Remove conditional “WANT” macros from numbered clauses
proposal for C2x

Jens Gustedt
INRIA and ICube, Université de Strasbourg, France

The recent integration of TS 18661-1 has moved the use of “WANT” macros into the main body of the C standard, making the added interfaces optional. We think that this is not optimal, neither for user code nor for implementations, an propose to change that to a set of more straight forward feature test macros for the version of the included headers. Along with that also a long list of names have been imposed to the standard. We propose some mild modifications to reduce the pain of the transition and keep C open for future directions.

1. INTRODUCTION
When it was designed, TS 18661-1 (and follow ups) invented a mechanism that would allow implementations to provide that extension in the concerned headers without imposing a pollution of the user name space for code that was not TS 18661-1 aware. Whereas in that context the approach made complete sense, continuing with the same setting once integrated into ISO/IEC 9899 is not very constructive.
— It makes interfaces optional that shouldn’t be.
— It reduces exposure of the new interfaces to a very restricted set of applications.
— It adds unnecessary complexity to implementations.

On the other hand, adding new mandatory interfaces to standard headers also has its cost, namely the increasing risk of name conflicts with an existing code base. This risk is relatively high for TS 18661-1:
— TS 18661-1 adds about 150 (13%) new interfaces (functions and macros) to the C standard.
— Some of these interfaces use plain English words (canonicalize), short abbreviations (daddl) or introduce unusual naming schemes (fromfp), that have an even higher risk of name conflicts that the usual prefix-oriented additions.

The proposal of this paper is to remove the conditionality of these interfaces by

1. removing the dependency from the __STDC_Want_iEC_60559_BFP_EXT__ macro,
2. by adding version test macros such as __STDC_FENV_VERSION__ to the headers that undergo changes,
3. by revisiting some of the naming choices, and
4. by reserving some identifier prefixes for future use.

2. REMOVING DEPENDENCY FROM __STDC_WANT_IEC_60559_BFP_EXT__
The only construct in the standard that would be similar to __STDC_WANT_IEC_60559_BFP_EXT__ is __STDC_WANT_LIB_EXT1__ as it used by Annex K. Since the features of Annex K are optional (testable by __STDC_LIB_EXT1__) such a macro makes complete sense there, because we don’t want an implementation that has Annex K to pollute the name space of all its users.

For the integration of TS 18661-1 the situation is different. It has mainly (see below) integrated directly into the body of the standard, and there is no reason (or feature test macro) that indicates that the interfaces should be optional. In the contrary, most of them are useful additions that should make coding with floating point data more convenient and numerical algorithms more robust.
There are only a few new interfaces that are not integrated into the body of the standard but into Annex F, where a dependency from `__STDC_WANT_IEC_60559_BFP_EXT__` makes perfect sense, namely for the same reasons as mentioned above for Annex K. Therefore we simply propose

— to move the boilerplate for WANT macros from 7.1.2 (Standard headers) to Annex F.
— to remove the use of `__STDC_WANT_IEC_60559_BFP_EXT__` from all numbered clauses, but to keep it in Annex F.

Editorially these two steps are quite easy, and we show their application in the attached diffmarks.

3. ADDING VERSION TEST MACROS

The addition of about 150 new interfaces for a new C version can be quite a burden for large code bases that wish to migrate to C2x. Conflicts will not occur often, but they are likely to occur somewhere and should be easy to track and to manage. Therefore we should provide an easy-to-use tool that allows for user code to control the possible damage, but on the other hand will not impose much of a maintenance burden for implementations either.

Another difficulty that appears when the community moves to a new C standard is the fact that nowadays compilers and C libraries often come from different hands, and thus their synchronization concerning a new standard is not trivial. History has shown that this has been major hurdle for early acceptance of new C standards, and that dependency of one single “language” version macro `__STDC_VERSION__` is not enough to clarify the situation. Therefore we propose to use a set of new macros of the form `__STDC___XXX_VERSION__`. For example `<math.h>` sets a new macro `__STDC_MATH_VERSION__` to a value greater than 202000L, and users can then test this as follows.

```c
#include <math.h>
#if __STDC_MATH_VERSION__ > 202000L
#   error "this code likes to daddl, fix before going further"
#endif
```

There is already large experience with the use of such version macros for library headers in ISO/IEC 9945, POSIX. There, such macros are defined for major branches of the standard and applications have learned to deal with them to adapt their code to the actual environment.

4. REVISITING SOME OF THE NAMING CHOICES

Many of the new interfaces would better have been introduced with a name prefix, much as other headers did when they were added to the C standard. It seems that this opportunity has been missed, though I think that we still could take a turn and use names such as `fp_canonicalize` instead of `canonicalize`, `fp_add` instead of `fadd`, etc.

Where these additions are particularly bad is where they introduce a new naming scheme (without admitting it) that is even contraproductive to a future encapsulation of these interfaces in a type generic function. These are the functions

```
fromfpf  fromfpxl  strfrommd  ufromfpf  ufromfpxl
fromfpfl fromfpfl  strfrommf  ufromfpfl ufromfpfl
fromfpfx fromfpfx  strfromml  ufromfpfx ufromfpfx
```
Here the usage of the particle `from` has no precedent in the standard. It is not a good choice because in C conversions do usually not specify the source type of a conversion (it can be deduced from the context) but, if so, the target type. By the naming choice, these interfaces cannot be easily extended to type generic interfaces, since by their nature these should have the source type implicit and the target type of feature explicit.

Therefore we propose to rename these interfaces to names starting with the reserved prefix `to`, namely

```plaintext
  tointf  tointxl  tostrd  touintf  touintxl
  tointl  tointx   tostrf  touintl  touintx
  tointxf toint    tostrl  touintxf  touint
```

This clears up the type generic interfaces in `<tgmath.h>` (to `toint` and `touint`) and will permit to propose another type generic interface in the sequel, in particular a macro `tostr` for a type generic and safe conversion interface conversion from any base type to a string.

5. Reserve active prefixes for future use

The integration of TS 18661-1 has also shown that four prefixes are actively used for new macro interfaces (namely `DBL_`, `FLT_`, `LDBL_` and `FP_`) and should thus not be used by user code. Therefore we propose to reserve these for future use. In addition, we propose also to extend the future use clauses of some other prefixes to the header files were they are actually used.

Appendix: pages with diffmarks of the proposed changes against the March 2019 working draft.

The following page numbers are from the particular snapshot and may vary once the changes are integrated.
— 63 nesting levels of parenthesized declarators within a full declarator
— 63 nesting levels of parenthesized expressions within a full expression
— 63 significant initial characters in an internal identifier or a macro name (each universal character name or extended source character is considered a single character)
— 31 significant initial characters in an external identifier (each universal character name specifying a short identifier of 0000FFFF or less is considered 6 characters, each universal character name specifying a short identifier of 00010000 or more is considered 10 characters, and each extended source character is considered the same number of characters as the corresponding universal character name, if any)\(^1\)
— 4095 external identifiers in one translation unit
— 511 identifiers with block scope declared in one block
— 4095 macro identifiers simultaneously defined in one preprocessing translation unit
— 127 parameters in one function definition
— 127 arguments in one function call
— 127 parameters in one macro definition
— 127 arguments in one macro invocation
— 4095 characters in a logical source line
— 4095 characters in a string literal (after concatenation)
— 65535 bytes in an object (in a hosted environment only)
— 15 nesting levels for \#included files
— 1023 case labels for a switch statement (excluding those for any nested switch statements)
— 1023 members in a single structure or union
— 1023 enumeration constants in a single enumeration
— 63 levels of nested structure or union definitions in a single member declaration list

5.2.4.2 Numerical limits

An implementation is required to document all the limits specified in this subclause, which are specified in the headers `<limits.h>` and `<float.h>`. Additional limits are specified in `<stdint.h>`.

Forward references: integer types `<stdint.h>` (7.20).

5.2.4.2.1 Sizes of integer types `<limits.h>`

The following identifiers are defined only if \_\_STDC\_WANT\_IEC\_60559\_BFP\_EXT \_ is defined as a macro at the point in the source file where is first included:

```c
CHAR_WIDTH
SCHAR_WIDTH
 UCHAR_WIDTH
 SHRT_WIDTH
 USHRT_WIDTH
 INT_WIDTH
 UINT_WIDTH
 LONG_WIDTH
 ULONG_WIDTH
```

\(^1\)See “future language directions” (6.11.3).
The values given below shall be replaced by constant expressions suitable for use in \#if preprocessing directives. Moreover, except for \texttt{CHAR\_BIT} and \texttt{MB\_LEN\_MAX}, and the width-of-type macros, the following shall be replaced by expressions that have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Their implementation-defined values shall be equal or greater in magnitude (absolute value) to those shown, with the same sign.

- number of bits for smallest object that is not a bit-field (byte)

\begin{verbatim}
  CHAR\_BIT  8
\end{verbatim}

- minimum value for an object of type \texttt{signed char}

\begin{verbatim}
  SCHAR\_MIN -127 // -(2\textsuperscript{7} - 1)
\end{verbatim}

- maximum value for an object of type \texttt{signed char}

\begin{verbatim}
  SCHAR\_MAX +127 // 2\textsuperscript{7} - 1
\end{verbatim}

- width of type \texttt{signed char}

\begin{verbatim}
  SCHAR\_WIDTH 8
\end{verbatim}

- maximum value for an object of type \texttt{unsigned char}

\begin{verbatim}
  UCHAR\_MAX 255 // 2\textsuperscript{8} - 1
\end{verbatim}

- width of type \texttt{unsigned char}

\begin{verbatim}
  UCHAR\_WIDTH 8
\end{verbatim}

- minimum value for an object of type \texttt{char}

\begin{verbatim}
  CHAR\_MIN see below
\end{verbatim}

- maximum value for an object of type \texttt{char}

\begin{verbatim}
  CHAR\_MAX see below
\end{verbatim}

- width of type \texttt{char}
3 If a file with the same name as one of the above `<` and `>` delimited sequences, not provided as part of the implementation, is placed in any of the standard places that are searched for included source files, the behavior is undefined.

4 Standard headers may be included in any order; each may be included more than once in a given scope, with no effect different from being included only once, except that the effect of including `<assert.h>` depends on the definition of `NDEBUG` (see 7.2). If used, a header shall be included outside of any external declaration or definition, and it shall first be included before the first reference to any of the functions or objects it declares, or to any of the types or macros it defines. However, if an identifier is declared or defined in more than one header, the second and subsequent associated headers may be included after the initial reference to the identifier. The program shall not have any macros with names lexically identical to keywords currently defined prior to the inclusion of the header or when any macro defined in the header is expanded.

5 Some standard headers define or declare identifiers contingent on whether certain macros whose names begin with `__STDC_WANT_IEC_60559_` and end with `__EXT__` are defined (by the user) at the point in the code where the header is first included. Within a preprocessing translation unit, that had not been present in previous versions of this document. To allow implementations and users to adapt to that situation, they also define a version macro for feature test of the form `__STDC_XXXX_VERSION__` which expands to `yyyyyyy`, where `XXXX` is the all-caps spelling of the same set of such macros shall be defined for the first inclusion of all such headers, corresponding header `<xxxx.h>`.

6 Any definition of an object-like macro described in this clause or Annex K shall expand to code that is fully protected by parentheses where necessary, so that it groups in an arbitrary expression as if it were a single identifier.

7 Any declaration of a library function shall have external linkage.

8 A summary of the contents of the standard headers is given in Annex B.

Forward references: diagnostics (7.2).

### 7.1.3 Reserved identifiers

Each header declares or defines all identifiers listed in its associated subclause, and optionally declares or defines identifiers listed in its associated future library directions subclause and identifiers which are always reserved either for any use or for use as file scope identifiers.

- All identifiers that begin with an underscore and either an uppercase letter or another underscore are always reserved for any use, except those identifiers which are lexically identical to keywords.\(^{190}\)

- All identifiers that begin with an underscore are always reserved for use as identifiers with file scope in both the ordinary and tag name spaces.

- Each macro name in any of the following subclauses (including the future library directions) is reserved for use as specified if any of its associated headers is included; unless explicitly stated otherwise (see 7.1.4).

- All identifiers with external linkage in any of the following subclauses (including the future library directions) and `errno` are always reserved for use as identifiers with external linkage.\(^{191}\)

- Each identifier with file scope listed in any of the following subclauses (including the future library directions) is reserved for use as a macro name and as an identifier with file scope in the same name space if any of its associated headers is included.

\(^{190}\) Allows identifiers spelled with a leading underscore followed by an uppercase letter that match the spelling of a keyword to be used as macro names by the program.

\(^{191}\) The list of reserved identifiers with external linkage includes `math_errhandling`, `setjmp`, `va_copy`, and `va_end`. 

---

136 Library § 7.1.3
7.6 Floating-point environment `<fenv.h>`

The header `<fenv.h>` defines several macros, and declares types and functions that provide access to the floating-point environment. The floating-point environment refers collectively to any floating-point status flags and control modes supported by the implementation.\footnote{This header is designed to support the floating-point exception status flags and directed-rounding control modes required by IEC 60559, and other similar floating-point state information. It is also designed to facilitate code portability among all systems.} A floating-point status flag is a system variable whose value is set (but never cleared) when a floating-point exception is raised, which occurs as a side effect of exceptional floating-point arithmetic to provide auxiliary information.\footnote{A floating-point status flag is not an object and can be set more than once within an expression.}

A floating-point control mode is a system variable whose value may be set by the user to affect the subsequent behavior of floating-point arithmetic.\footnote{With these conventions, a programmer can safely assume default floating-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the programmer or program that does so explicitly.}

The dynamic floating-point environment has thread storage duration. The initial state for a thread’s dynamic floating-point environment is the current state of the dynamic floating-point environment of the thread that creates it at the time of creation.

Certain programming conventions support the intended model of use for the dynamic floating-point environment:\footnote{With these conventions, a programmer can safely assume default floating-point control modes (or be unaware of them). The responsibilities associated with accessing the floating-point environment fall on the programmer or program that does so explicitly.}

- a function call does not alter its caller’s floating-point control modes, clear its caller’s floating-point status flags, nor depend on the state of its caller’s floating-point status flags unless the function is so documented;
- a function call is assumed to require default floating-point control modes, unless its documentation promises otherwise;
- a function call is assumed to have the potential for raising floating-point exceptions, unless its documentation promises otherwise.

The following identifiers are defined or declared only if `__STDC_WANT_IEC_60559_BFP_EXT__` is defined as a macro at the point in the source file where is first included:

- `femode_t`
- `FE_DFL_MODE`
- `FE_SNANS_ALWAYS_SIGNAL`
- `fesetexcept`
- `fetestexceptflag`
- `fegetmode`
- `fesetmode`

The feature test macro `__STDC_FENV_VERSION__` expands to the token `yyyymmLL`.

The type

```
fenv_t
```

represents the entire dynamic floating-point environment.

The type

```
femode_t
```

represents the collection of dynamic floating-point control modes supported by the implementation, including the dynamic rounding direction mode.
Each `<math.h>` function listed in the table above indicates the family of functions of all supported types (for example, `acosf` and `acosl` as well as `acos`).

**NOTE** Constant rounding modes (other than `FE_DYNAMIC`) could be implemented using dynamic rounding modes as illustrated in the following example:

```c
{
    #pragma STDC FENV_ROUND direction
    // compiler inserts:
    // #pragma STDC FENV_ACCESS ON
    // int __savedrnd;
    // __savedrnd = __swapround(direction);
    ... operations affected by constant rounding mode ...
    // compiler inserts:
    // __savedrnd = __swapround(__savedrnd);
    ... operations not affected by constant rounding mode ...
    // compiler inserts:
    // __savedrnd = __swapround(__savedrnd);
    ... operations affected by constant rounding mode ...
    // compiler inserts:
    // __swapround(__savedrnd);
}
```

where `__swapround` is defined by:

```c
static inline int __swapround(const int new) {
    const int old = fegetround();
    fesetround(new);
    return old;
}
```

### 7.6.3 Floating-point exceptions

The following functions provide access to the floating-point status flags. The `int` input argument for the functions represents a subset of floating-point exceptions, and can be zero or the bitwise OR of one or more floating-point exception macros, for example `FE_OVERFLOW` | `FE_INEXACT`. For other argument values, the behavior of these functions is undefined.

#### 7.6.3.1 The `feclearexcept` function

**Synopsis**

```c
#include <fenv.h>
int feclearexcept(int excepts);
```

**Description**

The `feclearexcept` function attempts to clear the supported floating-point exceptions represented by its argument.

---

222) The functions `fetestexcept`, `feraiseexcept`, and `feclearexcept` support the basic abstraction of flags that are either set or clear. An implementation can endow floating-point status flags with more information — for example, the address of the code which first raised the floating-point exception; the functions `fegetexceptflag` and `fesetexceptflag` deal with the full content of flags.
7.12 Mathematics <math.h>

The header <math.h> declares two types and many mathematical functions and defines several macros. Most synopses specify a family of functions consisting of a principal function with one or more `double` parameters, a `double` return value, or both; and other functions with the same name but with `f` and `l` suffixes, which are corresponding functions with `float` and `long double` parameters, return values, or both. Integer arithmetic functions and conversion functions are discussed later.

The following identifiers are defined or declared only if `__STDC_WANT_IEC_60559_BFP_EXT__` is defined as a macro at the point in the source file where it is first included:

- `FP_INT_UPWARD`
- `FP_INT_DOWNWARD`
- `FP_INT_TOWARDZERO`
- `FP_INT_TONEARESTFROGMZERO`
- `FP_INT_TONEAREST`
- `FP_LLQGB0`
- `FP_LLQGBNAN`
- `SNANF`
- `SNAN`
- `SNANL`
- `FP_FAST_FADD`
- `FP_FAST_FADDL`
- `FP_FAST_FADDL`
- `FP_FAST_FSUB`
- `FP_FAST_FSUBL`
- `FP_FAST_DSUBL`
- `FP_FAST_FMUL`
- `FP_FAST_FMULL`
- `FP_FAST_DMULL`
- `FP_FAST_FDIV`
- `FP_FAST_FDIVL`
- `FP_FAST_DDIVL`
- `FP_FAST_FMA`
- `FP_FAST_FMAL`
- `FP_FAST_DFMAL`
- `FP_FAST_FSQRT`
- `FP_FAST_FSQRTL`
- `FP_FAST_DSQRTL`
- `iseqsig`
- `iscanonical`
- `issignaling`
- `issubnormal`
- `iszero`
- `fromfp`
- `fromfpf`
- `fromfpl`
- `ufromfp`
- `ufromfpf`
- `ufromfpl`
- `fromfpx`
- `fromfpxf`
- `fromfpxl`
- `ufromfpx`
- `ufromfpxf`
- `ufromfpxl`

particularly on systems with wide expression evaluation, a <math.h> function might pass arguments and return values in wider format than the synopsis prototype indicates.
The types `float_t` and `double_t` are floating types at least as wide as `float` and `double`, respectively, and such that `double_t` is at least as wide as `float_t`. If `FLT_EVAL_METHOD` equals 0, `float_t` and `double_t` are `float` and `double`, respectively; if `FLT_EVAL_METHOD` equals 1, they are both `double`; if `FLT_EVAL_METHOD` equals 2, they are both `long double`; and for other values of `FLT_EVAL_METHOD`, they are otherwise implementation-defined.\(^{235}\)

The macro

```
HUGE_VAL
```

\(^{235}\)The types `float_t` and `double_t` are intended to be the implementation’s most efficient types at least as wide as `float` and `double`, respectively. For `FLT_EVAL_METHOD` equal 0, 1, or 2, the type `float_t` is the narrowest type used by the implementation to evaluate floating expressions.
expands to a positive double constant expression, not necessarily representable as a float. The macros

```c
HUGE_VALF
HUGE_VALL
```

are respectively float and long double analogs of `HUGE_VAL`.\(^{236}\) The macro

```c
INFINITY
```

expands to a constant expression of type float representing positive or unsigned infinity, if available; else to a positive constant of type float that overflows at translation time.\(^{237}\) The macro

```c
NAN
```

is defined if and only if the implementation supports quiet NaNs for the float type. It expands to a constant expression of type float representing a quiet NaN.

The signaling NaN macros

```c
SNANF
SNAN
SNANL
```

each is defined if and only if the respective type contains signaling NaNs (5.2.4.2.2). They expand to a constant expression of the respective type representing a signaling NaN. If a signaling NaN macro is used for initializing an object of the same type that has static or thread-local storage duration, the object is initialized with a signaling NaN value.

The number classification macros

```c
FP_INFINITE
FP_NAN
FP_NORMAL
FP_SUBNORMAL
FP_ZERO
```

represent the mutually exclusive kinds of floating-point values. They expand to integer constant expressions with distinct values. Additional implementation-defined floating-point classifications, with macro definitions beginning with `FP_` and an uppercase letter, may also be specified by the implementation.

The math rounding direction macros

```c
FP_INT_UPWARD
FP_INT_DOWNWARD
FP_INT_TOWARDZERO
FP_INT_TONEARESTFROMZERO
FP_INT_TONEAREST
```

represent the rounding directions of the functions `ceil`, `floor`, `trunc`, `round`, and `roundeven`, respectively, that convert to integral values in floating-point formats. They expand to integer constant expressions with distinct values suitable for use as the second argument to the `fromfp`, `ufromfp`, `fromfpx`, and `ufromfpx`, `toint`, `touint`, `tointx`, and `touintx` functions.

\(^{236}\)`HUGE_VAL`, `HUGE_VALF`, and `HUGE_VALL` can be positive infinities in an implementation that supports infinities.

\(^{237}\)`In this case, using `INFINITY` will violate the constraint in 6.4.4 and thus require a diagnostic.
Description
The `fromfp` and `ufromfp` `toint` and `touint` functions round \( x \), using the math rounding direction indicated by \( \text{round} \), to a signed or unsigned integer, respectively, of \( \text{width} \) bits, and return the result value in the integer type designated by `intmax_t` or `uintmax_t`, respectively. If the value of the round argument is not equal to the value of a math rounding direction macro, the direction of rounding is unspecified. If the value of \( \text{width} \) exceeds the width of the function type, the rounding is to the full width of the function type. The `fromfp` and `ufromfp` `toint` and `touint` functions do not raise the “inexact” floating-point exception. If \( x \) is infinite or \( \text{NaN} \) or rounds to an integral value that is outside the range of any supported integer type\(^{248}\) of the specified width, or if \( \text{width} \) is zero, the functions return an unspecified value and a domain error occurs.

Returns
The `fromfp` and `ufromfp` `toint` and `touint` functions return the rounded integer value.

Example
Upward rounding of `double x` to type `int`, without raising the “inexact” floating-point exception, is achieved by

\[
\begin{align*}
\text{(int)} & \text{fromfp}(x, \ 	ext{FP\_INT\_UPWARD}, \ \text{INT\_WIDTH}) \\
\text{(int)} & \text{toint}(x, \ 	ext{FP\_INT\_UPWARD}, \ \text{INT\_WIDTH})
\end{align*}
\]

7.12.9.11 The `tointx` and `touintx` functions
Synopsis

\[
\begin{align*}
\text{#define } & \text{STDC\_WANT\_IEC\_60559\_BFP\_EXT} \\
& \text{#include } <\text{stdint.h}> \\
& \text{#include } <\text{math.h}> \\
& \text{intmax_t} \text{ fromfp}(\text{double } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width}) \\
& \text{intmax_t} \text{ fromfpf}(\text{float } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width}) \\
& \text{uintmax_t} \text{ ufromfp}(\text{double } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width}) \\
& \text{uintmax_t} \text{ ufromfpf}(\text{float } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width}) \\
& \text{intmax_t} \text{ tointx}(\text{double } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width}) \\
& \text{intmax_t} \text{ tointxf}(\text{float } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width}) \\
& \text{uintmax_t} \text{ touintx}(\text{double } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width}) \\
& \text{uintmax_t} \text{ touintxf}(\text{float } x, \ \text{int } \text{round}, \ \text{unsigned } \text{int} \ \text{width})
\end{align*}
\]

Description
The `fromfp` and `ufromfp` `tointx` and `touintx` functions differ from the `fromfp` and `ufromfp` `toint` and `touint` functions, respectively, only in that the `fromfp` and `ufromfp` `tointx` and `touintx` functions raise the “inexact” floating-point exception if a rounded result not exceeding the specified width differs in value from the argument \( x \).

Returns
The `fromfp` and `ufromfp` `tointx` and `touintx` functions return the rounded integer value.

Note
Conversions to integer types that are not required to raise the inexact exception can be done simply by rounding to integral value in floating type and then converting to the target integer type. For example, the conversion of `long double x` to `uint64_t`, using upward rounding, is done by

\[^{248}\text{For signed types, 6.2.6.2 permits three representations, which differ in whether a value of } - (2^M), \text{ where } M \text{ is the number of value bits, can be represented.}\]
7.12.10 Remainder functions

7.12.10.1 The fmod functions

Synopsis

```
#include <math.h>

double fmod(double x, double y);
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

Description

2 The fmod functions compute the floating-point remainder of $x/y$.

Returns

3 The fmod functions return the value $x - ny$, for some integer $n$ such that, if $y$ is nonzero, the result has the same sign as $x$ and magnitude less than the magnitude of $y$. If $y$ is zero, whether a domain error occurs or the fmod functions return zero is implementation-defined.

7.12.10.2 The remainder functions

Synopsis

```
#include <math.h>

double remainder(double x, double y);
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

Description

2 The remainder functions compute the remainder $x \text{ REM } y$ required by IEC 60559.249

Returns

3 The remainder functions return $x \text{ REM } y$. If $y$ is zero, whether a domain error occurs or the functions return zero is implementation defined.

7.12.10.3 The remquo functions

Synopsis

```
#include <math.h>

double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

Description

2 The remquo functions compute the same remainder as the remainder functions. In the object pointed to by quo they store a value whose sign is the sign of $x/y$ and whose magnitude is congruent modulo $2^n$ to the magnitude of the integral quotient of $x/y$, where $n$ is an implementation-defined integer greater than or equal to 3.

Returns

3 The remquo functions return $x \text{ REM } y$. If $y$ is zero, the value stored in the object pointed to by quo is unspecified and whether a domain error occurs or the functions return zero is implementation defined.

249) “When $y \neq 0$, the remainder $r = x \text{ REM } y$ is defined regardless of the rounding mode by the mathematical relation $r = x - ny$, where $n$ is the integer nearest the exact value of $x/y$; whenever $|n - x/y| = 1/2$, then $n$ is even. If $r = 0$, its sign shall be that of $x$.” This definition is applicable for all implementations.
7.20 Integer types <stdint.h>

1 The header <stdint.h> declares sets of integer types having specified widths, and defines corresponding sets of macros.\(^{275}\) It also defines macros that specify limits of integer types corresponding to types defined in other standard headers.

2 Types are defined in the following categories:

   - integer types having certain exact widths;
   - integer types having at least certain specified widths;
   - fastest integer types having at least certain specified widths;
   - integer types wide enough to hold pointers to objects;
   - integer types having greatest width.

(Some of these types may denote the same type.)

3 Corresponding macros specify limits of the declared types and construct suitable constants.

4 For each type described herein that the implementation provides,\(^{276}\) <stdint.h> shall declare that typedef name and define the associated macros. Conversely, for each type described herein that the implementation does not provide, <stdint.h> shall not declare that typedef name nor shall it define the associated macros. An implementation shall provide those types described as “required”, but need not provide any of the others (described as “optional”).

5 The following identifiers are defined only if \_STDC\_WANT\_IEC\_60559\_BFP\_EXT\_ is defined as a macro at the point in the source file where it is first included:

   - INT_N\_WIDTH
   - UINT_N\_WIDTH
   - INT8\_WIDTH
   - UINT8\_WIDTH
   - INT16\_WIDTH
   - UINT16\_WIDTH
   - INT32\_WIDTH
   - UINT32\_WIDTH
   - INTPTR\_WIDTH
   - UINTPTR\_WIDTH
   - INTPTR\_WIDTH
   - UINTPTR\_WIDTH
   - INTMAX\_WIDTH
   - UINTMAX\_WIDTH
   - PTRDIFF\_WIDTH
   - SIG_ATOMIC\_WIDTH
   - SIZE\_WIDTH
   - WCHAR\_WIDTH
   - WINT\_WIDTH
   - WINT\_WIDTH

The feature test macro \_STDC\_STDINT\_VERSION\_ expands to the token yyyy_mm.

7.20.1 Integer types

1 When typedef names differing only in the absence or presence of the initial u are defined, they shall denote corresponding signed and unsigned types as described in 6.2.5; an implementation providing one of these corresponding types shall also provide the other.

2 In the following descriptions, the symbol N represents an unsigned decimal integer with no leading zeros (e.g., 8 or 24, but not 04 or 048).

7.20.1.1 Exact-width integer types

1 The typedef name int\_N\_t designates a signed integer type with width N, no padding bits, and a two’s complement representation. Thus, int8\_t denotes such a signed integer type with a width of exactly 8 bits.

\(^{275}\)See “future library directions” (7.31.12).

\(^{276}\)Some of these types might denote implementation-defined extended integer types.
7.22 General utilities <stdlib.h>

The header <stdlib.h> declares five types and several functions of general utility, and defines several macros. The following identifiers are declared only if __STDC_WANT_IEC_60559_BFP_EXT__ is defined as a macro at the point in the source file where is first included: strfromd, strfromf, strfroml. The feature test macro __STDC_STDLIB_VERSION__ expands to the token yyyymmLL.

The types declared are size_t and wchar_t (both described in 7.19),

\[
\text{div} \\
\text{ldiv} \\
\text{lldiv}
\]

which is a structure type that is the type of the value returned by the \text{div} function,

\[
\text{ldiv} \\
\text{lldiv}
\]

which is a structure type that is the type of the value returned by the \text{ldiv} function, and

\[
\text{exit} \\
\text{rand} \\
\text{MB_CUR_MAX}
\]

which expands to an integer constant expression that is the maximum value returned by the \text{rand} function; and

\[
\text{exit} \\
\text{exit}
\]

which expand to integer constant expressions that can be used as the argument to the \text{exit} function to return unsuccessful or successful termination status, respectively, to the host environment;

The macros defined are \text{NULL} (described in 7.19);

\[
\text{NULL} \\
\text{NULL}
\]

and

which expands to a positive integer expression with type size_t that is the maximum number of bytes in a multibyte character for the extended character set specified by the current locale (category \text{LC_CTYPE}), which is never greater than MB_LEN_MAX.

7.22.1 Numeric conversion functions

The functions \text{atof}, \text{atoi}, \text{atol}, and \text{atoll} need not affect the value of the integer expression \text{errno} on an error. If the value of the result cannot be represented, the behavior is undefined.

7.22.1.1 The \text{atof} function

Synopsis

```
#include <stdlib.h>

double atof(const char *nptr);
```
Description
2 The \texttt{atof} function converts the initial portion of the string pointed to by \texttt{nptr} to \texttt{double} representation. Except for the behavior on error, it is equivalent to

\verb|strtod(nptr, (char **)NULL)|

Returns
3 The \texttt{atof} function returns the converted value.

Forward references: the \texttt{strtod}, \texttt{strtof}, and \texttt{strtold} functions (7.22.1.4).

7.22.1.2 The \texttt{atoi}, \texttt{atol}, and \texttt{atoll} functions

Synopsis
1

\begin{verbatim}
#include <stdlib.h>
int atoi(const char *nptr);
long int atol(const char *nptr);
long long int atoll(const char *nptr);
\end{verbatim}

Description
2 The \texttt{atoi}, \texttt{atol}, and \texttt{atoll} functions convert the initial portion of the string pointed to by \texttt{nptr} to \texttt{int}, \texttt{long int}, and \texttt{long long int} representation, respectively. Except for the behavior on error, they are equivalent to

\begin{verbatim}
atoi: (int)strtol(nptr, (char **)NULL, 10)
atol: strtol(nptr, (char **)NULL, 10)
atoll: strtoll(nptr, (char **)NULL, 10)
\end{verbatim}

Returns
3 The \texttt{atoi}, \texttt{atol}, and \texttt{atoll} functions return the converted value.

Forward references: the \texttt{strtol}, \texttt{strtoll}, \texttt{strtoul}, and \texttt{strtoull} functions (7.22.1.5).

7.22.1.3 The \texttt{strtod}, \texttt{strtof}, and \texttt{strtold} functions

Synopsis
1

\begin{verbatim}
#define __STDC_WANT_IEC_60559_BFP_EXT__
#include <stdlib.h>
int strftime(char *restrict s, size_t n, const char *restrict format, double fp);
int strftime(char *restrict s, size_t n, const char *restrict format, float fp);
int strftime(char *restrict s, size_t n, const char *restrict format, long double fp);
int tostrd(char *restrict s, size_t n, const char *restrict format, double fp);
int tostrf(char *restrict s, size_t n, const char *restrict format, float fp);
int tostrl(char *restrict s, size_t n, const char *restrict format, long double fp);
\end{verbatim}

Description
2 The \texttt{strftime}, \texttt{strtof}, and \texttt{strtold} functions are equivalent to \texttt{snprintf}(s, n, format, fp) (7.21.6.5), except that the format string shall only contain the character \%', an optional precision that does not contain an asterisk *, and one of the conversion specifiers a, A, e, E, f, F, g, or G, which applies to the type (\texttt{double}, \texttt{float}, or \texttt{long double}) indicated by the function suffix (rather than by a length modifier).

Returns
The \texttt{strftime}, \texttt{strtof}, and \texttt{strtold} functions return the number of characters that would have been written had \texttt{n} been sufficiently large, not counting the terminating null character. Thus, the null-terminated output has been completely written if and only if the returned value is less than \texttt{n}.
7.25 Type-generic math `<tgmath.h>`

The header `<tgmath.h>` includes the headers `<math.h>` and `<complex.h>` and defines several type-generic macros.

The following identifiers are defined as type-generic macros if `__STDC_WANT_IEC_60559_BFP_EXT__` is defined at the point in the source file where it is first included:

- round even
- `llogb`
- `fmaxmag`
- `fminmag`
- `nextup`
- `nextdown`
- `fromfp`
- `ufromfp`
- `fromfpx`
- `ufromfpx`
- `fadd`
- `dadd`
- `fsub`
- `dsub`
- `fmul`
- `dmul`
- `fdiv`
- `ddiv`
- `ffma`
- `dfma`
- `fsqrt`
- `dsqrt`

The feature test macro `__STDC_TGMATH_VERSION__` expands to the token `yyyymm.L`

Of the `<math.h>` and `<complex.h>` functions without an `f` (float) or `l` (long double) suffix, several have one or more parameters whose corresponding real type is `double`. For each such function, except the functions that round result to narrower type (7.12.14) (which are covered below) and `modf`, there is a corresponding type-generic macro.\(^{328}\) The parameters whose corresponding real type is `double` in the function synopsis are `generic parameters`. Use of the macro invokes a function whose corresponding real type and type domain are determined by the arguments for the generic parameters.\(^{329}\)

Use of the macro invokes a function whose generic parameters have the corresponding real type determined as follows:

- First, if any argument for generic parameters has type `long double`, the type determined is `long double`.
- Otherwise, if any argument for generic parameters has type `double` or is of integer type, the type determined is `double`.
- Otherwise, the type determined is `float`.

For each unsuffixed function in `<math.h>` for which there is a function in `<complex.h>` with the same name except for a `c` prefix, the corresponding type-generic macro (for both functions) has the same name as the function in `<math.h>`. The corresponding type-generic macro for `fabs` and `cabs` is `fabs`.

<table>
<thead>
<tr>
<th><code>&lt;math.h&gt;</code> function</th>
<th><code>&lt;complex.h&gt;</code> function</th>
<th>type-generic macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acos</code></td>
<td><code>cacos</code></td>
<td><code>acos</code></td>
</tr>
<tr>
<td><code>asin</code></td>
<td><code>casin</code></td>
<td><code>asin</code></td>
</tr>
<tr>
<td><code>atan</code></td>
<td><code>catan</code></td>
<td><code>atan</code></td>
</tr>
<tr>
<td><code>acosh</code></td>
<td><code>cacosh</code></td>
<td><code>acosh</code></td>
</tr>
<tr>
<td><code>asinh</code></td>
<td><code>casinh</code></td>
<td><code>asinh</code></td>
</tr>
<tr>
<td><code>atanh</code></td>
<td><code>catanh</code></td>
<td><code>atanh</code></td>
</tr>
<tr>
<td><code>cos</code></td>
<td><code>ccos</code></td>
<td><code>cos</code></td>
</tr>
<tr>
<td><code>sin</code></td>
<td><code>csin</code></td>
<td><code>sin</code></td>
</tr>
<tr>
<td><code>tan</code></td>
<td><code>ctan</code></td>
<td><code>tan</code></td>
</tr>
<tr>
<td><code>cosh</code></td>
<td><code>ccosh</code></td>
<td><code>cosh</code></td>
</tr>
<tr>
<td><code>sinh</code></td>
<td><code>csinh</code></td>
<td><code>sinh</code></td>
</tr>
<tr>
<td><code>tanh</code></td>
<td><code>ctanh</code></td>
<td><code>tanh</code></td>
</tr>
<tr>
<td><code>exp</code></td>
<td><code>cexp</code></td>
<td><code>exp</code></td>
</tr>
<tr>
<td><code>log</code></td>
<td><code>clog</code></td>
<td><code>log</code></td>
</tr>
<tr>
<td><code>pow</code></td>
<td><code>cpow</code></td>
<td><code>pow</code></td>
</tr>
<tr>
<td><code>sqrt</code></td>
<td><code>csqrt</code></td>
<td><code>sqrt</code></td>
</tr>
<tr>
<td><code>fabs</code></td>
<td><code>cabs</code></td>
<td><code>fabs</code></td>
</tr>
</tbody>
</table>

If at least one argument for a generic parameter is complex, then use of the macro invokes a complex function; otherwise, use of the macro invokes a real function.

\(^{328}\) Like other function-like macros in standard libraries, each type-generic macro can be suppressed to make available the corresponding ordinary function.

\(^{329}\) If the type of the argument is not compatible with the type of the parameter for the selected function, the behavior is undefined.
For each unsuffixed function in `<math.h>` without a `c`-prefixed counterpart in `<complex.h>` (except functions that round result to narrower type, `modf`, and `canonicalize`), the corresponding type-generic macro has the same name as the function. These type-generic macros are:

- `atan2`, `fdim`, `frexp`, `llrint`, `nearbyint`, `round`
- `cbrt`, `floor`, `_fromfp_toint`, `llround`, `nextafter`, `rroundeven`
- `ceil`, `fma`, `_fromfp_tointx`, `log10`, `ndtdown`, `scalbn`
- `copysign`, `fmax`, `hypot`, `log1p`, `nexttoward`, `scalbln`
- `erf`, `fmaxmag`, `ilogb`, `log2`, `nextup`, `tgamma`
- `erfc`, `fmin`, `ldexp`, `logb`, `remainder`, `trunc`
- `exp2`, `fminmag`, `lgamma`, `lrint`, `remquo`, `_uf ormfp_toint`
- `expm1`, `fmod`, `llogb`, `lround`, `rint`, `_uf ormfp_tointx`

If all arguments for generic parameters are real, then use of the macro invokes a real function; otherwise, use of the macro is undefined.

For each unsuffixed function in `<complex.h>` that is not a `c`-prefixed counterpart to a function in `<math.h>`, the corresponding type-generic macro has the same name as the function. These type-generic macros are:

- `carg`, `cimag`, `conj`, `cproj`, `creal`

Use of the macro with any real or complex argument invokes a complex function.

The functions that round result to a narrower type have type-generic macros whose names are obtained by omitting any `l` suffix\(^3\) from the function names. Thus, the macros are:

- `fadd`, `dadd`, `fsub`, `dsub`, `fmul`, `dmul`, `fdiv`, `ddiv`, `ffma`, `dfma`, `ffsqrt`, `dfsqrt`

All arguments shall be real. If any argument has type `long double`, or if the macro prefix is `d`, the function invoked has the name of the macro with an `l` suffix. Otherwise, the function invoked has the name of the macro (with no suffix).

A type-generic macro corresponding to a function indicated in the table in 7.6.2 is affected by constant rounding modes (7.6.3).

**NOTE** The type-generic macro definition in the example in 6.5.1.1 does not conform to this specification. A conforming macro could be implemented as follows:

```c
#define cbrt(X) _Generic((X),
    long double: cbrel(X),
    default: _Roundwise_cbrt(X),
    float: cbrtf(X))
```

where `_Roundwise_cbrt()` is equivalent to `cbrt()` invoked without macro-replacement suppression.

\(^3\) There are no functions with these macro names and the `f` suffix.
7.31 Future library directions

The following names are grouped under individual headers for convenience. All external names described below are reserved no matter what headers are included by the program.

7.31.1 Complex arithmetic <complex.h>

The function names

```
cerf   cexpm1   clog2
cerfc  clog10   clgamma
cexp2  clog1p   ctgamma
```

and the same names suffixed with f or l may be added to the declarations in the <complex.h> header.

7.31.2 Character handling <ctype.h>

Function names that begin with either is or to, and a lowercase letter may be added to the declarations in the <ctype.h> header.

7.31.3 Errors <errno.h>

Macros that begin with E and a digit or E and an uppercase letter may be added to the macros defined in the <errno.h> header.

7.31.4 Floating-point environment <fenv.h>

Macros that begin with FE_ and an uppercase letter may be added to the macros defined in the <fenv.h> header.

7.31.5 Format conversion of integer types <inttypes.h>

Macros that begin with either PRI or SCN, and either a lowercase letter or X may be added to the macros defined in the <inttypes.h> header.

```
Function names that begin with str, or wcs and a lowercase letter may be added to the declarations in the <inttypes.h> header.
```

7.31.6 Localization <locale.h>

Macros that begin with LC_ and an uppercase letter may be added to the macros defined in the <locale.h> header.

7.31.7 Mathematics <math.h>

```
Function names that begin with either is or to, and a lowercase letter may be added to the declarations in the <math.h> header.
```

```
Macros that begin with DBL_, FLT_, FP_, or LDBL_ and an uppercase letter may be added to the macros defined in the <math.h> header.
```

7.31.8 Signal handling <signal.h>

Macros that begin with either SIG and an uppercase letter or SIG_ and an uppercase letter may be added to the macros defined in the <signal.h> header.

7.31.9 Atomics <stdatomic.h>

Macros that begin with ATOMIC_ and an uppercase letter may be added to the macros defined in the <stdatomic.h> header. Typedef names that begin with either atomic_ or memory_, and a lowercase letter may be added to the declarations in the <stdatomic.h> header. Enumeration constants that begin with memory_order_ and a lowercase letter may be added to the definition of the memory_order type in the <stdatomic.h> header. Function names that begin with atomic_ and a lowercase letter may be added to the declarations in the <stdatomic.h> header.

The macro ATOMIC_VAR_INIT is an obsolescent feature.
7.31.10  Boolean type and values <stdbool.h>
The ability to undefine and perhaps then redefine the macros `bool`, `true`, and `false` is an obsolescent feature.

7.31.11  Integer types <stdint.h>
Typedef names beginning with `int` or `uint` and ending with `_t` may be added to the types defined in the `<stdint.h>` header. Macro names beginning with `INT` or `UINT` and ending with `_MAX`, `_MIN`, `_WIDTH`, or `_C` may be added to the macros defined in the `<stdint.h>` header.

7.31.12  Input/output <stdio.h>
Lowercase letters may be added to the conversion specifiers and length modifiers in `fprintf` and `fscanf`. Other characters may be used in extensions.

2 The use of `ungetc` on a binary stream where the file position indicator is zero prior to the call is an obsolescent feature.

7.31.13  General utilities <stdlib.h>
Function names that begin with `str` or `wcs` and a lowercase letter may be added to the declarations in the `<stdlib.h>` header.

2 Invoking `realloc` with a `size` argument equal to zero is an obsolescent feature.

7.31.14  String handling <string.h>
Function names that begin with `str`, `mem`, or `wcs` and a lowercase letter may be added to the declarations in the `<string.h>` header.

7.31.15  Date and time <time.h>
Macros beginning with `TIME_` and an uppercase letter may be added to the macros in the `<time.h>` header.

7.31.16  Threads <threads.h>
Function names, type names, and enumeration constants that begin with either `cnd_`, `mtx_`, `thrd_`, or `tss_`, and a lowercase letter may be added to the declarations in the `<threads.h>` header.

7.31.17  Extended multibyte and wide character utilities <wchar.h>
Function names that begin with `wcs` and a lowercase letter may be added to the declarations in the `<wchar.h>` header.

2 Lowercase letters may be added to the conversion specifiers and length modifiers in `fwprintf` and `fwscanf`. Other characters may be used in extensions.

7.31.18  Wide character classification and mapping utilities <wctype.h>
Function names that begin with `is` or `to` and a lowercase letter may be added to the declarations in the `<wctype.h>` header.
B.24 Type-generic math <tgmath.h>

<table>
<thead>
<tr>
<th>acos</th>
<th>erf</th>
<th>llround</th>
<th>ufcmpeqfpint</th>
</tr>
</thead>
<tbody>
<tr>
<td>asin</td>
<td>erfc</td>
<td>log10</td>
<td>ufcmpeqfpint</td>
</tr>
<tr>
<td>atan</td>
<td>exp2</td>
<td>log1p</td>
<td>carg</td>
</tr>
<tr>
<td>acosh</td>
<td>expm1</td>
<td>log2</td>
<td>cimag</td>
</tr>
<tr>
<td>asinh</td>
<td>fdim</td>
<td>logb</td>
<td>conj</td>
</tr>
<tr>
<td>atanh</td>
<td>fmod</td>
<td>lrint</td>
<td>cproj</td>
</tr>
<tr>
<td>cos</td>
<td>fma</td>
<td>lround</td>
<td>creal</td>
</tr>
<tr>
<td>sin</td>
<td>fmax</td>
<td>nearbyint</td>
<td>fadd</td>
</tr>
<tr>
<td>tan</td>
<td>fmaxmag</td>
<td>nextafter</td>
<td>dadd</td>
</tr>
<tr>
<td>cosh</td>
<td>fmin</td>
<td>nextdown</td>
<td>fsub</td>
</tr>
<tr>
<td>sinh</td>
<td>fminmag</td>
<td>nexttoward</td>
<td>dsbub</td>
</tr>
<tr>
<td>tanh</td>
<td>fmod</td>
<td>nextup</td>
<td>fmul</td>
</tr>
<tr>
<td>exp</td>
<td>frexp</td>
<td>remainder</td>
<td>dmul</td>
</tr>
<tr>
<td>log</td>
<td>logbreak</td>
<td>remquo</td>
<td>fdiv</td>
</tr>
<tr>
<td>pow</td>
<td>logbfpint</td>
<td>rint</td>
<td>ddiv</td>
</tr>
<tr>
<td>sqrt</td>
<td>hypot</td>
<td>round</td>
<td>ffma</td>
</tr>
<tr>
<td>fabs</td>
<td>ilogb</td>
<td>roundeven</td>
<td>dfma</td>
</tr>
<tr>
<td>atan2</td>
<td>ldexp</td>
<td>scalbn</td>
<td>fsqrt</td>
</tr>
<tr>
<td>cbrt</td>
<td>lgamma</td>
<td>scalbln</td>
<td>dsqrt</td>
</tr>
<tr>
<td>ceil</td>
<td>logb</td>
<td>tgamma</td>
<td></td>
</tr>
<tr>
<td>copysign</td>
<td>llrint</td>
<td>trunc</td>
<td></td>
</tr>
</tbody>
</table>

B.25 Threads <threads.h>

<table>
<thead>
<tr>
<th><strong>STDC_NO_THREADS</strong></th>
<th>mtx_t</th>
<th>thrd_timedout</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread_local</td>
<td>tss_dtor_t</td>
<td>thrd_success</td>
</tr>
<tr>
<td>DECLARE_4_FLAG_INIT</td>
<td>thrd_start_t</td>
<td>thrd_busy</td>
</tr>
<tr>
<td>TSS_DTOR_ITERATIONS</td>
<td>once_flag</td>
<td>thrd_error</td>
</tr>
<tr>
<td>cnd_t</td>
<td>mtx_plain</td>
<td>thrd_nomem</td>
</tr>
<tr>
<td>thrd_t</td>
<td>mtx_recursive</td>
<td></td>
</tr>
<tr>
<td>tss_t</td>
<td>mtx_timed</td>
<td></td>
</tr>
</tbody>
</table>
Annex F  
(normative)  
IEC 60559 floating-point arithmetic

F.1 Introduction


An implementation that defines __STDC_IEC_60559_BFP__ to yyyymmm shall conform to the specifications in this annex and shall also define __STDC_IEC_559__ to 1. Where a binding between the C language and IEC 6059 is indicated, the IEC 6059-specified behavior is adopted by reference, unless stated otherwise.

This annex amends some standard headers with declarations or definitions of identifiers contingent on whether certain macros whose names begin with __STDC_WANT_IEC_60559__ and end with __EXT__ are defined (by the user) at the point in the code where the header is first included. Within a preprocessing translation unit, the same set of such macros shall be defined for the first inclusion of all such headers.

F.2 Types

1 The C floating types match the IEC 60559 formats as follows:

- The float type matches the IEC 60559 binary32 format.
- The double type matches the IEC 60559 binary64 format.
- The long double type matches the IEC 60559 binary128 format, else an IEC 60559 binary64-extended format, else a non-IEC 60559 extended format, else the IEC 60559 binary64 format.

Any non-IEC 60559 extended format used for the long double type shall have more precision than IEC 60559 binary64 and at least the range of IEC 60559 binary64.

The value of FLT_ROUNDS applies to all IEC 60559 types supported by the implementation, but need not apply to non-IEC 60559 types.

Recommended practice

2 The long double type should match the IEC 60559 binary128 format, else an IEC 60559 binary64-extended format.

F.2.1 Infinities and NaNs

1 Since negative and positive infinity are representable in IEC 60559 formats, all real numbers lie within the range of representable values (5.2.4.2.2).

2 The NAN and INFINITY macros and the nan functions in <math.h> provide designations for IEC 60559 quiet NaNs and infinities. The SNANF, SNAN, and SNANL macros in <math.h> provide designations for IEC 60559 signaling NaNs.

373) Implementations that do not define either of __STDC_IEC_60559_BFP__ and __STDC_IEC_559__ are not required to conform to these specifications. New code should not use the obsolescent macro __STDC_IEC_559__ to test for conformance to this annex.

374) IEC 60559 binary64-extended formats include the common 80-bit IEC 60559 format.

375) A non-IEC 60559 long double type is required to provide infinity and NaNs, as its values include all double values.
3 This annex does not require the full support for signaling NaNs specified in IEC 60559. This annex uses the term NaN, unless explicitly qualified, to denote quiet NaNs. Where specification of signaling NaNs is not provided, the behavior of signaling NaNs is implementation-defined (either treated as an IEC 60559 quiet NaN or treated as an IEC 60559 signaling NaN).\(^{376}\)

4 Any operator or `<math.h>` function that raises an “invalid” floating-point exception, if delivering a floating type result, shall return a quiet NaN.

5 In order to support signaling NaNs as specified in IEC 60559, an implementation should adhere to the following recommended practice.

**Recommended practice**

6 Any floating-point operator or `<math.h>` function or macro with a signaling NaN input, unless explicitly specified otherwise, raises an “invalid” floating-point exception.

7 **NOTE** Some functions do not propagate quiet NaN arguments. For example, `hypot(x, y)` returns infinity if \(x\) or \(y\) is infinite and the other is a quiet NaN. The recommended practice in this subclause specifies that such functions (and others) raise the “invalid” floating-point exception if an argument is a signaling NaN, which also implies they return a quiet NaN in these cases.

8 The `<fenv.h>` header defines the macro `FE_SNANS_ALWAYS_SIGNAL` if and only if the implementation follows the recommended practice in this subclause. If defined, `FE_SNANS_ALWAYS_SIGNAL` expands to the integer constant 1.

### F.3 Operations

1 C operators, functions, and function-like macros provide the operations required by IEC 60559 as shown in the following table. Specifications for the C facilities are provided in the listed clauses. The C specifications are intended to match IEC 60559, unless stated otherwise.

#### Operation binding

<table>
<thead>
<tr>
<th>IEC 60559 operation</th>
<th>C operation</th>
<th>Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>roundToIntegralTiesToEven</td>
<td><code>roundeven</code></td>
<td>7.12.9.8, F.10.6.8</td>
</tr>
<tr>
<td>roundToIntegralTiesAway</td>
<td><code>round</code></td>
<td>7.12.9.6, F.10.6.6</td>
</tr>
<tr>
<td>roundToIntegralTowardZero</td>
<td><code>trunc</code></td>
<td>7.12.9.9, F.10.6.9</td>
</tr>
<tr>
<td>roundToIntegralTowardPositive</td>
<td><code>ceil</code></td>
<td>7.12.9.1, F.10.6.1</td>
</tr>
<tr>
<td>roundToIntegralTowardNegative</td>
<td><code>floor</code></td>
<td>7.12.9.2, F.10.6.2</td>
</tr>
<tr>
<td>roundToIntegralExact</td>
<td><code>rint</code></td>
<td>7.12.9.4, F.10.6.4</td>
</tr>
<tr>
<td>nextUp</td>
<td><code>nextup</code></td>
<td>7.12.11.5, F.10.8.5</td>
</tr>
<tr>
<td>nextDown</td>
<td><code>nextdown</code></td>
<td>7.12.11.6, F.10.8.6</td>
</tr>
<tr>
<td>remainder</td>
<td><code>remainder</code>, <code>remquo</code></td>
<td>7.12.10.2, F.10.7.2,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.12.10.3, F.10.7.3</td>
</tr>
<tr>
<td>minNum</td>
<td><code>fmin</code></td>
<td>7.12.12.3, F.10.9.3</td>
</tr>
<tr>
<td>maxNum</td>
<td><code>fmax</code></td>
<td>7.12.12.2, F.10.9.2</td>
</tr>
<tr>
<td>minNumMag</td>
<td><code>fminmag</code></td>
<td>7.12.12.5, F.10.9.5</td>
</tr>
<tr>
<td>maxNumMag</td>
<td><code>fmaxmag</code></td>
<td>7.12.12.4, F.10.9.4</td>
</tr>
<tr>
<td>scaleB</td>
<td><code>scalbn</code>, <code>scalbln</code></td>
<td>7.12.6.14, F.10.3.14</td>
</tr>
<tr>
<td>logB</td>
<td><code>logb</code>, <code>ilogb</code>, <code>llogb</code></td>
<td>7.12.6.12, F.10.3.12,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.12.6.5, F.10.3.5,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.12.6.7, F.10.3.7</td>
</tr>
<tr>
<td>addition</td>
<td><code>+</code>, <code>fadd</code>, <code>faddl</code>, <code>daddl</code></td>
<td>6.5.6, 7.12.14.1,  F.10.11</td>
</tr>
<tr>
<td>subtraction</td>
<td><code>-</code>, <code>fsub</code>, <code>fsubl</code>, <code>dsubl</code></td>
<td>6.5.6, 7.12.14.2, F.10.11</td>
</tr>
<tr>
<td>multiplication</td>
<td><code>*</code>, <code>fmul</code>, <code>fmul</code>, <code>dmull</code></td>
<td>6.5.5, 7.12.14.3, F.10.11</td>
</tr>
</tbody>
</table>

\(^{376}\)Since NaNs created by IEC 60559 operations are always quiet, quiet NaNs (along with infinities) are sufficient for closure of the arithmetic.
<table>
<thead>
<tr>
<th>Function / Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>division</td>
<td><code>/</code>, <code>fdiv</code>, <code>fdivl</code>, <code>ddivl</code></td>
</tr>
<tr>
<td>squareRoot</td>
<td><code>sqrt</code>, <code>fsqrt</code>, <code>fsqrtl</code>, <code>dsqrtl</code></td>
</tr>
<tr>
<td>fusedMultiplyAdd</td>
<td><code>fma</code>, <code>ffma</code>, <code>ffmal</code>, <code>dfmal</code></td>
</tr>
<tr>
<td>convertToInt</td>
<td>cast and implicit conversion</td>
</tr>
<tr>
<td>convertToIntTiesToEven</td>
<td><code>fromfp</code>, <code>ufromfp</code>, <code>toint</code>, <code>touint</code></td>
</tr>
<tr>
<td>convertToIntTowardZero</td>
<td><code>fromfp</code>, <code>ufromfp</code>, <code>toint</code>, <code>touint</code></td>
</tr>
<tr>
<td>convertToIntTiesToAway</td>
<td><code>fromfp</code>, <code>ufromfp</code>, <code>lround</code>, <code>llround</code></td>
</tr>
<tr>
<td>convertToIntExactTiesToEven</td>
<td><code>fromfp</code>, <code>ufromfp</code>, <code>tointx</code>, <code>touintx</code></td>
</tr>
<tr>
<td>convertToDecimalCharacter</td>
<td><code>strtod</code>, <code>wcstod</code>, <code>scanf</code>, <code>wscanf</code>, <code>strfromd</code>, <code>tostrd</code></td>
</tr>
<tr>
<td>convertToHexCharacter</td>
<td><code>strtod</code>, <code>wcstod</code>, <code>scanf</code>, <code>wscanf</code>, <code>strfromd</code>, <code>tostrd</code></td>
</tr>
<tr>
<td>copy</td>
<td><code>memcpy</code>, <code>memmove</code></td>
</tr>
<tr>
<td>negate</td>
<td><code>-(x)</code></td>
</tr>
<tr>
<td>abs</td>
<td><code>fabs</code></td>
</tr>
<tr>
<td>copySign</td>
<td><code>copysign</code></td>
</tr>
<tr>
<td>compareQuietEqual</td>
<td><code>==</code></td>
</tr>
<tr>
<td>compareQuietNotEqual</td>
<td><code>!=</code></td>
</tr>
<tr>
<td>compareSignalingEqual</td>
<td><code>iseqsig</code></td>
</tr>
<tr>
<td>compareSignalingGreaterEqual</td>
<td><code>&gt;=</code></td>
</tr>
<tr>
<td>compareSignalingLessEqual</td>
<td><code>&lt;=</code></td>
</tr>
<tr>
<td>compareSignalingLess</td>
<td><code>! (x &gt; y)</code></td>
</tr>
<tr>
<td>compareSignalingLessUnordered</td>
<td><code>! (x &gt;= y)</code></td>
</tr>
<tr>
<td>compareSignalingGreaterUnordered</td>
<td><code>! (x &lt; y)</code></td>
</tr>
<tr>
<td>compareSignalingGreaterUnordered</td>
<td><code>! (x &lt;= y)</code></td>
</tr>
<tr>
<td>compareQuietGreater</td>
<td><code>isgreater</code></td>
</tr>
<tr>
<td>compareQuietGreaterEqual</td>
<td><code>isgreaterequal</code></td>
</tr>
<tr>
<td>compareQuietLess</td>
<td><code>isless</code></td>
</tr>
<tr>
<td>compareQuietLessEqual</td>
<td><code>islessequal</code></td>
</tr>
<tr>
<td>compareQuietUnordered</td>
<td><code>isunordered</code></td>
</tr>
<tr>
<td>compareQuietNotGreater</td>
<td><code>! isgreater(x, y)</code></td>
</tr>
<tr>
<td>compareQuietNotGreaterUnordered</td>
<td><code>! isgreaterequal(x, y)</code></td>
</tr>
<tr>
<td>compareQuietNotLessUnordered</td>
<td><code>! islessequal(x, y)</code></td>
</tr>
<tr>
<td>compareQuietNotLessUnordered</td>
<td><code>! isless(x, y)</code></td>
</tr>
</tbody>
</table>

§ F.3 IEC 60559 floating-point arithmetic 393
2 The IEC 60559 requirement that certain of its operations be provided for operands of different formats (of the same radix) is satisfied by C’s usual arithmetic conversions (6.3.1.8) and function-call argument conversions (6.5.2.2). For example, the following operations take float \( f \) and double \( d \) inputs and produce a long double result:

\[
(\text{long double})f + d = \text{powl}(f, d)
\]

3 Whether C assignment (6.5.16) (and conversion as if by assignment) to the same format is an IEC 60559 convertFormat or copy operation\(^{377}\) is implementation-defined, even if \(<fenv.h>\) defines the macro \texttt{FE_SNANS_ALWAYS_SIGNAL} (F.2.1). If the return expression of a \texttt{return} statement is evaluated to the floating-point format of the return type, it is implementation-defined whether a convertFormat operation is applied to the result of the return expression.

4 The unary \texttt{-} operator raises no floating-point exceptions, even if the operand is a signaling NaN.

5 The C classification macros \texttt{fpclassify, iscanonical, isfinite, isninf, isnan, isnormal, issignaling, issubnormal}, and \texttt{iszero} provide the IEC 60559 operations indicated in the table above provided their arguments are in the format of their semantic type. Then these macros raise no floating-point exceptions, even if an argument is a signaling NaN.

6 The C \texttt{nearbyint} functions (7.12.9.3, F.10.6.3) provide the nearbyinteger function recommended in the Appendix to (superseded) ANSI/IEEE 854.

7 The C \texttt{nextafter} (7.12.11.3, F.10.8.3) and \texttt{nexttoward} (7.12.11.4, F.10.8.4) functions provide the

\(^{377}\)Where the source and destination formats are the same, convertFormat operations differ from copy operations in that convertFormat operations raise the “invalid” floating-point exception on signaling NaN inputs and do not propagate non-canonical encodings.
nextafter function recommended in the Appendix to (superseded) IEC 60559:1989 (but with a minor change to better handle signed zeros).

8 The C getpayload, setpayload, and setpayloadsig (F.10.13) functions provide program access to NaN payloads, defined in IEC 60559.

9 The macros (7.6) FE_DOWNWARD, FE_TONEAREST, FE_TOWARDZERO, and FE_UPWARD, which are used in conjunction with the fegetround and fesetround functions and the FENV_ROUND pragma, represent the IEC 60559 rounding-direction attributes roundTowardNegative, roundTiesToEven, roundTowardZero, and roundTowardPositive, respectively.

10 The C fegetenv (7.6.5.1), feholdexcept (7.6.5.2), fesetenv (7.6.5.3) and feupdateenv (7.6.5.4) functions provide a facility to manage the dynamic floating-point environment, comprising the IEC 60559 status flags and dynamic control modes.

11 IEC 60559 requires operations with specified operand and result formats. Therefore, math functions that are bound to IEC 60559 operations (see table above) must remove any extra range and precision from arguments or results.

12 IEC 60559 requires operations that round their result to formats the same as and wider than the operands, in addition to the operations that round their result to narrower formats (see 7.12.14). Operators (+, -, *, and /) whose evaluation formats are wider than the semantic type (5.2.4.2.2) might not support some of the IEEE 60559 operations, because getting a result in a given format might require a cast that could introduce an extra rounding error. The functions that round result to narrower type (7.12.14) provide the IEC 60559 operations that round result to same and wider (as well as narrower) formats, in those cases where built-in operators and casts do not. For example, ddivl(x, y) computes a correctly rounded double divide of float x by float y, regardless of the evaluation method.

F.4 Floating to integer conversion
1 If the integer type is _Bool, 6.3.1.2 applies and the conversion raises no floating-point exceptions if the floating-point value is not a signaling NaN. Otherwise, if the floating value is infinite or NaN or if the integral part of the floating value exceeds the range of the integer type, then the “invalid” floating-point exception is raised and the resulting value is unspecified. Otherwise, the resulting value is determined by 6.3.1.4. Conversion of an integral floating value that does not exceed the range of the integer type raises no floating-point exceptions; whether conversion of a non-integral floating value raises the “inexact” floating-point exception is unspecified.\(^{378}\)

F.5 Conversions between binary floating types and decimal character sequences
1 Conversion from the widest supported IEC 60559 format to decimal with DECIMAL_DIG digits and back is the identity function.\(^{379}\)

2 Conversions involving IEC 60559 formats follow all pertinent recommended practice. In particular, conversion between any supported IEC 60559 format and decimal with DECIMAL_DIG or fewer significant digits is correctly rounded (honoring the current rounding mode), which assures that conversion from the widest supported IEC 60559 format to decimal with DECIMAL_DIG digits and back is the identity function.

3 The <float.h> header defines the macro

```c
#define CR_DECIMAL_DIG
```

if and only if __STDC_WANT_IEC_60559_BFP_EXT__ is defined as a macro at the point in the source

\(^{378}\)IEC 60559 recommends that implicit floating-to-integer conversions raise the “inexact” floating-point exception for non-integer in-range values. In those cases where it matters, library functions can be used to effect such conversions with or without raising the “inexact” floating-point exception. See fromfp, ufromfp, fromfpf, fromfpf_toint, fromfpf_tointx, ufromfp, ufromfpf, toint, lrint, fround, and nearbyint in <math.h>.

\(^{379}\)If the minimum-width IEC 60559 binary64-extended format (64 bits of precision) is supported, DECIMAL_DIG is at least 21. If IEC 60559 binary64 (53 bits of precision) is the widest IEC 60559 format supported, then DECIMAL_DIG is at least 17. (By contrast, LDBL_DIG and DBL_DIG are 18 and 15, respectively, for these formats.)
file where `<float.h>` is first included. If defined, `CR_DECIMAL_DIG` expands to an integral constant expression suitable for use in `#if` preprocessing directives whose value is a number such that conversions between all supported types with IEC 60559 binary formats and character sequences with at most `CR_DECIMAL_DIG` significant decimal digits are correctly rounded. The value of `CR_DECIMAL_DIG` shall be at least `DECIMAL_DIG` + 3. If the implementation correctly rounds for all numbers of significant decimal digits, then `CR_DECIMAL_DIG` shall have the value of the macro `UINTMAX_MAX`.

4 Conversions of types with IEC 60559 binary formats to character sequences with more than `CR_DECIMAL_DIG` significant decimal digits shall correctly round to `CR_DECIMALDIG` significant digits and pad zeros on the right.

5 Conversions from character sequences with more than `CR_DECIMAL_DIG` significant decimal digits to types with IEC 60559 binary formats shall correctly round to an intermediate character sequence with `CR_DECIMAL_DIG` significant decimal digits, according to the applicable rounding direction, and correctly round the intermediate result (having `CR_DECIMAL_DIG` significant decimal digits) to the destination type. The “inexact” floating-point exception is raised (once) if either conversion is inexact.380) (The second conversion may raise the “overflow” or “underflow” floating-point exception.)

6 Functions such as `strtod` that convert character sequences to floating types honor the rounding direction. Hence, if the rounding direction might be upward or downward, the implementation cannot convert a minus-signed sequence by negating the converted unsigned sequence.

7 The `fprintf` family of functions in `<stdio.h>` and the `fwprintf` family of functions in `<wchar.h>` should behave as if floating-point operands were passed through the `canonicalize` function of the same type.381)

F.6 The return statement

If the return expression is evaluated in a floating-point format different from the return type, the expression is converted as if by assignment382) to the return type of the function and the resulting value is returned to the caller.

F.7 Contracted expressions

1 A contracted expression is correctly rounded (once) and treats infinities, NaNs, signed zeros, subnormals, and the rounding directions in a manner consistent with the basic arithmetic operations covered by IEC 60559.

Recommended practice

2 A contracted expression should raise floating-point exceptions in a manner generally consistent with the basic arithmetic operations.

F.8 Floating-point environment

1 The floating-point environment defined in `<fenv.h>` includes the IEC 60559 floating-point exception status flags and directed-rounding control modes. It includes also IEC 60559 dynamic rounding precision and trap enablement modes, if the implementation supports them.383)

F.8.1 Environment management

1 IEC 60559 requires that floating-point operations implicitly raise floating-point exception status flags, and that rounding control modes can be set explicitly to affect result values of floating-point operations. These changes to the floating-point state are treated as side effects which respect sequence points.384)

380) The intermediate conversion is exact only if all input digits after the first `CR_DECIMAL_DIG` digits are 0.

381) This is a recommendation instead of a requirement so that implementations may choose to print signaling NaNs differently from quiet NaNs.

382) Assignment removes any extra range and precision.

383) This specification does not require dynamic rounding precision nor trap enablement modes.

384) If the state for the `FENV_ACCESS` pragma is “off”, the implementation is free to assume the dynamic floating-point control modes will be the default ones and the floating-point status flags will not be tested, which allows certain optimizations (see...
— \texttt{trunc}(\pm 0) \text{ returns } \pm 0.
— \texttt{trunc}(\pm \infty) \text{ returns } \pm \infty.

2 The returned value is exact and is independent of the current rounding direction mode.

F.10.6.10 The \texttt{toint} and \texttt{touint} functions
1 The \texttt{fromfp} and \texttt{ufromfp} \texttt{toint} and \texttt{touint} functions raise the “invalid” floating-point exception and return an unspecified value if the floating-point argument \( x \) is infinite or NaN or rounds to an integral value that is outside the range of any supported integer type of the specified width.

2 These functions do not raise the “inexact” floating-point exception.

F.10.6.11 The \texttt{tointx} and \texttt{touintx} functions
1 The \texttt{fromfp} and \texttt{ufromfp} \texttt{tointx} and \texttt{touintx} functions raise the “invalid” floating-point exception and return an unspecified value if the floating-point argument \( x \) is infinite or NaN or rounds to an integral value that is outside the range of any supported integer type of the specified width.

2 These functions raise the “inexact” floating-point exception if a valid result differs in value from the floating-point argument \( x \).

F.10.7 Remainder functions
F.10.7.1 The \texttt{fmod} functions
1 — \texttt{fmod}(\pm 0, y) \text{ returns } \pm 0 \text{ for } y \text{ not zero.}
— \texttt{fmod}(x, y) \text{ returns a NaN and raises the “invalid” floating-point exception for } x \text{ infinite or } y \text{ zero (and neither is a NaN).}
— \texttt{fmod}(x, \pm \infty) \text{ returns } x \text{ for } x \text{ not infinite.}

2 When subnormal results are supported, the returned value is exact and is independent of the current rounding direction mode.

3 The \texttt{double} version of \texttt{fmod} behaves as though implemented by

```c
#include <math.h>
#include <fenv.h>
#pragma STDC FENV_ACCESS ON

double fmod(double x, double y)
{
    double result;
    result = remainder(fabs(x), (y = fabs(y)));
    if (signbit(result)) result += y;
    return copysign(result, x);
}
```

F.10.7.2 The \texttt{remainder} functions
1 — \texttt{remainder}(\pm 0, y) \text{ returns } \pm 0 \text{ for } y \text{ not zero.}
— \texttt{remainder}(x, y) \text{ returns a NaN and raises the “invalid” floating-point exception for } x \text{ infinite or } y \text{ zero (and neither is a NaN).}
— \texttt{remainder}(x, \pm \infty) \text{ returns } x \text{ for } x \text{ not infinite.}

2 When subnormal results are supported, the returned value is exact and is independent of the current rounding direction mode.

F.10.7.3 The \texttt{remquo} functions
1 The \texttt{remquo} functions follow the specifications for the \texttt{remainder} functions. They have no further specifications special to IEC 60559 implementations.

2 When subnormal results are supported, the returned value is exact and is independent of the current rounding direction mode.
J.5.11 Multiple external definitions
1 There may be more than one external definition for the identifier of an object, with or without the explicit use of the keyword extern; if the definitions disagree, or more than one is initialized, the behavior is undefined (6.9.2).

J.5.12 Predefined macro names
1 Macro names that do not begin with an underscore, describing the translation and execution environments, are defined by the implementation before translation begins (6.10.8).

J.5.13 Floating-point status flags
1 If any floating-point status flags are set on normal termination after all calls to functions registered by the atexit function have been made (see 7.22.4.4), the implementation writes some diagnostics indicating the fact to the stderr stream, if it is still open,

J.5.14 Extra arguments for signal handlers
1 Handlers for specific signals are called with extra arguments in addition to the signal number (7.14.1.1).

J.5.15 Additional stream types and file-opening modes
1 Additional mappings from files to streams are supported (7.21.2).
2 Additional file-opening modes may be specified by characters appended to the mode argument of the fopen function (7.21.5.3).

J.5.16 Defined file position indicator
1 The file position indicator is decremented by each successful call to the ungetc or ungetwc function for a text stream, except if its value was zero before a call (7.21.7.10, 7.29.3.10).

J.5.17 Math error reporting
1 Functions declared in <complex.h> and <math.h> raise SIGFPE to report errors instead of, or in addition to, setting errno or raising floating-point exceptions (7.3, 7.12).

J.6 Reserved identifiers and keywords
1 A lot of identifier preprocessing tokens are used for specific purposes in regular clauses or appendices from translation phase 3 onwards. Using any of these for a purpose different from their description in this document, even if the use is in a context where they are normatively permitted, may have an impact on the portability of code and should thus be avoided.

J.6.1 Rule based identifiers
1 The following regular expressions characterize identifiers that are systematically reserved by some clause this document.

atomic_[a-z][a-zA-Z0-9-]*
ATOMIC_[A-Z][a-zA-Z0-9-]*
_[a-zA-Z]_[a-zA-Z0-9-]*
cnd_[a-z]_[a-zA-Z0-9-]*
DBL_[A-Z]_[a-zA-Z0-9-]*
E_[0-9A-Z]_[a-zA-Z0-9-]*
FE__[A-Z]_[a-zA-Z0-9-]*
FLT_[A-Z]_[a-zA-Z0-9-]*
FP__[A-Z]_[a-zA-Z0-9-]*
INT_[a-zA-Z0-9-]*_C
INT_[a-zA-Z0-9-]*_MAX
INT_[a-zA-Z0-9-]*_MIN
int_[a-zA-Z0-9-]*_t
INT_[a-zA-Z0-9-]*_WIDTH
2 The following 462 identifiers or keywords match these patterns and have particular semantics provided by this document.

- _Alignas
- __alignas_is_defined
- _Alignof
- __alignof_is_defined
- _Atomic
- atomic_bool
- atomic_char
- atomic_char16_t
- atomic_char32_t
- atomic_compare_exchange_strong
- atomic_compare_exchange_strong_explicit
- atomic_compare_exchange_weak
- atomic_compare_exchange_weak_explicit
- atomic_exchange
- atomic_exchange_explicit
- atomic_fetch_
- atomic_fetch_add
- atomic_fetch_add_explicit
- atomic_fetch_and
- atomic_fetch_and_explicit
- atomic_fetch_or
- atomic_fetch_or_explicit
- atomic_fetch_sub
- atomic_fetch_sub_explicit
- atomic_fetch_xor
- atomic_fetch_xor_explicit
- atomic_flag
- atomic_flag_clear
- atomic_flag_compare_exchange
- atomic_flag_compare_exchange_explicit
- atomic_flag_init
- atomic_flag_test_and_set
- atomic_flag_test_and_set_explicit
- atomic_init
- atomic_int
- atomic_int_fast16_t
- atomic_int_fast32_t
- atomic_int_fast64_t
- atomic_int_fast8_t
- atomic_int_least16_t
- atomic_int_least32_t
- atomic_int_least64_t
- atomic_int_least8_t
- atomic_intmax_t
- atomic_intptr_t
- atomic_uchar
- atomic_uint
- atomic_uint_fast16_t
- atomic_uint_fast32_t
- atomic_uint_fast64_t
- atomic_uint_fast8_t
- atomic_uint_least16_t
- atomic_uint_least32_t
- atomic_uint_least64_t
- atomic_uint_least8_t
- atomic_uintmax_t
- atomic_uintptr_t
- atomic_ulong
- atomic_ulonglong
- atomic_ushort
- atomic_wchar_t
- atomic_wchar16_t
- atomic_wchar32_t
- atomic_wchar_t_lock_free
- _Bool
- __bool_false_are_defined
- cnd_broadcast
- cnd_destroy
- cnd_init
- cnd_signal
- cnd_t
- cnd_timedwait
- cnd_wait
- Complex
- __DBL_DECIMAL_DIG
- __DBL_DIG
- __DBL_EPSILON
- __DATE__
- __DBL_HAS_SUBNORM
- __DBL_MANT_DIG
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBL_MAX</td>
<td>Floating-point maximum value</td>
</tr>
<tr>
<td>DBL_MAX_10_EXP</td>
<td>Floating-point maximum value in base 10 notation</td>
</tr>
<tr>
<td>DBL_MAX_EXP</td>
<td>Floating-point maximum value</td>
</tr>
<tr>
<td>DBL_MIN</td>
<td>Floating-point minimum value</td>
</tr>
<tr>
<td>DBL_MIN_10_EXP</td>
<td>Floating-point minimum value in base 10 notation</td>
</tr>
<tr>
<td>DBL_MIN_EXP</td>
<td>Floating-point minimum value</td>
</tr>
<tr>
<td>DBL_TRUE_MIN</td>
<td>True minimum value</td>
</tr>
<tr>
<td>EDOM</td>
<td>Floating-point domain error</td>
</tr>
<tr>
<td>EILSEQ</td>
<td>Floating-point invalid error</td>
</tr>
<tr>
<td>EOF</td>
<td>End of file error</td>
</tr>
<tr>
<td>EOL</td>
<td>End of line error</td>
</tr>
<tr>
<td>ERANGE</td>
<td>Floating-point range error</td>
</tr>
<tr>
<td>_Exit</td>
<td>Exit function</td>
</tr>
<tr>
<td>EXIT_FAILURE</td>
<td>Exit failure</td>
</tr>
<tr>
<td>EXIT_SUCCESS</td>
<td>Exit success</td>
</tr>
<tr>
<td>FE_ALL_EXCEPT</td>
<td>All floating-point exceptions</td>
</tr>
<tr>
<td>FE_DFL_ENV</td>
<td>Default floating-point environment</td>
</tr>
<tr>
<td>FE_DFL_MODE</td>
<td>Default floating-point mode</td>
</tr>
<tr>
<td>FE_DIVBYZERO</td>
<td>Divide by zero</td>
</tr>
<tr>
<td>FE_DOWNWARD</td>
<td>Downward rounding</td>
</tr>
<tr>
<td>FE_DYNAMIC</td>
<td>Dynamic floating-point</td>
</tr>
<tr>
<td>FE_INEXACT</td>
<td>Inexact operation</td>
</tr>
<tr>
<td>FE_INVALID</td>
<td>Invalid operation</td>
</tr>
<tr>
<td>FE_OVERFLOW</td>
<td>Overflow</td>
</tr>
<tr>
<td>FE_SNANS_ALWAYS_SIGNAL</td>
<td>Signal NaNs always signal</td>
</tr>
<tr>
<td>FE_TONEAREST</td>
<td>Toned nearest</td>
</tr>
<tr>
<td>FE_TOWARDZERO</td>
<td>Toward zero</td>
</tr>
<tr>
<td>FE_UNDERFLOW</td>
<td>Underflow</td>
</tr>
<tr>
<td>_FILE</td>
<td>File function</td>
</tr>
<tr>
<td>FLT_DECIMAL_DIG</td>
<td>Floating-point decimal digits</td>
</tr>
<tr>
<td>_func</td>
<td>Function function</td>
</tr>
<tr>
<td>_Generic</td>
<td>Generic function</td>
</tr>
<tr>
<td>_Imaginary</td>
<td>Imaginary function</td>
</tr>
<tr>
<td>_ImaginaryFLT_EVAL_METHOD</td>
<td>Imaginary floating-point evaluation method</td>
</tr>
<tr>
<td>_ImaginaryFLT_HAS_SUBNORM</td>
<td>Imaginary floating-point has subnormal</td>
</tr>
<tr>
<td>FLT_MANT_DIG</td>
<td>Floating-point mantissa digits</td>
</tr>
<tr>
<td>FLT_MAX</td>
<td>Floating-point maximum</td>
</tr>
<tr>
<td>FLT_MAX_10_EXP</td>
<td>Floating-point maximum in base 10 notation</td>
</tr>
<tr>
<td>FLT_MAX_EXP</td>
<td>Floating-point maximum</td>
</tr>
<tr>
<td>FLT_MIN</td>
<td>Floating-point minimum</td>
</tr>
<tr>
<td>FLT_MIN_10_EXP</td>
<td>Floating-point minimum in base 10 notation</td>
</tr>
<tr>
<td>FLT_MIN_EXP</td>
<td>Floating-point minimum</td>
</tr>
<tr>
<td>FLT_RADIX</td>
<td>Floating-point radix</td>
</tr>
<tr>
<td>FLT_ROUNDS</td>
<td>Floating-point rounding</td>
</tr>
<tr>
<td>FLT_TRUE_MIN</td>
<td>True minimum value</td>
</tr>
<tr>
<td>FP_CONTRACT</td>
<td>Floating-point contract</td>
</tr>
<tr>
<td>FP_FAST_DADD</td>
<td>Fast addition function</td>
</tr>
<tr>
<td>FP_FAST_DDIV</td>
<td>Fast division function</td>
</tr>
<tr>
<td>FP_FAST_DFMAL</td>
<td>Fast floating-point multiplication</td>
</tr>
<tr>
<td>FP_FAST_DMUL</td>
<td>Fast floating-point multiplication</td>
</tr>
<tr>
<td>FP_FAST_DSQRT</td>
<td>Fast square root function</td>
</tr>
<tr>
<td>FP_FAST_DSUBL</td>
<td>Fast subtraction function</td>
</tr>
<tr>
<td>FP_FAST_FADD</td>
<td>Fast addition function</td>
</tr>
<tr>
<td>FP_FAST_FADDL</td>
<td>Fast floating-point addition function</td>
</tr>
<tr>
<td>FP_FAST_FDIV</td>
<td>Fast division function</td>
</tr>
<tr>
<td>FP_FAST_FDIVL</td>
<td>Fast floating-point division function</td>
</tr>
<tr>
<td>FP_FAST_FFMAL</td>
<td>Fast floating-point multiplication</td>
</tr>
<tr>
<td>FP_FAST_FMA</td>
<td>Fast multiplication function</td>
</tr>
<tr>
<td>FP_FAST_FMAF</td>
<td>Fast floating-point multiplication</td>
</tr>
<tr>
<td>FP_FAST_FMFULL</td>
<td>Fast floating-point multiplication</td>
</tr>
<tr>
<td>FP_FAST_FSUB</td>
<td>Fast subtraction function</td>
</tr>
<tr>
<td>FP_FAST_FSUBL</td>
<td>Fast subtraction function</td>
</tr>
<tr>
<td>FP_FAST_FSQRT</td>
<td>Fast square root function</td>
</tr>
<tr>
<td>FP_FAST_FSQRTL</td>
<td>Fast square root function</td>
</tr>
<tr>
<td>FP_FAST_ILOGB</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBNAN</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGB0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBNAN</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGB0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBF</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBF0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBF0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFN</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFNC</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFNC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFNC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFN0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFN0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFN0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
<tr>
<td>FP_FAST_ILOGBFC0</td>
<td>Fast base 10 logarithm</td>
</tr>
</tbody>
</table>
INTPTR_MAX
INTPTR_MIN
 intptr_t
INTPTR_WIDTH
INT_WIDTH
_IOFBF
_IOLBF
_ONEBF
isalnum
isalpha
isblank
iscanonical
iscntrl
isdigit
iseqsig
finite
isgraph
isgreater
isgreaterequal
isinf
isless
islessequal
islessgreater
islower
isnan
isnormal
isprint
ispunct
issignaling
isspace
issubnormal
isunordered
isupper
iswalnum
iswalpha
iswblank
iswctrl
iswctype
iswdigit
iswgraph
iswlower
iswprint
iswpunct
iswspace
iswupper
iswxdigit
isxdigit
iszerodigit
iszeroword
LC_ALL
LC_COLLATE
LC_CTYPE
LC_MONETARY
LC_NUMERIC
LC_TIME
_LINE__LDBL_DECIMAL_DIG
LDBL_DECIMAL_DIG
LDBL_DIG
LDBL_EPSILON
LDBL_HAS_SUBNORM
LDBL_MANT_DIG
LDBL_MAX
LDBL_MAX_10_EXP
LDBL_MAX_EXP
LDBL_MIN
LDBL_MIN_10_EXP
LDBL_MIN_EXP
LDBL_TRUE_MIN
memchr
memcpy
memcmp
memcpy_s
memmove
memmove_s
memory_order
memory_order_acquire
memory_order_acquire
memory_order_acq_rel
memory_order_consume
memory_order_consume
memory_order_acquire
memory_order_acquire
memory_order_acquire
memory_order_consume
memory_order_consume
memory_order_release
memory_order_release
memory_order_acq_rel
memory_order_acq_rel
memory_order_release
memory_order_release
memory_order_seq_cst
memory_order_seq_cst
memcpy
mtx_destroy
mtx_init
mtx_lock
mtx_plain
mtx_recursive
mtx_t
mtx_timed
mtx_timedlock
mtx_trylock
mtx_unlock
_Noreturn
__Pragma
PRIId32
PRIId64
PRIIdFAST32
PRIIdFAST64
PRIIdLEAST32
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMAX
PRIIdLEAST64
PRIIdLEAST64
PRIIdMA
PRIoFAST64 __STDC_NO_THREADS__
PRIoLEAST32 __STDC_NO_VLA__
PRIoLEAST64 __STDC_UTF_16__
PRIoMAX __STDC_UTF_32__
PRIoPTR __STDC_VERSION__
PRIu32 __STDC_WANT_IEC_60559__
PRIu64 __STDC_WANT_IEC_60559_BFP_EXT__
PRIuFAST32 __STDC_WANT_LIB_EXT1__
PRIuFAST64 strcat
PRIuLEAST32 strcat_s
PRIuLEAST64 strchr
PRIuMAX strcmp
PRIuPTR strcoll
PRIx32 strcpys
PRIx64 strcpys
PRIXFAST32 strcspn
PRIXFAST64 sterror
PRIXLEAST32 sterrorlen_s
PRIXLEAST64 sterrorlen_s
PRIXMAX strftime
PRIXPTR strftime
SCNdMAX strftime
SCNdPTR strftime
SCNiMAX strlen
SCNiPTR strncat
SCNipTR strncat_s
SCNoMAX strncmp
SCNoPTR strncpy
SCNuMAX strnlen
SCNuPTR strnlen_s
SCNxMAX strpbrk
SCNxPTR strpbrk
SIGABRT strrench
SIG_ATOMIC_MAX strspn
SIG_ATOMIC_MIN strstr
SIG_ATOMIC_WIDTH strtod
SIG_DFL strtof
SIG_ERR strtoimax
SIGFPE strtok
SIG_INGN strtok_s
SIGINT strto1
SIGSEGV strto1l
SIGTERM strto1l
_Static_assert statues
__STDC__ strto1ll
__STDC_ANALYZABLE__ struct
__STDC_HOSTED__ strxfrm
__STDC_IEC_559__ thrd busy
__STDC_IEC_559_COMPLEX__ thrd create
__STDC_IEC_60559_BFP__ thrd current
__STDC_IEC_60559_COMPLEX__ thrd detach
__STDC_ISO_10646__ thrd equal
__STDC_LIB_EXT1__ thrd error
__STDC_MB_MIGHT_NEQ_WC__ thrd exit
__STDC_NO_ATOMICS__ thrd join
__STDC_NO_COMPLEX__ thrd nomem

§ J.6.1 Portability issues 455
Portability issues § J.6.1
J.6.2 Particular identifiers or keywords

The following identifiers or keywords are not covered by the above and have particular semantics provided by this document.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Semantic Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td>cabsf</td>
</tr>
<tr>
<td>abort_handler_s</td>
<td>cabsl</td>
</tr>
<tr>
<td>abs</td>
<td>cacos</td>
</tr>
<tr>
<td>acos</td>
<td>cacosf</td>
</tr>
<tr>
<td>acosh</td>
<td>cacosh</td>
</tr>
<tr>
<td>acoshf</td>
<td>cacoshf</td>
</tr>
<tr>
<td>acoshl</td>
<td>cacosl</td>
</tr>
<tr>
<td>acosl</td>
<td>call_once</td>
</tr>
<tr>
<td>alignas</td>
<td>canonicalize</td>
</tr>
<tr>
<td>aligned_alloc</td>
<td>canonicalizef</td>
</tr>
<tr>
<td>alignof</td>
<td>at_quick_exit</td>
</tr>
<tr>
<td>and</td>
<td>carg</td>
</tr>
<tr>
<td>and_eq</td>
<td>cargf</td>
</tr>
<tr>
<td>asctime</td>
<td>case</td>
</tr>
<tr>
<td>asctime_s</td>
<td>casin</td>
</tr>
<tr>
<td>asinh</td>
<td>casinf</td>
</tr>
<tr>
<td>asinhf</td>
<td>casinh</td>
</tr>
<tr>
<td>asinhf</td>
<td>casinhf</td>
</tr>
<tr>
<td>atan</td>
<td>catan</td>
</tr>
<tr>
<td>atanh</td>
<td>catanh</td>
</tr>
<tr>
<td>atanhf</td>
<td>catanhf</td>
</tr>
<tr>
<td>atanhl</td>
<td>catanhl</td>
</tr>
<tr>
<td>atanl</td>
<td>catanl</td>
</tr>
<tr>
<td>atexit</td>
<td>calloc</td>
</tr>
<tr>
<td>atof</td>
<td>cabsf</td>
</tr>
<tr>
<td>atol</td>
<td>cabsl</td>
</tr>
<tr>
<td>atoll</td>
<td>cacos</td>
</tr>
<tr>
<td>atan</td>
<td>cacosf</td>
</tr>
<tr>
<td>atanl</td>
<td>cacosh</td>
</tr>
<tr>
<td>atanhf</td>
<td>cacoshf</td>
</tr>
<tr>
<td>canonicalize</td>
<td>cacosl</td>
</tr>
<tr>
<td>canonicalizef</td>
<td>calloc</td>
</tr>
<tr>
<td>carg</td>
<td>catanl</td>
</tr>
<tr>
<td>case</td>
<td>catanhl</td>
</tr>
<tr>
<td>casin</td>
<td>catanh</td>
</tr>
<tr>
<td>casinh</td>
<td>catanhf</td>
</tr>
<tr>
<td>casinhf</td>
<td>catanl</td>
</tr>
<tr>
<td>casinhf</td>
<td>catanhl</td>
</tr>
<tr>
<td>catan</td>
<td>cabsf</td>
</tr>
<tr>
<td>cabsf</td>
<td>cabsl</td>
</tr>
</tbody>
</table>

§ J.6.2 Portability issues 457
Portability issues § J.6.2
§ J.6.2 Portability issues 459
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTRDIFF_WIDTH</td>
<td>setpayloadl</td>
<td>tgammaf</td>
</tr>
<tr>
<td>putc</td>
<td>setpayloadsig</td>
<td>tgammal</td>
</tr>
<tr>
<td>putchar</td>
<td>setpayloadsigf</td>
<td>thousands_sep</td>
</tr>
<tr>
<td>puts</td>
<td>setpayloadsigl</td>
<td>thread_local</td>
</tr>
<tr>
<td>putwc</td>
<td>setvbuf</td>
<td>time</td>
</tr>
<tr>
<td>putwchar</td>
<td>short</td>
<td>timespec</td>
</tr>
<tr>
<td>qsort</td>
<td>SHRT_MAX</td>
<td>timespec_get</td>
</tr>
<tr>
<td>qsort_s</td>
<td>SHRT_MIN</td>
<td>time_t</td>
</tr>
<tr>
<td>quick_exit</td>
<td>SHRT_WIDTH</td>
<td>tm</td>
</tr>
<tr>
<td>raise</td>
<td>sig_atomic_t</td>
<td>tm_hour</td>
</tr>
<tr>
<td>rand</td>
<td>signal</td>
<td>tm_isdst</td>
</tr>
<tr>
<td>RAND_MAX</td>
<td>signed</td>
<td>tm_mday</td>
</tr>
<tr>
<td>realloc</td>
<td>sin</td>
<td>tm_mon</td>
</tr>
<tr>
<td>register</td>
<td>sinf</td>
<td>tmfile</td>
</tr>
<tr>
<td>remainder</td>
<td>sinh</td>
<td>tmfile_s</td>
</tr>
<tr>
<td>remainderf</td>
<td>sinhf</td>
<td>TMP_MAX</td>
</tr>
<tr>
<td>remainderl</td>
<td>sinh1</td>
<td>TMP_MAX_S</td>
</tr>
<tr>
<td>remove</td>
<td>sinl</td>
<td>tmpnam</td>
</tr>
<tr>
<td>remquo</td>
<td>sizeof</td>
<td>tmpnam_s</td>
</tr>
<tr>
<td>remquo</td>
<td>size_t</td>
<td>tm_sec</td>
</tr>
<tr>
<td>rename</td>
<td>SIZE_MAX</td>
<td>tm_wday</td>
</tr>
<tr>
<td>restrict</td>
<td>SIZE_WIDTH</td>
<td>tm_year</td>
</tr>
<tr>
<td>return</td>
<td>SNAN</td>
<td>true</td>
</tr>
<tr>
<td>rewind</td>
<td>SNANF</td>
<td>trunct</td>
</tr>
<tr>
<td>rint</td>
<td>SNANL</td>
<td>truncf</td>
</tr>
<tr>
<td>rintf</td>
<td>snprintf</td>
<td>trunccl</td>
</tr>
<tr>
<td>rintl</td>
<td>snprintf_s</td>
<td>TSS_DTOR_ITERATIONS</td>
</tr>
<tr>
<td>round</td>
<td>snprintf</td>
<td>TV_nsec</td>
</tr>
<tr>
<td>roundeven</td>
<td>snprintf_s</td>
<td>TV_sec</td>
</tr>
<tr>
<td>roundevenf</td>
<td>sqrt</td>
<td>typedef</td>
</tr>
<tr>
<td>roundf</td>
<td>sqrtf</td>
<td>UCHAR_MAX</td>
</tr>
<tr>
<td>roundl</td>
<td>sqrtl</td>
<td>UCHAR_WIDTH</td>
</tr>
</tbody>
</table>
| RSIZE_MAX | srand | ufromfp-
| rsize_t | sscanf | ufromfpf-
| scalbln | scanf | ufromfpl-
| scalblnf | static | ufromfpx-
| scalblnl | static_assert | ufromfpzf-
| scalbn | STDCL | ufromfpzl-
| scalbnf | stderr | ULLONG_MAX |
| scalbnl | stdin | ULLONG_WIDTH |
| scanf | stdout | ULONG_MAX |
| scanf_s | switch | ULONG_WIDTH |
| SCHAR_MAX | swprintf | undef |
| SCHAR_MIN | swprintf_s | undef |
| SCHAR_WIDTH | sscanf | ungetc |
| SEEK_CUR | swscanf | ungetwc |
| SEEK_END | system | union |
| SEEK_SET | tan | unsigned |
| setbuf | tanf | USHRT_MAX |
| set_constraint_handler_s | tanh | USHRT_WIDTH |
| setjmp | tanhf | va_arg |
| setlocale | tanhl | va_copy |
| setpayload | tanl | va_end |
| setpayloadf | va_list | va_start |
Portability issues § J.6.2

vfprintf
vfprintf_s
vfscanf
vfscanf_s
vfwprintf
vfwprintf_s
vfswscanf
vfswscanf_s
void volatile
vprintf
vprintf_s
vscanf
vscanf_s
vsnprintf
vsnprintf_s
vswprintf
vswprintf_s
vswscanf
vswscanf_s
WCHAR_MAX
WCHAR_MIN
WCHAR_WIDTH
wcrtomb
wcrtomb_s
wctob
wctomb
wctrans
wctrans_t
wctype
wctype_t

WEOF
while
WINT_MAX
WINT_MIN
wint_t
WINT_WIDTH
wmemchr
wmemcmp
wmemcpy
wmemcpy_s
wmemmove
wmemmove_s
wmemset
wprintf
wprintf_s
wscanf
wscanf_s
xor
xor_eq