Document Number: P3851R0
Date: 2025/09/29
Reply to: J. Daniel Garcia
e-mail: jdgarcia@inf.uc3m.es
Audience: WG21

Position on contracts assertion for C++26

J. Daniel Garcia
Computer Science and Engineering Dept.
University Carlos III of Madrid

Jose Gomez
Lely Industries

Raul Huertas
Qualcomm Europe Inc.

Javier Lopez-Gomez
Zimperium, Inc.

Jesus Martinez
Languages and Computer Sciences Dept.
University of Malaga

Francisco Palomo
Computer Science and Engineering Dept.
University of Cadiz

Victor Sanchez
Scalian Consulting Spain

Abstract

This paper presents our position on the contracts assertion feature for C++26. We collect a number of concerns that have been previously stated and that we consider critical in the context of the C++ contracts feature. Given those concerns, we conclude that contracts as proposed are not appropriate for C++26.

1 Change log

1.1 R0

Initial version of the paper.

2 Presentation

Contracts were incorporated into the C++26 draft. However, there are a number of concerns that still remain unsolved. We recommend that unless those concerns are properly addressed the proposal in its current form should be removed from the C++26 draft.

Here is a summary:

- **Lack of sufficient experience**: The contracts feature is a major feature that lacks enough deployment experience. While it is true that some portions have been experimented in projects, there are many features that have not been tried and tested sufficiently.

  Moreover, at this point we also lack enough user experience, implementation experience and build system experience. The latter is especially significant in presence of mixed mode builds. We should also have experience in multiple domains. What is acceptable in one problem domain becomes critically unacceptable in a different domain.

  The safest path would be to get more experience by providing the feature either in a technical specification or in a white paper, so that all the issues are better understood.

- **Mixed mode builds**: In P3835 [1] examples are given for a header file that contains an **inline** function with a contract assertion and is used from different translations units with different evaluation semantics. The same applies to **constexpr** functions, **consteval** functions and templates.

  This is a major safety issue as the same assertion might be checked or not depending on the caller. Yet subsequent code might depend on the validity of that assertion.

- **Multiple dependent assertions**: When there are multiple dependent assertions (a precondition that checks for null pointer and another precondition that dereferences that pointer), the evaluation in non-terminating modes may lead to undefined behavior.

  This is a safety issue. It may be mitigated by not evaluating dependent contract assertions when the first assertion fails.

- **Unsafe consequences**: In paper P3829 [2] examples are given on how contract assertions may lead to situations where a critical check may end-up elided. As the papers states this might be a powerful tool for new supply-chain attacks.

  This is another major safety issue. Contracts should not worsen the safety dimension of the language.

- **Constification**: Clause 6.11.1/4 makes any variable to be **const** within the predicate of a contract assertion. This also applies to the **this** pointer. This is specially problematic when invoking an overloaded function, as the overload resolution mechanism might select a different version than in other context.

  In addition, clause 9.4.1/7 states that by using a non-reference parameter of a function in a post-condition the parameter magically becomes **const**. In this case, adding a post-condition to a function is changing the signature of the function, which seems counter-intuitive.

  This is a major concern from the teachability point of view, as it will make the code harder to understand. Moreover, this might not be acceptable for projects where maintainability and simplicity are major drivers. It is also worth noting that constification violates a fundamental design principle for C++: given the same object, the same operation should have the same effect.

- **No support for function pointers**: Clause 9.4.1/8 states (inside a note) that " *A pointer to function, pointer to member function, or function type alias cannot have a function-contract-specifier-seq associated directly with it*". Pointer to functions are a first class citizen in C++.

  Pointer to functions and pointer to member functions are widely used in many code bases. Not supporting contracts on them will be a barrier to adopt contracts in those code bases.

- **No support for virtual functions**: C++ supports multiple styles of programming, including object-oriented programming where virtual functions play a major role. However, the contract feature explicitly forbids the use of contracts specifiers for virtual functions (clause 9.4.1/6). This is a major drawback of the feature.

  There are large codebases written in C++ that make extensive use of virtual functions. By providing a solution that does not support one of the major styles, the language is doing a disservice to such portion of the user community.

3 Conclusion

Given the list of identified issues we strongly recommend that, unless all issues can be solved, the contract feature itself is removed from C++26 and moved to a technical specification or white paper. We see unrealistic that the issues can be solved in the C++26 time frame. However, a TS or white paper could give the adequate feedback to have a more consistent and baked feature for C++29.

If we really adopt C++26 as it is, it may be a major failure. Moreover, the fact that contracts have strong safety issues and even might become a door for supply-chain attacks are a very strong concern. They decrease the overall safety of the language. Given the current focus on software safety, that does not seem an acceptable route.

If the committee decides to go ahead with contracts for C++26, still some issues pointed out in this paper are worth solving. But that seems like a partial and not very recommendable path.

References

[1] John Spicer, Ville Voutilainen, and Jose Daniel Garcia Sanchez. Contracts make c++ less safe – full stop. WG21 proposal P3835R0, ISO/IEC JTC1/SC22/WG21, September 2025.

[2] David Chisnall, John Spicer, Gabriel Dos Reis, Ville Voutilainen, and Jose Daniel Garcia Sanchez. Contracts do not belong in the language. WG21 proposal P3829R0, ISO/IEC JTC1/SC22/WG21, September 2025.

[3] Bjarne Stroustrup, Michael Hava, J. Daniel Garcia Sanchez, Ran Regev, Gabriel Dos Reis, John Spicer, J. C. van Winkel, David Vandevoorde, and Ville Voutilainen. Contract concerns. WG21 proposal P3573R0, ISO/IEC JTC1/SC22/WG21, January 2025.