— 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)[19)]

— 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 **#include**d 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 struct-declaration-list

5.2.4.2 Numerical limits

1 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>

1 The values given below shall be replaced by constant expressions suitable for use in **#if** preprocessing directives.

Moreover, except for **CHAR_BIT** and **MB_LEN_MAX**, 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)

```
CHAR_BIT                    8
```

— minimum value for an object of type **signed char**

```
SCHAR_MIN                 -127 // −(2^7 − 1)
```

[19)] See "future language directions" (6.11.3).

— maximum value for an object of type **signed char**

SCHAR_MAX	+127 // $2^7 - 1$

— maximum value for an object of type **unsigned char**

UCHAR_MAX	255 // $2^8 - 1$

— minimum value for an object of type **char**

CHAR_MIN	*see below*

— maximum value for an object of type **char**

CHAR_MAX	*see below*

— maximum number of bytes in a multibyte character, for any supported locale

MB_LEN_MAX	1

— minimum value for an object of type **short int**

SHRT_MIN	-32767 // $-(2^{15} - 1)$

— maximum value for an object of type **short int**

SHRT_MAX	+32767 // $2^{15} - 1$

— maximum value for an object of type **unsigned short int**

USHRT_MAX	65535 // $2^{16} - 1$

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

INT_MIN	-32767 // $-(2^{15} - 1)$

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

INT_MAX	+32767 // $2^{15} - 1$

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

UINT_MAX	65535 // $2^{16} - 1$

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

LONG_MIN	-2147483647 // $-(2^{31} - 1)$

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

LONG_MAX	+2147483647 // $2^{31} - 1$

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

```
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.