Attributes for contract assertions

Timur Doumler (papers@timur.audio)
Joshua Berne (jberne4@bloomberg.net)

Document #: P3088R1
Date: 2024-02-13
Project: Programming Language C++
Audience: SG21

Abstract

In this paper, we discuss several ways in which attributes could appertain to a contract assertion. The grammar in the Contracts MVP does not currently allow this, but it would be useful to allow it. We propose an infix location for contracts-specific attributes appertaining to any form of contract assertion, which could be useful for vendor-specific mechanisms to add local control to the various implementation-defined aspects of the Contracts MVP. Further, we propose that assertion statements, like all other statements evaluated at runtime, support a prefix location for non-contracts-specific attributes.

1 Introduction

There is a long-standing practice in the C++ grammar to permit attributes to appertain to any entity where this might be useful. Where such a permission is missing, it has often been added later via a Core Issue. For example, [CWG1657] added attributes for namespaces and enumerators.

This is common even in cases where there is no attribute in the C++ standard itself that could appertain to the given entity, and no plans to add such an attribute to the C++ standard; rather, the allowance in the grammar to place attributes on the entity in question is motivated by a desire to enable vendor-specific extensions via non-standard attributes where fine-grained control beyond the semantics of the C++ language might be useful. For example, [CWG2262] added attributes for *asm-definition*s with the explicit motivation that, first, there does not seem to be a good reason to not allow it, and second, it might be handy for vendor-specific extensions.

The same motivation applies to contract assertions, which the Contracts MVP paper [P2900R5] proposes to add to C++26. We can distinguish two distinct categories of attributes that might appertain to contract assertions:

1. Contracts-specific attributes that can appertain to any contract assertion (`pre`, `post`, and `contract_assert`), for example, vendor-specific labels that influence the implementation-defined choice of semantic with which a contract assertion will be evaluated — something the contracts MVP does not currently allow for.

2. Attributes such as `[[likely]]` and `[[unlikely]]` that can appertain to any statement that involves some runtime evaluation. The grammar in [P2900R5] does not allow such attributes to appertain to an *assertion-statement*, making it the odd duck among C++ statements that have runtime behaviour.

This paper contains a proposal for each category of attributes, described in Section 2 and Section 3, respectively. These two proposals can be adopted into the Contracts MVP either together or separately.

2 Attributes appertaining to contract assertions

2.1 Motivation

There are many well-known use cases for annotations or *labels* on contract assertions. They were featured in many previous proposals to add contracts to C++, reaching as far back as 2005 (the "importance ordering" annotation in [N1866]). A particularly important use case for such labels is to annotate a contract assertion with a property that will affect which contract semantic is chosen for its evaluation at runtime, for example:

— a contract level such as `audit`, indicating that checking the contract assertion would violate the complexity and/or performance guarantees of the function (included in C++20 Contracts, see [P0542R5]);

— a contract property such as `new`, indicating that the contract assertion was newly added to a legacy codebase;

— an explicit contract semantic such as `enforce`, `assume`, etc, indicating that the contract assertion should be evaluated with this semantic (proposed for C++20 Contracts in [P1429R3] and other papers).

The MVP offers no tools whatsoever for such control and leaves the selection mechanism for the contract semantic entirely up to the implementation. For an MVP, this is fine. We do not propose adding labels to contract assertions for C++26; instead, they should be considered as a post-MVP extension for C++29 (see discussion in [P2755R0] and [P2885R3]). However, it would be very helpful if compiler vendors could already start experimenting with adding labels to contract assertions as vendor extensions to gain the necessary implementation and usage experience. Importantly, tools to influence the selected semantic might vary from vendor to vendor, but all would apply equally to the same contract assertions.

The natural approach is to implement such extensions of the Contracts facility as vendor-specific attributes, e.g. `[[vendor::audit]]`. The attribute-based approach provides a great deal of portability compared to other possible approaches such as using vendor-specific keyword specifiers like `__vendor_audit`, or using a vendor-specific syntax for the entire contract annotation.

To enable this approach, we need to add a syntactic location for an *attribute-specifier-seq* appertaining to the contract assertion to the proposed Contracts MVP grammar. In the following subsection, we consider different syntactic locations in order to find the ideal one.

2.2 Possible syntactic locations

The syntactic location for an *attribute-specifier-seq* appertaining to the contract assertion should be consistent across all three syntactic constructs that can introduce a contract assertion: *precondition-specifier*s, *postcondition-specifier*s, and *assertion-statement*s. Inconsistent locations for contracts-specific attributes would be user-hostile given that all three syntactic constructs are part of the same facility and have a consistent syntax otherwise.

2.2.1 Prefix location

We could consider placing the *attribute-specifier-seq* before the contract assertion:

```
bool binary_search(Range r, const T& value)
  [[vendor::audit]] pre (is_sorted(r));

void f() {
  int i = get_i();
  [[vendor::assume]] contract_assert (i > 0);
  // ...
}
```

For `pre` and `post`, the prefix location has at least three problems. First, it would involve identifying something as part of a contract assertion before seeing the contextual keyword `pre` or `post`, which would be naturally challenging for implementations and readability. Second, it would collide with the existing grammar for an attribute appertaining to a *lambda-expression*:

```
auto f = [](int i) [[vendor::xxx]]  // this attribute appertains to the lambda-expression
  pre (i > 0) {
  // ...
}
```

Finally, when there are multiple function-contract assertions on a single function it becomes highly visually unclear whether such attributes appertain to the contract assertion before or after the attribute:

```
void f() pre(p1()) [[ vendor::audit ]] post(p2());
```

This option is therefore not viable for `pre` or `post`. It follows that it is also not viable for `contract_assert`, since we want all three syntactic forms to be consistent with each other. On the other hand, for `contract_assert`, attributes in the prefix location would be consistent with all other statements.

To resolve this seeming contradiction, we propose that for `contract_assert`, attributes that are specific to assertion statements, such as labels controlling the contract semantic, should be in a non-prefix location consistent with `pre` and `post`, as discussed in this section, while attributes that can appertain to any statement, such as `[[likely]]` and `[[unlikely]]`, should be in the prefix location. This solution makes `contract_assert` consistent with `pre` and `post` and simultaneously consistent with all other statements (see Section 3 for a more detailed discussion of attributes that can appertain to any statement).

2.2.2 Suffix location

We could consider placing the *attribute-specifier-seq* after the contract assertion:

```
bool binary_search(Range r, const T& value)
  pre (is_sorted(r)) [[vendor::audit]];

void f() {
  int i = get_i();
  contract_assert (i > 0) [[vendor::assume]];
  // ...
}
```

While the suffix location seems *technically* possible — as far as we can tell, it does not create any outright collisions with existing grammar — it is far from ideal, in particularly because, just like the prefix location, it is not obvious to the reader what the attribute appertains to:

```
// which pre does the attribute appertain to?
void f(int* i)
  pre(i != nullptr) [[vendor::xxx]] pre (*i > 0);

// does the attribute appertain to the pre or to the function g as a whole?
void g(int j) pre(j > 0) [[vendor::yyy]];
```

We therefore do not propose the suffix location.

2.2.3 Inside the predicate

We could consider placing the *attribute-specifier-seq* somewhere inside the contract predicate, for example

```
void f(int i)
  pre ([[vendor::xxx]] i > 0);
```

or

```
void f(int i)
  pre (i > 0 [[vendor::xxx]]);
```

However, this option is confusing as it suggests that the attribute would appertain to the expression rather than to the contract assertion as a whole. The latter is not actually possible (under the current grammar rules, an *attribute-specifier-seq* cannot appertain directly to a *conditional-expression*; in every case where an *attribute-specifier-seq* might be part of an expression is nested within another grammar construct and not adjacent to other constructs that can have attributes) and therefore technically there is no ambiguity, but this choice does not seem friendly to the human reader. The first variant also has the problem that for `post`, it is unclear whether the attribute would instead appertain to the *result-name-introducer*. We therefore propose that the location should be outside of the predicate.

2.2.4 Infix location (between keyword and predicate)

The remaining possible syntactic location is between the `pre`, `post`, or `contract_assert` keyword and the predicate:

```
bool binary_search(Range r, const T& value)
  pre [[vendor::audit]] (is_sorted(r));

void f() {
  int i = get_i();
  contract_assert [[vendor::assume]] (i > 0);
  // ...
}
```

Being between the two other elements that are part of the contract assertion itself — the keyword and the predicate — this location is always unambiguous, both for the C++ parser and the human reader. It also has the advantage that attributes applying to varied precondition or postcondition specifiers can be formatted and indented in a consistent manner, not hanging off far from the `pre` or `post` separated by an arbitrarily complex expression.

Since this infix location is the only one not suffering from any obvious problems, it is the location we propose in this paper for contracts-specific attributes appertaining to a contract assertion.

At some point in the future additional features might be proposed (such as captures for postconditions or `requires`-clauses on function contract assertions) that might also be placed in this same syntactic location. As each such post-MVP feature is considered, its location relative to attributes that appertain to the contract assertion should be considered as well. We recommend, though, that such features follow similar orderings to the full function declaration where possible, while keeping those things which are runtime-evaluated like the predicate as close to the predicate as possible. Therefore, `requires`-clauses should precede the *attribute-specifier-seq* and captures should come immediately prior to the predicate.

3 Attributes appertaining to an *assertion-statement*

In [P2900R5], `contract_assert` is syntactically a statement: the *assertion-statement*. Initially, when the "natural" syntax was adopted, `contract_assert` was an expression (see [P2961R2]). However, SG21 recently downgraded `contract_assert` to a statement, because having it as an expression would require answering the question whether this expression is potentially-throwing or non-throwing for the purposes of the `noexcept` operator and deduced exception specifications, and SG21 failed to reach a consensus on this question (see [P2932R3], [P2969R0], [P3113R0], and [P3114R0]).

The infix location for attributes appertaining to contract assertions makes `contract_assert` seemingly inconsistent with other statements: an attribute that appertains to a statement, such as `[[likely]]` and `[[unlikely]]`, always goes into the prefix location. Note that the C++ grammar does not give blanket permission to prefix any statement with an *attribute-specifier-seq*; instead, this permission is given explicitly for each kind of statement. The proposed grammar in [P2900R5] lacks this explicit permission for *assertion-statement*, making it the odd duck among C++ statements.

To resolve this seeming contradiction, we propose that for `contract_assert`, attributes that are specific to assertion statements, such as vendor-specific labels controlling the contract semantic, should be in a non-prefix location consistent with `pre` and `post`, as discussed in Section 2, while attributes that can appertain to any statement, such as `[[likely]]` and `[[unlikely]]`, should be in the prefix location. This solution makes `contract_assert` consistent with `pre` and `post` and simultaneously consistent with all other statements. To make the distinction between the different kinds of attributes clear, we propose wording specifying that for `contract_assert`, attributes in the prefix location appertain to the *assertion-statement*, while attributes in the infix location appertain to the contract assertion introduced by the *assertion-statement*.

Note that having two syntactic locations for attributes in the same entity is not novel: declarations can be prefixed with an attribute, but declarations can also have attributes on the *declarator-id*, i.e. deep inside the declaration.

Note further that this solution with two different locations with two different meanings is perfectly forward-compatible with upgrading `contract_assert` to an expression later. In case we decide to do so, code like

```
[[likely]] contract_assert(x);
```

will automatically become legal anyway, with the `[[likely]]` attribute now appertaining to the *expression-statement* that contains the `contract_assert` expression. At the same time, attributes in the infix position can continue to appertain to the contract assertion introduced by the expression. Just like before, the former location is for attributes that can appertain to any statement, while the latter is for contracts-specific ones; the meaning of the code does not change.

4 Proposed wording

The proposed changes are relative to the draft wording in [P2900R5].

4.1 Attributes appertaining to contract assertions

Modify [stmt.contract.assert]:

assertion-statement:
`contract_assert` *attribute-specifier-seq*$_{opt}$ `(` *conditional-expression* `)` `;`

An *assertion-statement* introduces a contract assertion ([basic.contract]). The optional *attribute-specifier-seq* appertains to the introduced contract assertion. [*Note:* An *assertion-*

statement allows the programmer to specify a state of the program that is considered incorrect when control flow reaches the assertion-statement.

Modify [dcl.contract.func]:

precondition-specifier:
`pre` *attribute-specifier-seq*$_{opt}$ `(` *conditional-expression* `)`

postcondition-specifier:
`post` *attribute-specifier-seq*$_{opt}$ `(` *result-name-introducer*$_{opt}$ *conditional-expression* `)`

result-name-introducer:
identifier `:`

A *function contract assertion* is a contract assertion ([basic.contract]) associated with a function. Each *function-contract-specifier* of a *function-contract-specifier-seq* (if any) of an unspecified first declaration of a function introduces a corresponding function contract assertion for that function. The optional *attribute-specifier-seq* appertains to the introduced contract assertion. [*Note:* The *function-contract-specifier-seq* of a *lambda-declarator* applies to the call operator or operator template of the corresponding closure type ([expr.prim.lambda.closure]).

Modify [dcl.attr.grammar], paragraph 1:

Attributes specify additional information for various source constructs such as types, variables, names, contract assertions, blocks, or translation units.

4.2 Attributes appertaining to an *assertion-statement*

The changes below can be adopted together with or separately from the changes above for attributes appertaining to contract assertions.

Modify [stmt.pre], paragraph 1:

statement:
attribute-specifier-seq$_{opt}$ *expression-statement*
attribute-specifier-seq$_{opt}$ *compound-statement*
attribute-specifier-seq$_{opt}$ *selection-statement*
attribute-specifier-seq$_{opt}$ *iteration-statement*
attribute-specifier-seq$_{opt}$ *jump-statement*
attribute-specifier-seq$_{opt}$ *assertion-statement*
declaration-statement
attribute-specifier-seq$_{opt}$ *try-block*

Should SG21 decide to not adopt these changes, we should add a section to the front matter of [P2900R5] clarifying explicitly that unlike other kinds of statements, an *assertion-statement* may not have attributes appertaining to it.

Document history

— **R0**, 2024-02-06: Initial version.

— **R1**, 2024-02-13: Added discussion and proposal for non-contracts-specific attributes appertaining to an *assertion-satement*; updated proposed wording

Acknowledgements

The authors wish to thank Tom Honermann, Andrzej Krzemieński, Ville Voutilainen, and Jens Maurer for their feedback on the initial version of this paper.

References

[CWG1657] Richard Smith. Core Issue 1657: Attributes for namespaces and enumerators. https://cplusplus.github.io/CWG/issues/1657.html, 2013-08-26.

[CWG2262] Richard Smith. Core Issue 2262: Attributes for *asm-definition*. https://cplusplus.github.io/CWG/issues/2262.html, 2016-05-04.

[N1866] Lawrence Crowl and Thorsten Ottosen. Proposal to add Contract Programming to C++ (revision 3). https://wg21.link/n1866, 2005-08-24.

[P0542R5] G. Dos Reis, J. D. Garcia, J. Lakos, A. Meredith, N. Myers, and B. Stroustrup. Support for contract based programming in C++. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0542r5.html, 2018-06-08.

[P1429R3] Joshua Berne and John Lakos. Contracts That Work. https://wg21.link/p1429r3, 2019-07-23.

[P2755R0] Joshua Berne, Jake Fevold, and John Lakos. A Bold Plan for a Complete Contracts Facility. https://wg21.link/p2755r0, 2023-09-13.

[P2885R3] Timur Doumler, Gašper Ažman, Joshua Berne, Andrzej Krzemieński, Ville Voutilainen, and Tom Honermann. Requirements for a Contracts syntax. https://wg21.link/p2885r3, 2023-10-02.

[P2900R5] Joshua Berne, Timur Doumler, and Andrzej Krzemieński. Contracts for C++. https://wg21.link/p2900r5, 2024-02-15.

[P2932R3] Joshua Berne. A Principled Approach to Open Design Questions for Contracts. https://wg21.link/p2932r3, 2024-01-15.

[P2961R2] Timur Doumler and Jens Maurer. A natural syntax for Contracts. https://wg21.link/p2961r2, 2023-11-08.

[P2969R0] Timur Doumler, Ville Voutilainen, and Tom Honermann. Contract checks are potentially-throwing. https://wg21.link/p2969r0, 2023-12-04.

[P3113R0] Timur Doumler. Contract assertions, the `noexcept` operator, and deduced exception specifications. https://wg21.link/p3113r0, 2024-02-01.

[P3114R0] Andrzej Krzemieński. `noexcept(contract_assert(_))`. https://wg21.link/p3114r0, 2024-02-02.