Proposal for C2x
WG14 N2960

Title:	_BitInt Fixes
Author, affiliation:	Aaron Ballman, Intel Philipp Klaus Krause, Albert-Ludwigs-Universität Freiburg Elizabeth Andrews, Intel
Date:	2022-04-11
Proposal category:	Bug Fixes
Abstract:	Offers several minor corrections to the wording for bit-precise integer types.

_BitInt Fixes

Reply-to: Aaron Ballman (aaron@aaronballman.com),
Philipp Klaus Krause (krauseph@informatik.uni-freiburg.de)
Elizabeth Andrews (elizabeth.andrews@intel.com)
Document No: N2960
Date: 2022-04-11

Summary of Changes

N2960

- Prohibit using a bit-precise integer type as a compatible type for an enumeration

N2946

- Original proposal

Introduction and Rationale

After adding bit-precise integer types [N2763] to C23, some minor issues with the wording were discovered. This paper attempts to correct the issues known at this time.

The proposed changes in this document are orthogonal and should be voted on separately by the committee for inclusion in C23. All proposed wording in this document is a diff from WG14 NYYYY. Green text is new text, while ~~red~~ text is deleted text.

Clarifications to Integer Promotions

The wording on integer promotion is ambiguous. While the examples clarify it, the wording should still be improved.

Proposed Straw Poll

Does WG14 wish to adopt NXXXX Clarifications to Integer Promotions into C23?

Proposed Wording

Modify 6.3.1.1p2 (*drafting note: there is a colon inserted before the deleted text, but it may be hard to visually spot depending on how the text renders.*):

If the original type is not a bit-precise integer type (6.2.5): ~~and~~ if an int can represent all values of the original type (as restricted by the width, for a bit-field)...

Clarify Increment/Decrement Behavior

The wording for the prefix increment operator says that the expression `E++` is equivalent to `(E+=1)`. If `E` is a value of bit-precise integer type, this implies a conversion between the bit-precise integer and the integer literal `1`. Instead, it should be made clear that there is no conversion involved as the integer literal should be of the same type as the expression `E`.

Proposed Straw Poll

Does WG14 wish to adopt NXXXX Clarify Increment/Decrement Behavior into C23?

Proposed Wording

Modify 6.5.3.1p2:

The value of the operand of the prefix `++` operator is incremented. The result is the new value of the operand after incrementation. The expression `++E` is equivalent to `(E+=1)`, where the value 1 is of the appropriate type. …

Exempt [u]intmax_t From Being Large Enough to Represent a Bit-Precise Integer

The current wording requires `intmax_t` and `uintmax_t` to be large enough to hold any value of an arbitrary bit-precise integer width. This could be a burden on implementations wanting to support a large width for bit-precise integers, and on users of `intmax_t` and `uintmax_t`. We propose to drop the requirement.

Proposed Straw Poll

Does WG14 want to drop the requirement of [u]intmax_t being able to hold any value of a bit-precise integer for C23 as in NXXXX?

Proposed Wording

Modify 7.20.1.5p1:

The following type designates a signed integer type capable of representing any value of any signed integer type other than a signed bit-precise integer type:

```
intmax_t
```

The following type designates an unsigned integer type capable of representing any value of any unsigned integer type other than an unsigned bit-precise integer type:

Which Types in stdint.h Should be Allowed to be a Bit-Precise Integer Type?

The current wording disallows the use of bit-precise integer types for the types `[u]intN_t`, but is unclear on [u]`intptr_t` and [u]`intmax_t`. The intent of the original proposal for bit-precise integer types was to disallow their use for all types defined in stdint.h.

On the other hand, the rationale for disallowing their use was that bit-precise integers narrower than `int` behave differently with regards to integer promotion vs. other types. So, it could make sense to only disallow the use of bit-precise integer types for stdint.h types narrower than `int`. This would allow implementations to easily provide e.g. an efficient [u]`intptr_t` type when `int` is 16 bits but pointers are 24 bits without needing support for a 24-bit integer type other than the bit-precise type.

Proposed Three-Way Straw Poll

Which stdint.h types should be allowed to be bit-precise integer types?

0) Leave it as is – `[u]intN_t` may not be bit-precise, but `[u]intptr_t` and `[u]intmax_t` are unclear.

1) None of `[u]intN_t, [u]intptr_t and [u]intmax_t.`

2) None of `[u]intN_t, [u]intptr_t` and `[u]intmax_t`, unless they are wider than `int`.

Proposed Wording 1)

Modify 7.20.1.4p1:

The following type designates a signed integer type other than a bit-precise integer type with the property that…

…

The following type designates an unsigned integer type other than a bit-precise integer type with the property that…

Modify 7.20.1.5p1:

The following type designates a signed integer type other than a bit-precise integer type capable of representing any value of any signed integer type:

```
intmax_t
```

The following type designates an unsigned integer type other than a bit-precise integer type capable of representing any value of any unsigned integer type:

Proposed Wording 2)

Modify 7.20p4:

None of the types shall be defined as a synonym for a bit-precise integer type, unless the width of the type is wider than `int`.

Modify 7.20.1.4p1:

The following type designates a signed integer type other than a bit-precise integer type with a width the same as or narrower than `int` with the property that…

…

The following type designates an unsigned integer type other than a bit-precise integer type with a width the same as or narrower than `int` with the property that…

Modify 7.20.1.5p1:

The following type designates a signed integer type other than a bit-precise integer type with a width the same as or narrower than `int` capable of representing any value of any signed integer type:

```
intmax_t
```

The following type designates an unsigned integer type other than a bit-precise integer type with a width the same as or narrower than `int` capable of representing any value of any unsigned integer type:

Type Compatibility With Enumerations

The standard currently allows an enumerated type to be compatible with “a signed integer type, or an unsigned integer type”, which currently allows for an enumerated type to be compatible with a bit-precise integer type. However, bit-precise integer types do not undergo integer promotion, and (considering C90 DR013) it is unspecified whether the composite type of an enum and its compatible integer type is the enum or the compatible integer type. So we think it is best to restrict enumerated types such that they

cannot be compatible with a bit-precise integer type in order to reduce surprising behavior for users involving integer promotions.

Proposed Straw Poll

Does WG14 wish to adopt NXXXX Type Compatibility With Enumerations into C23?

Proposed Wording

Modify 6.7.2.2p5:

Each enumerated type shall be compatible with char, a signed integer type, or an unsigned integer type. The choice of type is implementation-defined, but shall not be a bit-precise integer type and shall be capable of representing the values of all the members of the enumeration.

Acknowledgements

We would like to recognize the following people for their help with this work: Jens Gustedt and Joseph Myers.

References

[N2763]
Adding a fundamental type for N-bit integers. Ballman, et al. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2763.pdf