Attributes for contract assertions

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

Document #: P3088R0
Date: 2024-02-06
Project: Programming Language C++
Audience: SG21

Abstract

In this paper, we propose to add an *attribute-specifier-seq* to the grammar for contract assertions in the Contracts MVP. While this does not directly change what the user can do with the Contracts MVP, it follows the usual practice of allowing an *attribute-specifier-seq* on any syntactic construct where this makes sense. It also allows compiler vendors to experiment with adding labels to contract assertions as vendor extensions while still being conformant.

1 Motivation

In the C++ Standard, we have a long-standing practice of allowing an *attribute-specifier-seq* on any syntactic construct where this makes sense. Where such allowance is missing, it is typically added later via Core Issues, see for example [CWG1657] and [CWG2262]. Note that the latter allows attributes on an *asm-definition*, even though the C++ Standard itself does not contain any attributes that would appertain to an *asm-definition*. This allowance exists for compiler vendors to be able to ship vendor extensions while still being conformant.

For contract assertions in particular, we know of many use cases for annotating them with labels, for example to express certain properties that will affect which contract semantic is chosen for their evaluation:

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

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

— explicit contract semantics such as `enforce`, `assume`, etc (this was proposed for C++20 Contracts in [P1429R3]).

We do not propose adding any such labels to contract assertions for C++26; they should be considered as a post-MVP extension for C++29 (see [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. The best approach we can think of is to implement such labels as vendor-specific attributes, e.g. `[[vendor::audit]]`. This is exactly what this paper seeks to allow.

2 Discussion

In order for this paper to be a complete proposal, we need to find the ideal location in the grammar for contract assertions where an *attribute-specifier-seq* can be inserted. The fundamental principle of the attribute grammar is that the places where we put attributes always clearly appertain to one entity. A syntax that makes it ambiguous whether an attribute in that location would apply to the contract assertion or to something else is not fit for purpose. In this section, we discuss the possible choices.

2.1 Before the contract assertion

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 `contract_assert`, this would be the choice most consistent with other statements: an attribute appertaining to a statement usually goes before that statement. However, for `pre` and `post`, this position has at least two 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) {
  // ...
}
```

This option is therefore not viable for `pre` or `post`. Regarding the possibility of adopting this option for `contract_assert`, we believe that it is more important for the syntax of `contract_assert` to be consistent with that of `pre` and `post` — since all three belong to the same language facility — than to be consistent with other kinds of statements.

2.2 After the contract assertion

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 this grammar location is *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 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 prefer to not adopt this option.

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. This 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 prefer a location outside of the predicate.

2.4 After the `pre`, `post`, or `contract_assert` keyword

The remaining possible syntactic position is just after the `pre`, `post`, or `contract_assert` keyword:

```
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 position is the only one not suffering from any obvious problems, it is the one we propose in this paper.

3 Proposed wording

Modify the proposed Contracts MVP grammar in [P2900R5] as follows:

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 `:`

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

The optional *attribute-specifier-seq* appertains to the introduced contract assertion.

Modify [dcl.attr.grammar] p1 as follows:

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

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.

[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.