`ULONG_MAX` `4294967295 //` $2^{32}-1$

— minimum value for an object of type **`long long int`**

`LLONG_MIN` `-9223372036854775807 //` $-(2^{63}-1)$

— maximum value for an object of type **`long long int`**

`LLONG_MAX` `+9223372036854775807 //` $2^{63}-1$

— maximum value for an object of type **`unsigned long long int`**

`ULLONG_MAX` `18446744073709551615 //` $2^{64}-1$

2 If an object of type **`char`** can hold negative values, the value of **`CHAR_MIN`** shall be the same as that of **`SCHAR_MIN`** and the value of **`CHAR_MAX`** shall be the same as that of **`SCHAR_MAX`**. Otherwise, the value of **`CHAR_MIN`** shall be 0 and the value of **`CHAR_MAX`** shall be the same as that of **`UCHAR_MAX`**.[20] The value **`UCHAR_MAX`** shall equal $2^{\texttt{CHAR\_BIT}}-1$.

Forward references: representations of types (6.2.6), conditional inclusion (6.10.1).

5.2.4.2.2 Characteristics of floating types `<float.h>`

1 The characteristics of floating types are defined in terms of a model that describes a representation of floating-point numbers and values that provide information about an implementation's floating-point arithmetic.[21] The following parameters are used to define the model for each floating-point type:

s	sign (± 1)
b	base or radix of exponent representation (an integer > 1)
e	exponent (an integer between a minimum $e_{\min}$ and a maximum $e_{\max}$)
p	precision (the number of base-b digits in the significand)
f_k	nonnegative integers less than b (the significand digits)

2 A *floating-point number* (x) is defined by the following model:

$$x = sb^e \sum_{k=1}^{p} f_k b^{-k}, \quad e_{\min} \le e \le e_{\max}$$

3 In addition to normalized floating-point numbers ($f_1 > 0$ if $x \neq 0$), floating types may be able to contain other kinds of floating-point numbers, such as *subnormal floating-point numbers* ($x \neq 0$, $e = e_{\min}$, $f_1 = 0$) and *unnormalized floating-point numbers* ($x \neq 0$, $e > e_{\min}$, $f_1 = 0$), and values that are not floating-point numbers, such as infinities and NaNs. A *NaN* is an encoding signifying Not-a-Number. A *quiet NaN* propagates through almost every arithmetic operation without raising a floating-point exception; a *signaling NaN* generally raises a floating-point exception when occurring as an arithmetic operand.[22]

4 An implementation may give zero and values that are not floating-point numbers (such as infinities and NaNs) a sign or may leave them unsigned. Wherever such values are unsigned, any requirement in this International Standard to retrieve the sign shall produce an unspecified sign, and any requirement to set the sign shall be ignored.

5 The minimum range of representable values for a floating type is the most negative finite floating-point number representable in that type through the most positive finite floating-point number representable in that type. In addition, if negative infinity is representable in a type, the range of

[20] See 6.2.5.

[21] The floating-point model is intended to clarify the description of each floating-point characteristic and does not require the floating-point arithmetic of the implementation to be identical.

[22] IEC 60559:1989 specifies quiet and signaling NaNs. For implementations that do not support IEC 60559:1989, the terms quiet NaN and signaling NaN are intended to apply to encodings with similar behavior.

that type is extended to all negative real numbers; likewise, if positive infinity is representable in a type, the range of that type is extended to all positive real numbers.

6 The accuracy of the floating-point operations (+,-,*,/) and of the library functions in <math.h> and <complex.h> that return floating-point results is implementation-defined, as is the accuracy of the conversion between floating-point internal representations and string representations performed by the library functions in <stdio.h>, <stdlib.h>, and <wchar.h>. The implementation may state that the accuracy is unknown.

7 All integer values in the <float.h> header, except **FLT_ROUNDS**, shall be constant expressions suitable for use in **#if** preprocessing directives; all floating values shall be constant expressions. All except **DECIMAL_DIG**, **FLT_EVAL_METHOD**, **FLT_RADIX**, and **FLT_ROUNDS** have separate names for all three floating-point types. The floating-point model representation is provided for all values except **FLT_EVAL_METHOD** and **FLT_ROUNDS**.

8 The rounding mode for floating-point addition is characterized by the implementation-defined value of **FLT_ROUNDS**:[23)]

-1 indeterminable

0 toward zero

1 to nearest

2 toward positive infinity

3 toward negative infinity

All other values for **FLT_ROUNDS** characterize implementation-defined rounding behavior.

9 Except for assignment and cast (which remove all extra range and precision), the values yielded by operators with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type. The use of evaluation formats is characterized by the implementation-defined value of **FLT_EVAL_METHOD**:[24)]

-1 indeterminable;

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

1 evaluate operations and constants of type **float** and **double** to the range and precision of the **double** type, evaluate **long double** operations and constants to the range and precision of the **long double** type;

2 evaluate all operations and constants to the range and precision of the **long double** type.

All other negative values for **FLT_EVAL_METHOD** characterize implementation-defined behavior.

10 The presence or absence of subnormal numbers is characterized by the implementation-defined values of **FLT_HAS_SUBNORM**, **DBL_HAS_SUBNORM**, and **LDBL_HAS_SUBNORM**:

-1 indeterminable[25)]

0 absent (type does not support subnormal numbers)[26)]

[23)]Evaluation of **FLT_ROUNDS** correctly reflects any execution-time change of rounding mode through the function **fesetround** in <fenv.h>.

[24)]The evaluation method determines evaluation formats of expressions involving all floating types, not just real types. For example, if **FLT_EVAL_METHOD** is 1, then the product of two **float _Complex** operands is represented in the **double _Complex** format, and its parts are evaluated to **double**.

[25)]Characterization as indeterminable is intended if floating-point operations do not consistently interpret subnormal representations as zero, nor as nonzero.

[26)]Characterization as absent is intended if no floating-point operations produce subnormal results from non-subnormal inputs, even if the type format includes representations of subnormal numbers.

1 present (type does support subnormal numbers)

11 The values given in the following list shall be replaced by constant expressions with implementation-defined values that are greater or equal in magnitude (absolute value) to those shown, with the same sign:

— radix of exponent representation, b

```
FLT_RADIX                         2
```

— number of base-**FLT_RADIX** digits in the floating-point significand, p

```
FLT_MANT_DIG
DBL_MANT_DIG
LDBL_MANT_DIG
```

— number of decimal digits, n, such that any floating-point number with p radix b digits can be rounded to a floating-point number with n decimal digits and back again without change to the value,

$$\begin{cases} p\log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lceil 1 + p\log_{10} b \rceil & \text{otherwise} \end{cases}$$

```
FLT_DECIMAL_DIG                    6
DBL_DECIMAL_DIG                   10
LDBL_DECIMAL_DIG                  10
```

— number of decimal digits, n, such that any floating-point number in the widest supported floating type with $p_{\max}$ radix b digits can be rounded to a floating-point number with n decimal digits and back again without change to the value,

$$\begin{cases} p_{\max}\log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lceil 1 + p_{\max}\log_{10} b \rceil & \text{otherwise} \end{cases}$$

```
DECIMAL_DIG                      10
```

— number of decimal digits, q, such that any floating-point number with q decimal digits can be rounded into a floating-point number with p radix b digits and back again without change to the q decimal digits,

$$\begin{cases} p\log_{10} b & \text{if } b \text{ is a power of } 10 \\ \lfloor (p-1)\log_{10} b \rfloor & \text{otherwise} \end{cases}$$

```
FLT_DIG                           6
DBL_DIG                          10
LDBL_DIG                         10
```

— minimum negative integer such that **FLT_RADIX** raised to one less than that power is a normalized floating-point number, $e_{\min}$

```
FLT_MIN_EXP
DBL_MIN_EXP
LDBL_MIN_EXP
```

— minimum negative integer such that 10 raised to that power is in the range of normalized floating-point numbers, $\lceil log_{10} b^{e_{\min}-1} \rceil$

```
FLT_MIN_10_EXP                   -37
DBL_MIN_10_EXP                   -37
LDBL_MIN_10_EXP                  -37
```

— maximum integer such that `FLT_RADIX` raised to one less than that power is a representable finite floating-point number, $e_{\max}$

```
FLT_MAX_EXP
DBL_MAX_EXP
LDBL_MAX_EXP
```

— maximum integer such that 10 raised to that power is in the range of representable finite floating-point numbers, $\lfloor log_{10}((1 - b^{-p})b^{e_{\max}}) \rfloor$

```
FLT_MAX_10_EXP                   +37
DBL_MAX_10_EXP                   +37
LDBL_MAX_10_EXP                  +37
```

12 The values given in the following list shall be replaced by constant expressions with implementation-defined values that are greater than or equal to those shown:

— maximum representable finite floating-point number, $(1 - b^{-p})b^{e_{\max}}$

```
FLT_MAX                        1E+37
DBL_MAX                        1E+37
LDBL_MAX                       1E+37
```

13 The values given in the following list shall be replaced by constant expressions with implementation-defined (positive) values that are less than or equal to those shown:

— the difference between 1 and the least value greater than 1 that is representable in the given floating point type, b^{1-p}

```
FLT_EPSILON                     1E-5
DBL_EPSILON                     1E-9
LDBL_EPSILON                    1E-9
```

— minimum normalized positive floating-point number, $b^{e_{\min}-1}$

```
FLT_MIN                        1E-37
DBL_MIN                        1E-37
LDBL_MIN                       1E-37
```

— minimum positive floating-point number[27)]

```
FLT_TRUE_MIN                   1E-37
DBL_TRUE_MIN                   1E-37
LDBL_TRUE_MIN                  1E-37
```

[27)] If the presence or absence of subnormal numbers is indeterminable, then the value is intended to be a positive number no greater than the minimum normalized positive number for the type.

Recommended practice

14 Conversion from (at least) **double** to decimal with **DECIMAL_DIG** digits and back should be the identity function.

15 **EXAMPLE 1** The following describes an artificial floating-point representation that meets the minimum requirements of this International Standard, and the appropriate values in a `<float.h>` header for type **float**:

$$x = s16^e \sum_{k=1}^{6} f_k 16^{-k}, \quad -31 \le e \le +32$$

```
FLT_RADIX                      16
FLT_MANT_DIG                    6
FLT_EPSILON       9.53674316E-07F
FLT_DECIMAL_DIG                 9
FLT_DIG                         6
FLT_MIN_EXP                   -31
FLT_MIN           2.93873588E-39F
FLT_MIN_10_EXP                -38
FLT_MAX_EXP                   +32
FLT_MAX           3.40282347E+38F
FLT_MAX_10_EXP                +38
```

16 **EXAMPLE 2** The following describes floating-point representations that also meet the requirements for single-precision and double-precision numbers in IEC 60559,[28] 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_RADIX                       2
DECIMAL_DIG                    17
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_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
```

[28]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.

```
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
```

If a type wider than `double` were supported, then `DECIMAL_DIG` would be greater than 17. For example, if the widest type were to use the minimal-width IEC 60559 double-extended format (64 bits of precision), then `DECIMAL_DIG` would be 21.

Forward references: conditional inclusion (6.10.1), complex arithmetic `<complex.h>` (7.3), extended multibyte and wide character utilities `<wchar.h>` (7.29), floating-point environment `<fenv.h>` (7.6), general utilities `<stdlib.h>` (7.22), input/output `<stdio.h>` (7.21), mathematics `<math.h>` (7.12).