Requires Clauses for Contract Assertions

Document #:	P4283R0
Date:	2026-07-15
Project:	Programming Language C++
Audience:	EWG

Reply-to: Joshua Berne <jberne4@bloomberg.net>

Abstract

Contract assertions in generic code can often be written for only a subset of the types that a template might support. Writing generic code when this is the case becomes an exercise in contorting around multiple otherwise-identical function declarations and associated definitions. The design of the C++26 Contracts allows for requires clauses on contract assertions to be easily added, and this proposal adds that support.

Contents

Revision History

Revision 0

- Original version of the paper

1 Introduction

The syntax of C++26 Contracts, introduced in [P2961R2], was designed to (amongst other things) allow for the introduction of a `requires` clause on function-contract assertions[1]:

```
template <typename T>
void f(T x)
  pre requires std::integral<T> (x > 0);
```

In generic code, situations like this come up often, and quite a few examples were explored in [P2755R1].

- An interesting case is when a generic container type might want to support an `assertInvariants` member that also checks the invariants of the contained object:

```
template <typename T>
concept has_assert_invariants
  = requires (const T& t) {
      t.assertInvariants();
    };

template <typename T>
void myContainer<T>::assertInvariants() const
  pre requires(has_assert_invariants<T>) (
    [&]{  // nested (immediately-invoked) lambda to iterate over members
      for (auto& x : *this) {
        x.assertInvariants();
      }}(), true );
```

- Another common case was shown in our sample implementation of the interface of `std::vector`, where for example many postconditions can only be checked for types that satisfy certain concepts:

```
template<class InputIterator>
constexpr vector(InputIterator first,
                 InputIterator last,
                 const Allocator& alloc = Allocator())
  pre requires(std::forward_iterator<InputIterator>)
      (std::distance(first, last) >= 0);   // validate that iterators form a range
```

[1] Note that [P2961R2] also pointed out that placing the requires clause after the conditional-expression would also be unambiguous. It would, however, be harder to read because it would depend on having an innate understanding of the correct order of components of the declaration after the function parameter list (where it might not be obvious to all readers that the requires clause applies to just the contract assertion and not the entire function declaration). Therefore we are proposing that the requires clause be unambiguously part of the assertion by placing it inside the assertion.

Within a function body this same goal can be achieved by wrapping uses of `contract_assert` inside `if constexpr` checks. On function declarations, however, the only way to achieve the same result is to duplicate the function declarations themselves with corresponding constraints (and then duplicate their definitions, or attempt to implement perfect forwarding to a shared implementation). The large amount of boilerplate and compile-time overhead to do this makes it highly unpleasant.

In the end, this is a simple proposal to add a small feature that has proven to be very useful in practice.

Proposal 1: Requires Clauses on Contract Assertions

Allow a requires clause on *precondition assertions*, *postcondition assertions*, and *assertion statements* associated with templated functions. When the requires clause is not satisfied, the assertion is discarded.

2 Implementation Experience

Requires clauses on contract assertions have been implemented in branches of GCC and Clang that are available on Compiler Explorer: https://godbolt.org/z/oTPGK4zqY.

Both implementations make these features available with the use of the `-fcontracts-p4283` command-line flag. They are also available with the flag `-fcontracts-p3850` that enables prototype implementations of many of the papers described in the overall plan in [P3850R1].

The changes to support this functionality were localized and largely leveraged existing tools in both compilers. The trickiest part was finding the specific point to check the constraint so that instantiation would not be triggered too aggressively. There are no other ABI implications to the change, as any contract assertion discarded during template instantiation is treated as if it is simply not present.

3 Wording Changes

These wording changes are relative to the C++29 working draft with git hash 14ed707b, last modified on Wed, 17 Jun 2026 14:18:34 +0100.

6 **Basics** **[basic]**

6.11 **Contract assertions** **[basic.contract]**

6.11.1 **General** **[basic.contract.general]**

Add a new paragraph after 6.11.1[basic.contract.general], paragraph 2:

2+a The optional *requires-clause* in a *precondition-specifier*, *postcondition-specifier*, or *assertion-statement* shall be present only if the associated or containing function is a templated function. If the associated constraints are not satisfied, the predicate is discarded and no contract assertion is introduced.

8 **Statements** [stmt]

8.9 **Assertion statement** [stmt.contract.assert]

Modify subclause 8.9[stmt.contract.assert]:

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

9 **Declarations** [dcl]

9.4 **Function contract specifiers** [dcl.contract]

9.4.1 **General** [dcl.contract.func]

Modify section 9.4.1[dcl.contract.func]:

function-contract-specifier-seq:
function-contract-specifier function-contract-specifier-seq$_{opt}$

function-contract-specifier:
precondition-specifier
postcondition-specifier

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

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

15 **Preprocessing directives** [cpp]

15.12 **Predefined macro names** [cpp.predefined]

Modify subclause 15.12[cpp.predefined], Table 22 [cpp.predefined.ft]:

Table 1: Table 22 — Feature-test macros [tab:cpp.predefined.ft]

Macro name	Value
⋮ *20 rows elided* ⋮	
`__cpp_constinit`	`201907L`
`__cpp_contracts`	`202502L`
`__cpp_contracts_requires`	`xxxxxxL`
`__cpp_decltype`	`200707L`
`__cpp_decltype_auto`	`201304L`
⋮ *56 rows elided* ⋮	

4 Conclusion

With this small change we improve the ergonomics of writing contract assertions in generic code. This improvement reduces barriers to writing correctness checks in generic code, allowing for more unfettered use of Contracts in general.

Acknowledgments

Claude (Anthropic) was used for editorial assistance during the preparation of this paper, as well as significant parts of the prototype implementations.

Bibliography

[P2755R1] Joshua Berne, Jake Fevold, and John Lakos, "A Bold Plan for a Complete Contracts Facility", 2024
http://wg21.link/P2755R1

[P2961R2] Timur Doumler and Jens Maurer, "A natural syntax for Contracts", 2023
http://wg21.link/P2961R2

[P3850R1] Timur Doumler and Joshua Berne, "A proposed plan for extending Contracts in C++29", 2026
http://wg21.link/P3850R1