>` - (since C++11) `std::array` container
- `<vector>` - `std::vector` container
- `<deque>` - `std::deque` container
- `<list>` - `std::list` container
- `<forward_list>` - (since C++11) `std::forward_list` container
- `<set>` - `std::set` and `std::multiset` associative containers
- `<map>` - `std::map` and `std::multimap` associative containers
- `<unordered_set>` - (since C++11) `std::unordered_set` and `std::unordered_multiset` unordered associative containers
- `<unordered_map>` - (since 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

Algorithms library

- `<algorithm>` - Algorithms that operate on containers
- `<execution>` - (C++17) Predefined execution policies for parallel versions of the algorithms

Iterators library

- `<iterator>` - Container iterators

Numerics library

- `<cmath>` - Common mathematics functions
- `<complex>` - Complex number type
- `<valarray>` - Class for representing and manipulating arrays of values
- `<random>` - (since C++11) Random number generators and distributions
- `<numeric>` - Numeric operations on values in containers
- `<ratio>` - (since C++11) Compile-time rational arithmetic
- `<cfenv>` - (since C++11) Floating-point environment access functions

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 typedefs
- `<sstream>` - `std::basic_stringstream`, `std::basic_istringstream`, `std::basic_ostringstream` class templates and several typedefs
- `<strstream>` - `std::strstream`, `std::istrstream`, `std::ostrstream`(deprecated)
- `<iomanip>` - Helper functions to control the format or input and output
- `<streambuf>` - `std::basic_streambuf` class template
- `<cstdio>` - C-style input-output functions

Localization library

- `<locale>` - Localization utilities
- `<clocale>` - C localization utilities
- `<codecvt>` - (since C++11) Unicode conversion facilities

Regular Expressions library

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

Atomic Operations library

- `<atomic>` - (since C++11) Atomic operations library

Thread support library

- `<thread>` - (since C++11) `std::thread` class and supporting functions
- `<mutex>` - (since C++11) mutual exclusion primitives
- `<shared_mutex>` - (since C++14) shared mutual exclusion primitives
- `<future>` - (since C++11) primitives for asynchronous computations
- `<condition_variable>` - (since C++11) thread waiting conditions

Filesystem library

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

Experimental libraries

- `<experimental/algorithm>` - (library fundamentals TS) Standard libraries extensions and Extensions for Parallelism
- `<experimental/any>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/chrono>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/deque>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/execution_policy>` - (parallelism TS) Extensions for Parallelism
- `<experimental/exception_list>` - (parallelism TS) Extensions for Parallelism
- `<experimental/filesystem>` - (filesystem TS) Filesystem library

- `<experimental/forward_list>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/future>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/list>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/functional>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/map>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/memory>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/memory_resource>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/numeric>` - (parallelism TS) Extensions for Parallelism
- `<experimental/optional>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/ratio>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/regex>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/set>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/string>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/string_view>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/system_error>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/tuple>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/type_traits>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/unordered_map>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/unordered_set>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/utility>` - (library fundamentals TS) Standard libraries extensions
- `<experimental/vector>` - (library fundamentals TS) Standard libraries extensions

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 as if each name from `<cassert>` - is placed in global namespace

- `<ctype.h>` - (deprecated) behaves as if each name from `<cctype>` - is placed in global namespace
- `<errno.h>` - (deprecated) behaves as if each name from `<cerrno>` - is placed in global namespace
- `<fenv.h>` - (deprecated) behaves as if each name from `<cfenv>` - is placed in global namespace
- `<float.h>` - (deprecated) behaves as if each name from `<cfloat>` - is placed in global namespace
- `<inttypes.h>` - (deprecated) behaves as if each name from `<stdintypes>` - is placed in global namespace
- `<limits.h>` - (deprecated) behaves as if each name from `<climits>` - is placed in global namespace
- `<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
- `<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
- `<stdint.h>` - (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>` - (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

Unsupported C headers

The C headers `<stdatomic.h>` , `<stdnoreturn.h>` , and `<threads.h>` - are not included in C++ and have no cxxx equivalents.

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>` - (since C++11)(deprecated in C++17) simply includes the header `<complex>` - `<complex.h>` - (deprecated) simply includes the header `<complex>` - `<ctgmath>` - (since C++11)(deprecated in C++17) simply includes the headers `<complex>` - and `<cmath>` : the overloads equivalent to the contents of the C header `tgmath.h` are already provided by those headers
- `<tgmath.h>` - (deprecated) behaves as if each name from `<ctgmath>` - is placed in global namespace

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>` - empty header. The macros that appear in `iso646.h` in C are keywords in C++
- `<iso646.h>` - (deprecated) behaves as if each name from `<ciso646>` - is placed in global namespace
- `<cstdalign>` - (since C++11)(deprecated in C++17) defines one compatibility macro constant
- `<stdalign.h>` - (deprecated) behaves as if each name from `<cstdalign>` - is placed in global namespace
- `<cstdbool>` - (since C++11)(deprecated in C++17) defines one compatibility macro constant
- `<stdbool.h>` - (deprecated) behaves as if each name from `<cstdbool>` - is placed in global namespace