Summary
10/08/2022 13:35:26

Differences exist between documents.

New Document:	**Old Document:**
Pages from n3047_C2X-draft	Pages from n2478_C3-Feb2020-draft
5 pages (309 KB)	4 pages (221 KB)
10/08/2022 13:35:18	10/08/2022 13:35:18
Used to display results.	

Get started: first change is on page 1.

No pages were deleted

How to read this report

Highlight indicates a change.
~~**Deleted**~~ indicates deleted content.
indicates pages were changed.
indicates pages were moved.

29 **EXAMPLE 2** The following describes floating-point representations that also meet the requirements for single-precision and double-precision numbers in IEC 60559,[29] and the appropriate values in a <float.h> header for types **float** and **double**:

$$x_f = s2^e \sum_{k=1}^{24} f_k 2^{-k}, \quad -125 \le e \le +128$$

$$x_d = s2^e \sum_{k=1}^{53} f_k 2^{-k}, \quad -1021 \le e \le +1024$$

```
FLT_IS_IEC_60559                2
FLT_RADIX                       2
FLT_MANT_DIG                   24
FLT_EPSILON        1.19209290E-07F // decimal constant
FLT_EPSILON               0X1P-23F // hex constant
FLT_DECIMAL_DIG                 9
FLT_DIG                         6
FLT_MIN_EXP                  -125
FLT_MIN            1.17549435E-38F // decimal constant
FLT_MIN                  0X1P-126F // hex constant
FLT_TRUE_MIN       1.40129846E-45F // decimal constant
FLT_TRUE_MIN             0X1P-149F // hex constant
FLT_HAS_SUBNORM                 1
FLT_MIN_10_EXP                -37
FLT_MAX_EXP                  +128
FLT_MAX            3.40282347E+38F // decimal constant
FLT_MAX            0X1.fffffeP127F // hex constant
FLT_MAX_10_EXP                +38
DBL_MANT_DIG                   53
DBL_IS_IEC_60559                2
DBL_EPSILON 2.2204460492503131E-16 // decimal constant
DBL_EPSILON                0X1P-52 // hex constant
DBL_DECIMAL_DIG                17
DBL_DIG                        15
DBL_MIN_EXP                 -1021
DBL_MIN   2.2250738585072014E-308 // decimal constant
DBL_MIN                 0X1P-1022 // hex constant
DBL_TRUE_MIN 4.9406564584124654E-324 // decimal constant
DBL_TRUE_MIN            0X1P-1074 // hex constant
DBL_HAS_SUBNORM                 1
DBL_MIN_10_EXP               -307
DBL_MAX_EXP                 +1024
DBL_MAX   1.7976931348623157E+308 // decimal constant
DBL_MAX     0X1.fffffffffffffP1023 // hex constant
DBL_MAX_10_EXP               +308
```

Forward references: conditional inclusion (6.10.1), predefined macro names (6.10.9), complex arithmetic <complex.h> (7.3), extended multibyte and wide character utilities <wchar.h> (7.31), floating-point environment <fenv.h> (7.6), general utilities <stdlib.h> (7.24), input/output <stdio.h> (7.23), mathematics <math.h> (7.12), IEC 60559 floating-point arithmetic (Annex F), IEC 60559-compatible complex arithmetic (Annex G).

5.2.4.2.3 Characteristics of decimal floating types in <float.h>

1 This subclause specifies macros in <float.h> that provide characteristics of decimal floating types (an optional feature) in terms of the model presented in 5.2.4.2.2. An implementation that does not support decimal floating types shall not provide these macros. The prefixes **DEC32_**, **DEC64_**, and **DEC128_** denote the types **_Decimal32**, **_Decimal64**, and **_Decimal128** respectively.

2 **DEC_EVAL_METHOD** is the decimal floating-point analog of **FLT_EVAL_METHOD** (5.2.4.2.2). Its implementation-defined value characterizes the use of evaluation formats for decimal floating

[29] The floating-point model in that standard sums powers of b from zero, so the values of the exponent limits are one less than shown here.

types:

−1 indeterminable;

0 evaluate all operations and constants just to the range and precision of the type;

1 evaluate operations and constants of type `_Decimal32` and `_Decimal64` to the range and precision of the `_Decimal64` type, evaluate `_Decimal128` operations and constants to the range and precision of the `_Decimal128` type;

2 evaluate all operations and constants to the range and precision of the `_Decimal128` type.

3 Each of the decimal signaling NaN macros

```
DEC32_SNAN
DEC64_SNAN
DEC128_SNAN
```

expands to a constant expression of the respective decimal floating type representing a signaling NaN. If an optional unary + or - operator followed by a signaling NaN macro is used for initializing an object of the same type that has static or thread storage duration, the object is initialized with a signaling NaN value.

4 The macro

```
DEC_INFINITY
```

expands to a constant expression of type `_Decimal32` representing positive infinity.

5 The macro

```
DEC_NAN
```

expands to a constant expression of type `_Decimal32` representing a quiet NaN.

6 The integer values given in the following lists shall be replaced by constant expressions suitable for use in `#if` preprocessing directives:

— radix of exponent representation, $b(=10)$

For the standard floating types, this value is implementation-defined and is specified by the macro `FLT_RADIX`. For the decimal floating types there is no corresponding macro, since the value 10 is an inherent property of the types. Wherever `FLT_RADIX` appears in a description of a function that has versions that operate on decimal floating types, it is noted that for the decimal floating-point versions the value used is implicitly 10, rather than `FLT_RADIX`.

— number of digits in the coefficient

```
DEC32_MANT_DIG      7
DEC64_MANT_DIG      16
DEC128_MANT_DIG     34
```

— minimum exponent

```
DEC32_MIN_EXP       -94
DEC64_MIN_EXP       -382
DEC128_MIN_EXP      -6142
```

— maximum exponent

```
DEC32_MAX_EXP      97
DEC64_MAX_EXP      385
DEC128_MAX_EXP     6145
```

— maximum representable finite decimal floating-point number (there are 6, 15 and 33 9's after the decimal points respectively)

```
DEC32_MAX          9.999999E96DF
DEC64_MAX          9.999999999999999E384DD
DEC128_MAX         9.999999999999999999999999999999999E6144DL
```

— the difference between 1 and the least value greater than 1 that is representable in the given floating type

```
DEC32_EPSILON      1E-6DF
DEC64_EPSILON      1E-15DD
DEC128_EPSILON     1E-33DL
```

— minimum normalized positive decimal floating-point number

```
DEC32_MIN          1E-95DF
DEC64_MIN          1E-383DD
DEC128_MIN         1E-6143DL
```

— minimum positive subnormal decimal floating-point number

```
DEC32_TRUE_MIN     0.000001E-95DF
DEC64_TRUE_MIN     0.000000000000001E-383DD
DEC128_TRUE_MIN    0.000000000000000000000000000000001E-6143DL
```

7 For decimal floating-point arithmetic, it is often convenient to consider an alternate equivalent model where the significand is represented with integer rather than fraction digits. With s, b, e, p, and f_k as defined in 5.2.4.2.2, a floating-point number x is defined by the model:

$$x = s \cdot b^{(e-p)} \sum_{k=1}^{p} f_k \cdot b^{(p-k)}$$

8 With b fixed to 10, a decimal floating-point number x is thus:

$$x = s \cdot 10^{(e-p)} \sum_{k=1}^{p} f_k \cdot 10^{(p-k)}$$

The *quantum exponent* is $q = e - p$ and the *coefficient* is $c = f_1 f_2 \cdots f_p$, which is an integer between 0 and $10^{(p-1)}$, inclusive. Thus, $x = s \cdot c \cdot 10^q$ is represented by the triple of integers (s, c, q). The *quantum* of x is 10^q, which is the value of a unit in the last place of the coefficient.

Quantum exponent ranges

Type	`_Decimal32`	`_Decimal64`	`_Decimal128`
Maximum Quantum Exponent (q_{max})	90	369	6111
Minimum Quantum Exponent (q_{min})	-101	-398	-6176

9 For binary floating-point arithmetic following IEC 60559, representations in the model described in 5.2.4.2.2 that have the same numerical value are indistinguishable in the arithmetic. However, for

decimal floating-point arithmetic, representations that have the same numerical value but different quantum exponents, e.g., $(+1, 10, -1)$ representing 1.0 and $(+1, 100, -2)$ representing 1.00, are distinguishable. To facilitate exact fixed-point calculation, operation results that are of decimal floating type have a *preferred quantum exponent*, as specified in IEC 60559, which is determined by the quantum exponents of the operands if they have decimal floating types (or by specific rules for conversions from other types). The table below gives rules for determining preferred quantum exponents for results of IEC 60559 operations, and for other operations specified in this document. When exact, these operations produce a result with their preferred quantum exponent, or as close to it as possible within the limitations of the type. When inexact, these operations produce a result with the least possible quantum exponent. For example, the preferred quantum exponent for addition is the minimum of the quantum exponents of the operands. Hence $(+1, 123, -2) + (+1, 4000, -3) = (+1, 5230, -3)$ or $1.23 + 4.000 = 5.230$.

10 The following table shows, for each operation delivering a result in decimal floating-point format, how the preferred quantum exponents of the operands, $Q(\mathtt{x})$, $Q(\mathtt{y})$, etc., determine the preferred quantum exponent of the operation result, provided the table formula is defined for the arguments. For the cases where the formula is undefined and the function result is $\pm\infty$, the preferred quantum exponent is immaterial because the quantum exponent of $\pm\infty$ is defined to be infinity. For the other cases where the formula is undefined and the function result is finite, the preferred quantum exponent is unspecified[30].

Preferred quantum exponents

Operation	Preferred quantum exponent of result
roundeven, **round**, **trunc**, **ceil**, **floor**, **rint**, **nearbyint**	$\max(Q(\mathtt{x}), 0)$
nextup, **nextdown**, **nextafter**, **nexttoward**	least possible
remainder	$\min(Q(\mathtt{x}), Q(\mathtt{y}))$
fmin, **fmax**, **fminimum**, **fmaximum**, **fminimum_mag**, **fmaximum_mag**, **fminimum_num**, **fmaximum_num**, **fminimum_mag_num**, **fmaximum_mag_num**	$Q(\mathtt{x})$ if x gives the result, $Q(\mathtt{y})$ if y gives the result
scalbn, **scalbln**	$Q(\mathtt{x}) + \mathtt{n}$
ldexp	$Q(\mathtt{x}) + \mathtt{p}$
logb	0
+, **d32add**, **d64add**	$\min(Q(\mathtt{x}), Q(\mathtt{y}))$
-, **d32sub**, **d64sub**	$\min(Q(\mathtt{x}), Q(\mathtt{y}))$
*, **d32mul**, **d64mul**	$Q(\mathtt{x}) + Q(\mathtt{y})$
/, **d32div**, **d64div**	$Q(\mathtt{x}) - Q(\mathtt{y})$
sqrt, **d32sqrt**, **d64sqrt**	$\lfloor Q(\mathtt{x})/2 \rfloor$
fma, **d32fma**, **d64fma**	$\min(Q(\mathtt{x}) + Q(\mathtt{y}), Q(\mathtt{z}))$
conversion from integer type	0
exact conversion from non-decimal floating type	0
inexact conversion from non-decimal floating type	least possible
conversion between decimal floating types	$Q(\mathtt{x})$
*cx returned by **canonicalize**	$Q(*\mathtt{x})$
strto, **wcsto**, **scanf**, floating constants of decimal floating type	see 7.24.1.6
-(x) ,+(x)	$Q(\mathtt{x})$
fabs	$Q(\mathtt{x})$
copysign	$Q(\mathtt{x})$
quantize	$Q(\mathtt{y})$

[30] Although unspecified in IEC 60559, a preferred quantum exponent of 0 for these cases would be a reasonable implementation choice.

quantum	$Q(\texttt{x})$
*`encptr` returned by **encodedec**, **encodebin**	$Q(*\texttt{xptr})$
*`xptr` returned by **decodedec**, **decodebin**	$Q(*\texttt{encptr})$
fmod	$\min(Q(\texttt{x}), Q(\texttt{y}))$
fdim	$\min((Q(\texttt{x}), Q(\texttt{y}))$ if $\texttt{x} > \texttt{y}$, 0 if $\texttt{x} \leq \texttt{y}$
cbrt	$\lfloor Q(\texttt{x})/3 \rfloor$
hypot	$\min(Q(\texttt{x}), Q(\texttt{y}))$
pow	$\lfloor \texttt{y} \times Q(\texttt{x}) \rfloor$
modf	$Q(\texttt{value})$
*`iptr` returned by **modf**	$\max(Q(\texttt{value}), 0)$
frexp	$Q(\texttt{value})$ if $\texttt{value} = 0$, –(length of coefficient of value) otherwise
*`res` returned by **setpayload**, **setpayloadsig**	0 if `pl` does not represent a valid payload, not applicable otherwise (NaN returned)
getpayload	0 if *`x` is a NaN, unspecified otherwise
compoundn	$\lfloor n \times \min(0, Q(x)) \rfloor$
pown	$\lfloor n \times Q(x) \rfloor$
powr	$\lfloor y \times Q(x) \rfloor$
rootn	$\lfloor Q(x)/n \rfloor$
rsqrt	$-\lfloor Q(x)/2 \rfloor$
transcendental functions	0

A function family listed in the table above indicates the functions for all decimal floating types, where the function family is represented by the name of the functions without a suffix. For example, **ceil** indicates the functions **ceild32**, **ceild64**, and **ceild128**.

Forward references: extended multibyte and wide character utilities `<wchar.h>` (7.31), floating-point environment `<fenv.h>` (7.6), general utilities `<stdlib.h>` (7.24), input/output `<stdio.h>` (7.23), mathematics `<math.h>` (7.12), type-generic mathematics `<tgmath.h>` (7.27), IEC 60559 floating-point arithmetic (Annex F).