Two's complement sign representation for C2x

Jean-François Bastien, Jens Gustedt

ISO/IEC JTC 1/SC 22/WG14 **N2412**

v 2

August 11, 2019

Two's complement sign representation for C2x

Modification request for C2x

JF Bastien and Jens Gustedt
Apple Inc., USA, and INRIA and ICube, Université de Strasbourg, France

We implement the agreed change to abandon ones' complement and sign-and-magnitude representation from C.

This is a follow-up to document **N2218**[1] which found positive WG14 support to make two's complement the only sign representation for the next C standard, and a follow-up to document **N2330**[2] which only generated partial consensus in the London 2019 meeting of WG14.

1. INTRODUCTION

Removing other sign representations than two's complement from C allows to simplify the specification of integer types substantially. As has been voted in the London 2019 meeting we only implement the essential changes to the sign representation in this paper, namely

— Remove the specifications of the other sign representations.
— Impose the value of the minimal value of a signed type to be -2^{N-1} where N is the width of the type.
— Derive the values of the `_MIN` and `_MAX` macros for all integer types from the corresponding `_WIDTH` macro.
— Clean-up the remaining parts of the standard from all obsolete mentions of two's complement and negative integer zeros.

WG21 has recently adapted the changes promoted in their document **p1236**[3]. Generally, C++ goes much beyond what is presented here:

— Bit-fields can have excess bits.
— Overflowing operations and out-of-range conversion are generally mapped to modulo operations and cannot trap or raise signals.
— Enumeration types and their underlying compatible integer types have precise definitions.

WG14 has not yet found consensus for these points, so we leave them as they are in the current specification.

2. TWO'S COMPLEMENT

Restricting the possible sign representations to two's complement is relatively straight forward and does not need much of deep thinking.
There are some other direct fallouts from doing this, such as other mentions of two's complement in the document that now become obsolete. This concerns in particular the definition of the exact width integer types, and of the (bogus) specifications of arithmetic on atomic types.

3. TIGHTENING OF INTEGER REPRESENTATIONS

3.1. Minimum values of signed integer types

Even for two's complement representation C17 allowed that the value with sign bit 1 and all other bits 0 might be a trap representation. We change this and are thereby in line with

[1] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2218.htm
[2] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2330.pdf
[3] http://wg21.link/p1236

the changes in C++. We force that for integer types with a width of N the minimum value is forced to -2^{N-1} (and the maximum value remains at $2^{N-1}-1$).

3.2. Adjust widths of signed and unsigned integer types

In C17, the widths of corresponding signed and unsigned may differ by one, in particular an unsigned may be realized by just masking out the sign bit of the signed type. This possibility does not seem to be used in the field, complicates arguing about integers and adds potential case analysis to programs.

4. WIDTH, MINIMUM AND MAXIMUM MACROS FOR INTEGER TYPES

With these aggreed changes the relationship between the unsigned maximum and signed minimum and maximum values now becomes much simpler and can easily be expressed through the widths of the types, namely if the width is N these values are now fixed to 2^N-1, -2^{N-1} and $2^{N-1}-1$, respectively. Since the integration of the floating point TS's already brings in macros that specify the width of the standard integer types, we simplify the presentation such that it is centered around the width.

This has the advantage that all requirements for the minimum width of integer types can now be presented as requirements of **_WIDTH** macros, and the specification of the **_MIN** and **_MAX** can be generic.

5. PROPOSED TEXT

As usual, we provide a diff-marked set of changed pages in an appendix. Unfortunately, for the central parts of the proposed changes the diff-marked text is not very readable so we provide the whole text for 5.2.4.2.1, 6.2.6.2, 7.20p5, 7.20.2, and 7.20.3, and only the changes that are textually small are reported in the diffmark appendix.

5.2.4.2.1 Characteristics of integer types <limits.h>

1 The values given below shall be replaced by constant expressions suitable for use in **#if** preprocessing directives. Their implementation-defined values shall be equal or greater to those shown.

— width for an object of type **_Bool**

```
BOOL_WIDTH                    1
```

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

```
CHAR_BIT                      8
```

The macros **CHAR_WIDTH**, **SCHAR_WIDTH**, and **UCHAR_WIDTH** that represent the width of the types **char**, **signed char** and **unsigned char** shall expand to the same value as **CHAR_BIT**.

— width for an object of type **unsigned short int**

```
USHRT_WIDTH                  16
```

The macro **SHRT_WIDTH** represents the width of the type **short int** and shall expand to the same value as **USHRT_WIDTH**.

— width for an object of type **unsigned int**

```
UINT_WIDTH                   16
```

The macro **INT_WIDTH** represents the width of the type **int** and shall expand to the same value as **UINT_WIDTH**.

— width for an object of type **unsigned long int**

```
ULONG_WIDTH                    32
```

The macro **LONG_WIDTH** represents the width of the type **long int** and shall expand to the same value as **ULONG_WIDTH**.

— width for an object of type **unsigned long long int**

```
ULLONG_WIDTH                   64
```

The macro **LLONG_WIDTH** represents the width of the type **long long int** and shall expand to the same value as **ULLONG_WIDTH**.

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

```
MB_LEN_MAX                     1
```

2 For all unsigned integer types for which <limits.h> or <stdint.h> define a macro with suffix **_WIDTH** holding its width N, there is a macro with suffix **_MAX** holding the maximal value $2^N - 1$ that is representable by the type, that is suitable for use in **#if** preprocessing directives and that has the same type as would an expression that is an object of the corresponding type converted according to the integer promotions.

3 For all signed integer types for which <limits.h> or <stdint.h> define a macro with suffix **_WIDTH** holding its width N, there are macros with suffix **_MIN** and **_MAX** holding the minimal and maximal values -2^{N-1} and $2^{N-1} - 1$ that are representable by the type, that are suitable for use in **#if** preprocessing directives and that have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions.

4 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**.[4]

...

6.2.6.2 Integer types

1 For unsigned integer types the bits of the object representation shall be divided into two groups: value bits and padding bits. If there are N value bits, each bit shall represent a different power of 2 between 1 and 2^{N-1}, so that objects of that type shall be capable of representing values from 0 to $2^N - 1$ using a pure binary representation; this shall be known as the value representation. The values of any padding bits are unspecified.The number of value bits N is called the *width* of the unsigned integer type. There need not be any padding bits; **unsigned char** shall not have any padding bits.

2 For signed integer types, the bits of the object representation shall be divided into three groups: value bits, padding bits, and the sign bit. If the corresponding unsigned type has width N, the signed type uses the same number of N bits, its *width*, as value bits and sign bit. $N-1$ are value bits and the remaining bit is the sign bit. Each bit that is a value bit shall have the same value as the same bit in the object representation of the corresponding unsigned type. If the sign bit

[4] See 6.2.5.

is zero, it shall not affect the resulting value. If the sign bit is one, it has value $-(2^{N-1})$. There need not be any padding bits; **signed char** shall not have any padding bits.

3 The values of any padding bits are unspecified. A valid (non-trap) object representation of a signed integer type where the sign bit is zero is a valid object representation of the corresponding unsigned type, and shall represent the same value. For any integer type, the object representation where all the bits are zero shall be a representation of the value zero in that type.

4 The *precision* of an integer type is the number of value bits.

NOTE 1. *Some combinations of padding bits might generate trap representations, for example, if one padding bit is a parity bit. Regardless, no arithmetic operation on valid values can generate a trap representation other than as part of an exceptional condition such as an overflow, and this cannot occur with unsigned types. All other combinations of padding bits are alternative object representations of the value specified by the value bits.*

NOTE 2. *The sign representation defined in this document is called* two's complement*. Previous revisions of this document additionally allowed other sign representations.*

NOTE 3. *For unsigned integer types the width and precision are the same, while for signed integer types the width is one greater than the precision.*

...

7.20 Integer types <stdint.h>

...

5 For all integer types for which there is a macro with suffix **_WIDTH** holding the width, maximum macros with suffix **_MAX** and, for all signed types, minimum macros with suffix **_MIN** are defined as by 5.2.4.2.

...

7.20.2 Widths of specified-width integer types

1 The following object-like macros specify the width of the types declared in <stdint.h>. Each macro name corresponds to a similar type name in 7.20.1.

2 Each instance of any defined macro shall be replaced by a constant expression suitable for use in **#if** preprocessing directives. Its implementation-defined value shall be equal to or greater than the value given below, except where stated to be exactly the given value. An implementation shall define only the macros corresponding to those typedef names it actually provides.[5]

7.20.2.1 Width of exact-width integer types

1

INTN_WIDTH	exactly N
UINTN_WIDTH	exactly N

7.20.2.2 Width of minimum-width integer types

[5] The exact-width and pointer-holding integer types are optional.

`INT_LEASTN_WIDTH`	exactly `UINT_LEASTN_WIDTH`
`UINT_LEASTN_WIDTH`	N

7.20.2.3 Width of fastest minimum-width integer types

1

`INT_FASTN_WIDTH`	exactly `UINT_FASTN_WIDTH`
`UINT_FASTN_WIDTH`	N

7.20.2.4 Width of integer types capable of holding object pointers

1

`INTPTR_WIDTH`	exactly `UINTPTR_WIDTH`
`UINTPTR_WIDTH`	16

7.20.2.5 Width of greatest-width integer types

1

`INTMAX_WIDTH`	exactly `UINTMAX_WIDTH`
`UINTMAX_WIDTH`	64

7.20.3 Characteristics of other integer types

1 The following object-like macros specify the width of integer types corresponding to types defined in other standard headers. If it is unspecified if a type is signed or unsigned and the implementation has it as an unsigned type, a minimum macro with extension `_MIN`, and value `0` of the corresponding type is defined.

2 Each instance of these macros shall be replaced by a constant expression suitable for use in `#if` preprocessing directives. Its implementation-defined value shall be equal to or greater than the corresponding value given below. An implementation shall define only the macros corresponding to those typedef names it actually provides.[6]

7.20.3.1 Width of `ptrdiff_t`

1

`PTRDIFF_WIDTH`	17

7.20.3.2 Width of `sig_atomic_t`

1

`SIG_ATOMIC_WIDTH`	8

7.20.3.3 Width of `size_t`

1

`SIZE_WIDTH`	16

[6] A freestanding implementation need not provide all of these types.

7.20.3.4 Width of `wchar_t`

```
1 WCHAR_WIDTH 8
```

7.20.3.5 Width of `wint_t`

```
1 WINT_WIDTH 16
```

Appendix: pages with diffmarks of the proposed changes

The following page numbers are from the particular snapshot and may vary once the changes are integrated.