Proposal for C2x
WG14 N3138

Title:	Rebuttal to N2713 Integer Constant Expressions
Author, affiliation:	Aaron Ballman, Intel
Date:	2023-06-21
Proposal category:	Issue

Abstract: The original paper makes it invalid to accept other forms of integer constant expressions, but this diverges from existing practice and breaks code.

Rebuttal to N2713 Integer Constant Expressions

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N3138
Date: 2023-06-21

Summary of Changes

N3138

- Made it implementation-defined whether an extended constant expression is an ICE
- Reworded the footnote accordingly

N3125

- Original proposal

Introduction and Rationale

WG14 N2713 took the committee resolution of DR312 and added its wording to the C2x draft standard, which proscribes implementations from accepting other forms of integer constant expressions. This was done because of concerns that extended integer constant expressions may form a constant array type in one implementation and a variable-length array type in another.

We would like N2713 (https://www.open-std.org/jtc1/sc22/wg14/www/docs/n2713.htm) removed from C2x because it deviates from existing implementation practice in too many areas and the result of the changes potentially introduces surprising, breaking effects on existing user code. We agree that the paper is trying to solve an important problem, but as we investigated the changes we'd need to make to our implementation, it appears to be sufficiently disruptive that we might not implement it in Clang due to these concerns.

Below are several examples of where this change diverges from existing implementation practice. While these examples may appear contrived, the problems are not limited to use of `_Static_assert` nor to such simple constant expressions in real world code. (Note, the ICC "failures" in the Compiler Explorer links below are sometimes due to printing the tool's deprecation warning and are unrelated to the example; you can expand the compilers to see their exact output.)

```
_Static_assert(_Generic(1, int : 1), "");
```
https://godbolt.org/z/5zTYcMYMb (All tested compilers accept)
(`_Generic` is perhaps not allowed in an ICE – the associations for the expression are operands that don't match the ones listed in 6.6p8.)

```
_Static_assert(_Generic(1, int : 1, float : 1.0f ), "");
```
https://godbolt.org/z/b4MPbbrff (All tested compilers accept)
(Assuming that association operands are fine, this use of `_Generic` has an association which includes a floating-point constant which is not allowed in an ICE.)

`_Static_assert((int)(float)1.0f, "");`
https://godbolt.org/z/5qTf3Eqx1 (GCC and Clang correctly diagnose while accepting the code, other implementations silently accept it)
(Cast operations are only allowed to convert an arithmetic type to an integer type, so the intermediate cast is not allowed in an ICE even though it has no effect.)

`_Static_assert((int)(0 ? 1.0f : 1), "");`
https://godbolt.org/z/Wdq1Yddee (GCC and Clang correctly diagnose while accepting the code, other implementations silently accept it)
(The float operand is not allowed in a constant expression despite the branch not being executed and the expression result being cast to int.)

`_Static_assert(1 ?: 1, "");`
(GCC, Clang, and ICC all implement this extension and do not diagnose it as being invalid in an ICE, but it's not clear if this is "another form" of constant expression or not. If it is a valid ICE, then consider `_Static_assert(__builtin_strlen("test"), "");` -- does the use of `"test"` prevent this from being an integer constant expression? What about the use of a function call operator? The second example is also accepted by GCC and Clang, but is diagnosed by ICC.)

We believe more examples exist and we are reasonably sure that more implementations exist with different behaviors. DR312 made it clear that the committee's intent was that implementations cannot add additional forms of integer constant expressions. However, the standard's unambiguous wording allowing arbitrary extensions to constant expressions was present since at least ANSI C and had not changed in 30+ years until adopting N2713. That's a long time for implementations to build up considerable uses of extensions to integer constant expressions they support and removing those extensions will change the meaning of user code. Hopefully this is sufficiently compelling to the committee to warrant reconsidering N2713 in light of the current implementation landscape and implementer concerns with the changes. The paper, while admirable in its goals and faithful to the committee's intent in 2006, has the potential to do more harm than good and we believe more work is needed in this area, especially to avoid silently converting constant arrays into variable length arrays, and that work is better suited to a future revision of C given where we are in the release schedule.

Discussion during the Jun 2023 WG14 meeting resulted in a suggestion that we change the wording to require implementations to define whether the extended constant expression is or is not an integer constant expression.

Proposed Straw Poll
Does WG14 want to adopt the proposed wording from N3138?

Proposed Wording
All proposed wording in this document is a diff from WG14 N3096. Green text is new text, while ~~red~~ text is deleted text.

Modify 6.6p14:

An implementation may accept other forms of constant expressions; however, ~~they are not~~ it is implementation-defined whether they are an integer constant expression. Footnote)

Modify footnote 135:

For example, in the statement `int arr_or_vla[(int)+1.0];`, while possible to be computed by some implementations as an array with a size of one, it is implementation-defined whether this ~~still~~ results in a variable length array declaration or a declaration of an array of known constant size of automatic storage duration.

Acknowledgements

I would like to recognize the following people for their help in this work: Joseph Myers, Jens Gustedt, Martin Uecker, and Robert Seacord.