Limit [[assume]] to conditional-expressions

Document No.	**P2507 R1**	Date	2021-12-15
Reply To	Peter Brett **pbrett@cadence.com**	Audience:	EWG

Revisions

R1	New title. Add citation to P2461R1.

Introduction

It is currently proposed in P1774R5 "Portable assumptions" that the `[[assume(…)]]` attribute should accept an *assignment-expression* [1]. However, there is no evidence that it is useful to assume any expression that is not a *conditional-expression*.

Design

Currently, P1774R5 requires the argument of the assume attribute to be an *assignment-expression* contextually-convertible to bool. An *assignment-expression* can be:

- *yield-expression*: an assume attribute is not a function suspension context
- *throw-expression*: never contextually convertible to bool
- *conditional-expression*
- *logical-or-expression assignment-operator initializer-clause*

The motivation and design for the assume attribute does not include any examples of assuming a *logical-or-expression assignment-operator initializer-clause* sequence, even when discussing examples of side-effect corner cases that need to be avoided.

Every motivating use-case that the author is aware of, both in P1774R5 and elsewhere, assumes a *conditional-expression*. This, along with the contextual conversion to bool, strongly suggests that *conditional-expression* is the best model of "things that can be assumed."

Note that:

1. Related compiler intrinsics such as MSVC/icc's `__assume()` or clang's `__builtin_assume()` accept an *assignment-expression*.
2. The `if` and `while` statements each accept an *expression* in their *condition*.

By changing `[[assume(…)]]` from *assignment-expression* to *conditional-expression*, we can:

- ensure that typos like `[[assume(x = 42)]]` are not silently accepted by conforming implementations
- continue to permit `[[assume((x = 42))]]` as an escape hatch (*primary-expression*)
- leave open the door to expanding the range of accepted expressions in the future

If we do not narrow the grammar before `[[assume(…)]]` appears in the IS, then it will not be possible to do so in the future.

As a point of comparison, P2461R1 "Closure-Based Syntax for Contracts" also proposes the use of *conditional-expression* to model a 'truthy' expression [2].

Proposed wording

Editing notes

All wording is relative to P1774R5 [1].

Assumption attribute [dcl.attr.assume]

Update ¶1:

> The *attribute-token* `assume` may be applied to a null statement; such a statement is an assumption. An *attribute-argument-clause* shall be present and shall have the form:
>
> (~~assignment~~conditional-expression)

Constant expressions [expr.const]

Update ¶5:

> If *E* satisfies the constraints of a core constant expression, but evaluation of *E* would evaluate an operation that has undefined behavior as specified in [library] through [thread] of this document, a statement with an assumption ([dcl.attr.assume]) whose converted ~~assignment~~conditional-expression would not evaluate to true, or an invocation of the va_start macro ([cstdarg.syn]), it is unspecified whether e is a core constant expression.

Update ¶6:

> For the purposes of determining whether an expression E is a core constant expression, the evaluation of a call to a member function of std::allocator<T> as defined in [allocator.members], where T is a literal type, does not disqualify E from being a core constant expression, even if the actual evaluation of such a call would otherwise fail the requirements for a core constant expression. Similarly, the evaluation of a call to std::construct_at or std::ranges::construct_at does not disqualify E from being a core constant expression unless the first argument, of type T*, does not point to storage allocated with std::allocator<T> or to an object whose lifetime began within the evaluation of E, or the evaluation of the underlying constructor call disqualifies E from being a core constant expression. Further, a statement with an assumption ([dcl.attr.assume]) whose converted ~~assignment~~conditional -expression is itself not a core constant expression does not disqualify E from being a core constant expression.

References

[1] T. Doumler, “D1774R5 Portable Assumptions,” 15 Dec 2021. [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2021/p1774r4.pdf.

[2] G. Ažman, C. Sunstrum and B. Kozicki, “P2461R1 Closure-Based Syntax for Contracts,” 15 Nov 2021. [Online]. Available: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2021/p2461r1.pdf.