C++26 should refer to C23 not C17

Doc. no.: P3348R0
Author: Jonathan Wakely
Audience: LEWG, SG6
Date: 2024-08-02

Introduction

There is a new version of the ISO C standard, so we should update our normative reference, and update the header synopses in the library clauses to match the content of the C standard library.

A similar change to rebase C++17 on C11 was previously done via P0063R3: C++17 should refer to C11 instead of C99 (see R0 for more detailed rationale) and then to trivially rebase on C17 via C++20 ballot comment US-019 (which just updated the normative reference, as there were no changes affecting the contents of the library in C17).

Changes in C23 since C17

The new standard is not published yet, but is likely to be ISO/IEC 9899:2024. All dated references to ISO/IEC 9899:2018 should change to reference the new document instead.

C23 adds versioning macros to its headers, for example __STDC_VERSION_STDDEF_H__ is defined in <stddef.h>. We should decide whether we want to require those macros to be defined in C++, or require them to *not* be defined, or leave it implementation-defined or unspecified (1).

C23 adds a typedef `nullptr_t` and a macro `unreachable` to `<stddef.h>`, which are already present in C++ (in `<cstddef>` and `<utility>` respectively).

The type `once_flag` and the function `call_once` are added to `<stdlib.h>`. We do not want those, as we have our own `std::once_flag` and `std::call_once` in `<mutex>`.

In C23 the `alignas` and `alignof` macros are now keywords. As a result, the header `<stdalign.h>` is empty in C23. In C++ it only defines two deprecated macros, which no longer exist at all in C23 (note related issues LWG 3827 and LWG 4036). I would prefer to deprecate the entire header, and eventually remove it (and add it to the zombie headers table) but that would be going beyond what C23 does. For the purposes of the current proposal, the header is unchanged and its content remains deprecated.

In C23 the `bool`, `true` and `false` macros are now keywords. As a result, the header `<stdbool.h>` is empty in C23, except for the obsolescent (i.e. deprecated) macro, `__bool_true_false_are_defined`. That macro is already deprecated in C++23. As with stdalign.h, I would prefer to deprecate then remove the entire header, but for the purposes of the current proposal, the headers are unchanged and their content remains deprecated.

In C23, the `asctime` and `ctime` functions are deprecated. We should do the same.

In C23 the `DECIMAL_DIG` macro is deprecated. We should do the same. The use of `FLT_HAS_SUBNORM`, `DBL_HAS_SUBNORM` and `DBL_HAS_SUBNORM` macros is marked as obsolescent. SG6 should decide if we want to do the same (2).

In C23, the `INFINITY` and `NAN` macros are defined in `<float.h>`. In C17 they were defined in `<math.h>`, and are still there in C23, but defining them there is deprecated. We should decide what to do here (3), but we probably want to do the same.

FLT_SNAN, DBL_SNAN, LDBL_SNAN added to <float.h>. We have equivalents in std::numeric_limits already, but it seems harmless to add them.

Additions to <fenv.h>: The `femode_t` type and `FE_DFL_MODE` macro. New rounding direction, `FE_TONEARESTFROMZERO`. `FENV_ROUND` pragma. The `fesetexcept` and `fetestexceptflag` functions. The `fegetmode` and `fesetmode` functions. Do we want them? (4)

Additions to <math.h>: Decimal floating-point types. New functions `fromfp`, `ufromfp`, `fromfpx`, `fromupx`, and the math rounding directions, `FP_INT_UPWARD` etc. New macros `FP_FAST_FMA`, `FP_FAST_FMAF`, and `FP_FAST_FMAL`. 18 new macros, `FP_FAST_FADD` etc. Do we want them? (5)

Also in <math.h>: New `iscanonical` macro and core language concept of canonical representations and non-canonical representations in floating-point types. These are primarily needed for decimal floating-point types, which we don't have. No need to add these to <cmath> at this time.

New header <stdbit.h> with overlapping functionality to the C++ header <bit>. LEWG had consensus for adding <stdbit.h> with the content re-specified using C++ features, but with the same names as C uses. That is not part of this rebasing proposal, so will need to be proposed and considered separately. There was no consensus to add a <cstdbit> to C++. I am very strongly against adding such a header, because code that needs to be compatible with C should use <stdbit.h> and code that doesn't need to be compatible with C should use <bit>. There is no reason for <cstdbit> to exist.

New header <stdckdint.h> with functions for checking for overflow in addition, subtraction and multiplication. C++ has no equivalent currently, but we probably don't want type-generic macros like C has. The APIs would be better as templates with clear *Mandates*: requirements for suitable integer types. LEWG had consensus for adding <stdckdint.h> with the content re-specified using C++ features. That is not part of this rebasing proposal, so will need to be proposed and considered separately.

LEWG discussed this at a 2024-07-30 telecon and took some polls, with consensus to add the new functions to <string.h>, <time.h> and <stdlib.h>, and to include the new %OB and %Ob formats for strftime. The new functions are shown [TODO: not all there yet, but they will be] in the proposed wording below. There's no change shown for strftime, because it happens automatically by making C23 our reference.

Wording

This wording is not yet complete, but there are questions to be answered by LEWG and SG6 before it reaches LWG anyway.

The changes shown are relative to N4981, Working Draft (2024-04-16).

All dated references to ISO/IEC 9899:2018 should change. This is done by updating the `\IsoC` LaTeX macro in one place, but all affected text is shown below so the changes can be reviewed for correctness.

[*Drafting note*: The new C standard has not been formally published by ISO so the proposed wording assumes it will get published in 2024]

TODO: need blanket wording saying no functions or macros for decimal floating-point types are declared in C++ headers.

Update 1.2 [intro.refs] p1.3:

> (1.3) — ISO/IEC 9899:~~2018~~2024, Programming languages — C

Update 3.8 [defns.c.lib]

> **C standard library**
> library described in ISO/IEC 9899:~~2018~~2024, Clause 7
> [*Note 1 to entry*: With the qualifications noted in Clause 17 through Clause 33 and in C.8, the C standard library is a subset of the C++ standard library. — *end note*]

Update 16.2 [c.library] p3:

> A call to a C standard library function is a non-constant library call (3.34) if it raises a floating-point exception other than `FE_INEXACT`. The semantics of a call to a C standard library function evaluated as a core constant expression are those specified in ISO/IEC 9899:~~2018~~2024, Annex F[136] to the extent applicable to the floating-point types (6.8.2) that are parameter types of the called function.
>
> 136) See also ISO/IEC 9899:~~2018~~2024, 7.6.
> [*Drafting note*: This subclause is “Floating-point environment<fenv.h>” and is still 7.6 in C23.]

Update 16.4.2.3 [headers] p10:

> ISO/IEC 9899:~~2018~~2024, Annex K describes a large number of functions, with associated types and macros, which “promote safer, more secure programming” than many of the traditional C library functions. The names of the functions have a suffix of `_s`; most of them provide the same service as the C library function with the unsuffixed name, but generally take an additional argument whose value is the size of the result array. If any C++ header is included, it is implementation-defined whether any of these names is declared in the global namespace. (None of them is declared in namespace `std`.)

Also in 16.4.2.3 [headers], update the caption of Table 26 [tab:c.annex.k.names]:

> **Table 26 — Names from ISO/IEC 9899:~~2018~~2024, Annex K [tab:c.annex.k.names]**

Update the footnote in 16.4.3.3 [using.linkage]

> Whether a name from the C standard library declared with external linkage has extern "C" or extern "C++" linkage is implementation-defined. It is recommended that an implementation use extern "C++" linkage for this purpose.[155]
>
> 155) The only reliable way to declare an object or function signature from the C standard library is by including the header that declares it, notwithstanding the latitude granted in ISO/IEC 9899:~~2018~~2024, 7.1.4.

[*Drafting note*: This subclause is “Use of library functions” and is still 7.1.4 in C23.]

Update 17.2.1 [cstddef.syn] p1:

The contents and meaning of the header `<cstddef>` are the same as the C standard library header `<stddef.h>`, except that it does not declare the type `wchar_t`, that it does not declare the macro `unreachable`, that it also declares the type `byte` and its associated operations (17.2.5), and as noted in 17.2.3 and 17.2.4.

[*Drafting note*: 17.2.3 describes how `nullptr_t` is defined in C++, overriding how C defines its version.]

See also: ISO/IEC 9899:~~2018~~2024, 7.~~19~~22

[*Drafting note*: This subclause is “Common definitions <stddef.h>”, which is 7.22 in C23.]

Update 17.2.2 [cstdlib.syn] p2:

TODO: add strfromd, strfromf, strfroml, free_sized, free_aligned_sized, memalignment, memccpy, gmtime_r, localtime_r, timespec_getres

The contents and meaning of the header `<cstdlib>` are the same as the C standard library header `<stdlib.h>`, except that it does not declare the type `wchar_t`, that it does not declare the type `once_flag` or the function `call_once`, and except as noted in 17.2.3, 17.2.4, 17.5, 20.2.12, 23.5.6, 27.13, 28.5.10, and 28.7.2.

[*Note 1* : Several functions have additional overloads in this document, but they have the same behavior as in the C standard library (16.2). — *end note*]

See also: ISO/IEC 9899:~~2018~~2024, 7.~~22~~24

[*Drafting note*: This subclause is “General utilities <stdlib.h>” , which is 7.24 in C23.]

Update 17.2.3 [support.types.nullptr] p2:

-1- The type `nullptr_t` is a synonym for the type of a `nullptr` expression, and it has the characteristics described in 6.8.2 and 7.3.12.

[*Note 1*: Although `nullptr`’s address cannot be taken, the address of another `nullptr_t` object that is an lvalue can be taken. — *end note*]

-2- The macro `NULL` is an implementation-defined null pointer constant.164

See also: ISO/IEC 9899:~~2018~~2024, 7.~~19~~22

[*Drafting note*: This subclause is “Common definitions <stddef.h>”, which is 7.22 in C23.]

Update 17.2.4 [support.types.layout] p5:

The type `max_align_t` is a trivial standard-layout type whose alignment requirement is at least as great as that of every scalar type, and whose alignment requirement is supported in every context (6.7.6).

See also: ISO/IEC 9899:~~2018~~2024, 7.~~19~~22

[*Drafting note*: This subclause is “Common definitions <stddef.h>”, which is 7.22 in C23.]

Update 17.3.6 [climits.syn] p1:

```
// all freestanding
```

```
#define BOOL_WIDTH see below
#define CHAR_BIT see below
#define CHAR_WIDTH see below
#define SCHAR_WIDTH see below
#define UCHAR_WIDTH see below
#define USHRT_WIDTH see below
#define SHRT_WIDTH see below
#define UINT_WIDTH see below
#define INT_WIDTH see below
#define ULONG_WIDTH see below
#define LONG_WIDTH see below
#define ULLONG_WIDTH see below
#define LLONG_WIDTH see below
#define SCHAR_MIN see below
#define SCHAR_MAX see below
#define UCHAR_MAX see below
#define CHAR_MIN see below
#define CHAR_MAX see below
#define MB_LEN_MAX see below
#define SHRT_MIN see below
#define SHRT_MAX see below
#define USHRT_MAX see below
#define INT_MIN see below
#define INT_MAX see below
#define UINT_MAX see below
#define LONG_MIN see below
#define LONG_MAX see below
#define ULONG_MAX see below
#define LLONG_MIN see below
#define LLONG_MAX see below
#define ULLONG_MAX see below
```

The header `<climits>` defines all macros the same as the C standard library header `<limits.h>`, except that it does not declare the macro `BITINT_MAXWIDTH`.

[*Note 1*: Except for the `WIDTH` macros, `CHAR_BIT` and `MB_LEN_MAX`, a macro referring to an integer type `T` defines a constant whose type is the promoted type of `T` (7.3.7). — *end note*]

See also: ISO/IEC 9899:~~2018~~2024, 5.2.4.2.1

[*Drafting note*: This subclause is “Sizes of integer types <limits.h>” and has changed name to “Characteristics of integer types <limits.h>” but is still 5.2.4.2.1 in C23.]

Update 17.3.7 [cfloat.syn] p1:

```
// all freestanding
#define FLT_ROUNDS see below
#define FLT_EVAL_METHOD see below
#define FLT_HAS_SUBNORM see below
#define DBL_HAS_SUBNORM see below
#define LDBL_HAS_SUBNORM see below
#define INFINITY see below
#define NAN see below
#define FLT_SNAN see below
#define DBL_SNAN see below
```

```
#define LDBL_SNAN see below
#define FLT_RADIX see below
…
```

The header `<cfloat>` defines all macros the same as the C standard library header `<float.h>`.
[*Drafting note*: See Annex D entry for DECIMAL_DIG being deprecated, which is “the same as” <float.h>.]
See also: ISO/IEC 9899:~~2018~~2024, 5.2.4.2.2
[*Drafting note*: This subclause is “Characteristics of floating types <float.h>” and is still 5.2.4.2.2 in C23.]

Update 17.4.1 [cstdint.syn] p1:

TODO: INT*N*_WIDTH, INT_LEAST*N*_WIDTH, SIZE_WIDTH etc.

The header defines all types and macros the same as the C standard library header `<stdint.h>`.
The types denoted by `intmax_t` and `uintmax_t` are not required to be able to represent all values of extended integer types wider than `long long` and `unsigned long long`, respectively.
[*Drafting note*: This text was added to 31.13.2 [cinttypes.syn] by LWG 3028, but <cinttypes> doesn’t define these types, so this is the correct place to say it.]
See also: ISO/IEC 9899:~~2018~~2024, 7.~~20~~22
[*Drafting note*: This subclause is “Integer types <stdint.h>” and is 7.22 in C23.]

Update 17.5 [support.start.term] p14:

Remarks: The function `quick_exit` is signal-safe (17.13.5) when the functions registered with `at_quick_exit` are.

See also: ISO/IEC 9899:~~2018~~2024, 7.~~22~~24.4
[*Drafting note*: This subclause is “Communication with the environment” and is 7.24.4 in C23.]

Update 17.13.2 [cstdarg.syn] p1:

```
// all freestanding
namespace std {
  using va_list = see below;
}
#define va_arg(V, P) see below
#define va_copy(VDST, VSRC) see below
#define va_end(V) see below
#define va_start(V, ~~P~~ ...) see below
```

The contents of the header `<cstdarg>` are the same as the C standard library header `<stdarg.h>`, with the following changes:
(1.1) — In lieu of the default argument promotions specified in ISO C 6.5.2.2, the definition in 7.6.1.3 applies.
[*Drafting note*: This subclause is “Function calls” and is still 6.5.2.2 in C23.]
(1.2) — The restrictions that ISO C places on the second parameter to the `va_start` macro in header `<stdarg.h>` are different in this document. The parameter `parmN` is the rightmost parameter in the variable
parameter list of the function definition (the one just before the `...`).[196] If the parameter `parmN` is a pack expansion (13.7.4) or an entity resulting from a lambda capture (7.5.5), the program is ill-formed, no diagnostic required. If the parameter `parmN` is of a reference type, or of a type that is not compatible with the type that results when passing an argument for which there is no parameter, the behavior is

undefined.
See also: ISO/IEC 9899:~~2018~~2024, 7.16~~.1.1~~
[*Drafting note*: This subclause is “The `va_arg` macro” and is still 7.16.1.1 C23, but it seems that 17.6 “Variable arguments <stdarg.h>” would be more appropriate here.]

Update 17.13.3 [csetjmp.syn] p2:

The function signature `longjmp(jmp_buf jbuf, int val)` has more restricted behavior in this document. A `setjmp`/`longjmp` call pair has undefined behavior if replacing the `setjmp` and `longjmp` by `catch` and `throw` would invoke any non-trivial destructors for any objects with automatic storage duration. A call to `setjmp` or `longjmp` has undefined behavior if invoked in a suspension context of a coroutine (7.6.2.4).
See also: ISO/IEC 9899:~~2018~~2024, 7.13
[*Drafting note*: This subclause is “Non-local jumps <setjmp.h>” and is still 7.13 in C23.]

Update 17.13.4 [csignal.syn] p4:

The function `signal` is signal-safe if it is invoked with the first argument equal to the signal number corresponding to the signal that caused the invocation of the handler.
See also: ISO/IEC 9899:~~2018~~2024, 7.14
[*Drafting note*: This subclause is “Signal handling <signal.h>” and is still 7.14 in C23.]

The cross references in 17.14.1 [support.c.headers.general] p1 will change:

For compatibility with the C standard library, the C++ standard library provides the C headers shown in Table 44. The intended use of these headers is for interoperability only. It is possible that C ++ source files need to include one of these headers in order to be valid ISO C. Source files that are not intended to also be valid ISO C should not use any of the C headers.
[Note 1 : The C headers either have no effect, such as `<stdbool.h>` (~~17.14.5~~ D.?) and `<stdalign.h>` (~~17.14.4~~ D.?), or otherwise the corresponding header of the form `<`*`cname`*`>` provides the same facilities and assuredly defines them in namespace `std`. — end note]

Update 17.14.4 [stdalign.h.syn], C23 no longer defines this macro, so there is no difference:

17.14.4 Header <stdalign.h> synopsis [stdalign.h.syn]
The contents of the C++ header `<stdalign.h>` are the same as the C standard library header `<stdalign.h>`~~, with the following changes: The header `<stdalign.h>` does not define a macro named `alignas`~~.
See also: ISO/IEC 9899:~~2018~~2024, 7.15
[*Drafting note*: This subclause is “Alignment <stdalign.h>” and is still 7.15 in C23.]

Update 17.14.5 [stdbool.h.syn], C23 no longer defines these macros, so there is no difference:

17.14.5 Header <stdbool.h> synopsis [stdbool.h.syn]
The contents of the C++ header `<stdbool.h>` are the same as the C standard library header `<stdbool.h>`~~, with the following changes: The header `<stdbool.h>` does not define macros named `bool`, `true`, or `false`~~.
See also: ISO/IEC 9899:~~2018~~2024, 7.~~18~~19
[*Drafting note*: This subclause is “Boolean type and values <stdbool.h>” and is 7.19 in C23.]

Update 23.5.3 [cstring.syn]:

```
namespace std {
  using size_t = see 17.2.4;
  void* memcpy(void* s1, const void* s2, size_t n);   // freestanding
  void* memmove(void* s1, const void* s2, size_t n); // freestanding
  char* strcpy(char* s1, const char* s2);            // freestanding
  char* strncpy(char* s1, const char* s2, size_t n); // freestanding
  char* strdup(const char* s);
  char* strndup(const char* s, size_t size);
  char* strcat(char* s1, const char* s2);            // freestanding
  char* strncat(char* s1, const char* s2, size_t n); // freestanding
  ...
  void* memset(void* s, int c, size_t n);            // freestanding
  void* memset_explicit(void* s, int c, size_t n);   // freestanding
  char* strerror(int errnum);
  ...
```

Update 28.3.1 [cfenv.syn]:

TODO:

Update 28.7.1 [cmath.syn] p1:

The contents and meaning of the header `<cmath>` are the same as the C standard library header `<math.h>`, ~~with~~ except for:

- the addition of a three-dimensional hypotenuse function (28.7.3), a linear interpolation function (28.7.4), and the mathematical special functions described in 28.7.6,
- the removal of all types, macros, and functions that depend on `__STDC_IEC_60559_DFP__`, and
- the removal of the `iscanonical` macro, the `canonicalize`, `canonicalizef`, and `canonicalizel` functions.

[*Note 1*: Several functions have additional overloads in this document, but they have the same behavior as in the C standard library (16.2). — *end note*]

TODO: fmaximum, fminimum, fmaximum_mag, fminimum_mag, fmaximum_num, fminimum_num, fmaximum_mag_num, fminimum_mag_num, nextdown, nextup

Update 29.15 [ctime.syn]:

```
#define NULL see 17.2.3
#define CLOCKS_PER_SEC see below
#define TIME_UTC see below
#define TIME_MONOTONIC see below
#define TIME_ACTIVE see below
#define TIME_THREAD_ACTIVE see below
```

[*Drafting note*: these new time bases are optional in C, do we need to state that explicitly here too?]

```
namespace std {
  using size_t = see 17.2.4;
  using clock_t = see below;
  using time_t = see below;

  struct timespec;
  struct tm;
  clock_t clock();
  double difftime(time_t time1, time_t time0);
  time_t mktime(tm* timeptr);
  time_t timegm(tm* timeptr);
  time_t time(time_t* timer);
  int timespec_get(timespec* ts, int base);
  int timespec_getres(timespec* ts, int base);
  [[deprecated]] char* asctime(const tm* timeptr);
  [[deprecated]] char* ctime(const time_t* timer);
  tm* gmtime(const time_t* timer);
  tm* localtime(const time_t* timer);
  size_t strftime(char* s, size_t maxsize, const char* format, const tm*
timeptr);
}
```

Update 30.4.6.4.2 [locale.time.put.members] to remove an outdated footnote. This footnote was true in C++98, but C99 added some modifiers, so the footnote has been wrong for a long time.

```
iter_type put(iter_type s, ios_base& str, char_type fill, const tm* t,
              const charT* pattern, const charT* pat_end) const;
iter_type put(iter_type s, ios_base& str, char_type fill, const tm* t,
              char format, char modifier = 0) const;
```

Effects: The first form steps through the sequence from pattern to `pat_end`, identifying characters that are part of a format sequence. Each character that is not part of a format sequence is written to `s` immediately, and each format sequence, as it is identified, results in a call to `do_put`; thus, format elements and other characters are interleaved in the output in the order in which they appear in the pattern. Format sequences are identified by converting each character c to a char value as if by `ct.narrow(c, 0)`, where `ct` is a reference to `ctype<charT>` obtained from `str.getloc()`. The first character of each sequence is equal to `'%'`, followed by an optional modifier character `mod`~~[251]~~ and a format specifier character spec as defined for the function `strftime`. If no modifier character is present, `mod` is zero. For each valid format sequence identified, calls `do_put(s, str, fill, t, spec, mod)`. The second form calls `do_put(s, str, fill, t, format, modifier)`.

~~251) Although the C programming language defines no modifiers, most vendors do.~~

Update 31.13.1 [cstdio.syn]:

```
namespace std {
  using size_t = see 17.2.4;
  using FILE = see below;
  using fpos_t = see below;
}
#define NULL see 17.2.3
#define _IOFBF see below
#define _IOLBF see below
#define _IONBF see below
#define BUFSIZ see below
#define EOF see below
```

```
#define FOPEN_MAX see below
#define FILENAME_MAX see below
#define _PRINTF_NAN_LEN_MAX see below
#define L_tmpnam see below
#define SEEK_CUR see below
#define SEEK_END see below
#define SEEK_SET see below
#define TMP_MAX see below
#define stderr see below
#define stdin see below
#define stdout see below
```

Update 31.13.2 [cinttypes.syn]:

```
#define PRIdN see below
#define PRIiN see below
#define PRIoN see below
#define PRIuN see below
#define PRIxN see below
#define PRIXN see below
#define PRIbN see below
#define PRIBN see below
#define SCNdN see below
#define SCNiN see below
#define SCNoN see below
#define SCNuN see below
#define SCNxN see below
#define SCNbN see below
#define PRIdLEASTN see below
#define PRIiLEASTN see below
#define PRIoLEASTN see below
#define PRIuLEASTN see below
#define PRIxLEASTN see below
#define PRIXLEASTN see below
#define PRIbLEASTN see below
#define PRIBLEASTN see below
#define SCNdLEASTN see below
#define SCNiLEASTN see below
#define SCNoLEASTN see below
#define SCNuLEASTN see below
#define SCNxLEASTN see below
#define SCNbLEASTN see below
#define PRIdFASTN see below
#define PRIiFASTN see below
#define PRIoFASTN see below
#define PRIuFASTN see below
#define PRIxFASTN see below
#define PRIXFASTN see below
#define PRIbFASTN see below
#define PRIBFASTN see below
#define SCNdFASTN see below
#define SCNiFASTN see below
#define SCNoFASTN see below
```

```
#define SCNuFASTN see below
#define SCNxFASTN see below
#define SCNbFASTN see below
#define PRIdMAX see below
#define PRIiMAX see below
#define PRIoMAX see below
#define PRIuMAX see below
#define PRIxMAX see below
#define PRIXMAX see below
#define PRIbMAX see below
#define PRIBMAX see below
#define SCNdMAX see below
#define SCNiMAX see below
#define SCNoMAX see below
#define SCNuMAX see below
#define SCNxMAX see below
#define SCNbMAX see below
#define PRIdPTR see below
#define PRIiPTR see below
#define PRIoPTR see below
#define PRIuPTR see below
#define PRIxPTR see below
#define PRIXPTR see below
#define PRIbPTR see below
#define PRIBPTR see below
#define SCNdPTR see below
#define SCNiPTR see below
#define SCNoPTR see below
#define SCNuPTR see below
#define SCNxPTR see below
#define SCNbPTR see below
```

1 The contents and meaning of the header `<cinttypes>` are the same as the C standard library header `<inttypes.h>`, with the following changes:

(1.1) — The header `<cinttypes>` includes the header `<cstdint>` (17.4.1) instead of `<stdint.h>`, and

~~(1.2) — `intmax_t` and `uintmax_t` are not required to be able to represent all values of extended integer types wider than `long long` and `unsigned long long`, respectively, and~~

[*Drafting note*: This text is moved to 17.4.1 [cstdint.syn], see above.]

(1.3) — if and only if the type `intmax_t` designates an extended integer type (6.8.2), the following function signatures are added:

```
      constexpr intmax_t abs(intmax_t);
      constexpr imaxdiv_t div(intmax_t, intmax_t);
```

which shall have the same semantics as the function signatures `constexpr intmax_t imaxabs(intmax_t)` and `constexpr imaxdiv_t imaxdiv(intmax_t, intmax_t)`, respectively.

See also: ISO/IEC 9899:~~2018~~2024, 7.8

[*Drafting note*: This subclause is “Format conversion of integer types <inttypes.h>” and is still 7.8 in C23.]

2 Each of the `PRI` macros listed in this subclause is defined if and only if the implementation defines the corresponding *typedef-name* in 17.4.1. Each of the `SCN` macros listed in this subclause is defined if and only if the implementation defines the corresponding *typedef-name* in 17.4.1 and has a suitable `fscanf` length

modifier for the type. Each of the `PRIB` macros listed in this subclause is defined if and only if `fprintf` supports the `B` conversion specifier.

Add a new subclause after C.1.4 [diff.cpp23.library],:

C.1.? Clause 20: Memory management library

Affected subclause: [c.malloc]
Change: Calling `realloc` with zero size has undefined behavior.
Rationale: Consistency with ISO C.
Effect on original feature: Valid C++ 2023 code that calls `realloc` with a size of zero has undefined behavior in this version of C++.

Update C.8.2 [diffs.mods.to.headers] p2:

There are no C++ headers for the C standard library's headers <stdbit.h>, <stdckdint.h>, <stdnoreturn.h> and <threads.h>, nor are these headers from the C standard library headers themselves part of C++.

Remove C.8.3.3 [diff.header.assert.h]:

~~1 The token `static_assert` is a keyword in C++. It does not appear as a macro name defined in `<cassert>` (19.3.2).~~

Remove C.8.3.5 [diff.header.stdalign.h]:

~~1 The token `alignas` is a keyword in C++ (5.11), and is not introduced as a macro by `<stdalign.h>` (17.14.4).~~

Remove C.8.3.6 [diff.header.stdbool.h]:

~~1 The tokens `bool`, `true`, and `false` are keywords in C ++ (5.11), and are not introduced as macros by `<stdbool.h>` (17.14.5).~~

Update D.11 [depr.c.macros]:

D.11 Deprecated C macros [depr.c.macros]

1 The header <cfloat> has the following macros:

```
#define FLT_HAS_SUBNORM see below
#define DBL_HAS_SUBNORM see below
#define LDBL_HAS_SUBNORM see below
#define DECIMAL_DIG 10
```

The header defines these macros the same as the C standard library header `<float.h>`.
See also: ISO/IEC 9899:2024, 5.2.4.2.2, 7.33.5
[*Drafting note*: C23 5.2.4.2.2 <float.h> has a cross-reference to 7.33.8 for DECIMAL_DIG being obsolescent, but that's incorrect and should be 7.33.5 as shown here.]

2 In addition to being available via inclusion of the `<cfloat>` header, the macros `INFINITY` and `NAN` are available when `<cmath>` is included.

See also: ISO/IEC 9899:2024, 7.12

~~1 The header <stdalign.h> has the following macro:~~

```
~~#define __alignas_is_defined 1~~
~~#define __alignof_is_defined 1~~
```

[*Drafting note*: The stdalign macros are removed entirely from C23, without deprecation.]

2 The header <stdbool.h> has the following macro:

```
#define __bool_true_false_are_defined 1
```

See also: ISO/IEC 9899:2024, 7.19

[*Drafting note*: This macro is still present in C23, but marked obsolete.]