SIS/TK611 considerations on Contract Assertions

P3849R1
Harald Achitz harald@swedencpp.se
Audience: EWG, CWG, LWG
2025-10-09

Revision history

- Revision 1, 2025-10-09
    - update with new paper references and some additional explanations
    - extend dependency section
    - add new section about 'marketing' aspect
    - layout document
- Revision 0, 2025-09-27, initial version

Introduction

The Swedish mirror committee has spent a notable amount of meeting time on the topic of contracts, now named contract assertions.

The outcome is:

- There is little strong support for contract assertions.
- There is considerable resistance to contract assertions.

This split makes a consensus affirmative vote, as we usually have it, unlikely.

Core Considerations Summarization

The main concerns have already been summarized in papers by other authors:

- P3573R0: Contract concerns
- P3835R0: Contracts make C++ less safe -- full stop!
- P3829R0: Contracts do not belong in the language

These papers outline safety concerns we share.

Additionally, there are other prominent voices in the community who have publicly stated that contracts, as they are, are problematic.

Contract assertions need guidelines that the compiler cannot check

An often-cited guideline is that, beyond runtime cost, contract checks must not change the behavior of a program, regardless of whether they are enabled or disabled.

There is a significant risk that this expectation will not be met in practice. We should not add features to the language that require extra directives to use correctly.

The impact on (binary) dependency management has not been sufficiently explored

Contracts introduce several new build configurations, but we have not yet seen concrete examples of how they interact with real-world build systems or complex dependency graphs. Potential problem scenarios are still not laid out, so the risks remain hard to judge.

We are aware that it will 'just add some more compiler flag', so there is no new info in this respect.

But we already have a flag soup. In today's world, we have debug builds, release builds, various sanitizer builds, profiling and coverage builds, and maybe more. And the average C++ developer has a hard time managing this, let alone doing complex builds with complex dependency trees.

With the current contract proposal we get four flavors more, potentially for each of the existing configurations. An interesting new aspect is, with contracts we can have a build of a library that falls in all contract categories, in case the library does not use contracts. None of the flags might have an impact. But we can't know that, and we still need to prepare all flavors if we want binary dependency management enabled for our users. Will this make an artifactory repo, or any other backend that stores binary dependencies, of 10 gigabytes blow up to 40 gigabytes?

Attractive new points of view have been added

P3853R0, A thesis+antithesis=synthesis rumination on Contracts adds an interesting view on the topic.

That such new views are added indicates that there is more to explore on the topic.

The topic has divided the community

Besides existing and likely newly added papers, discussions on public forums reflect the division.

Both camps have significant voices to make their point. And the opposition to P2900 seems more significant than the Hagenberg voting results indicate.

Additionally, the argumentation seems to some extent to no longer bring any new insight. The points have been made, and each camp references papers, often written by themselves. No progress towards agreement in sight.

This process is negative for the C++ community and is a marketing disaster.

New features should not create technical or reputational risks for the language

C++, both the language and the community, already face criticism from parts of the safety community for being unsafe and therefore unsuitable for certain domains.

Adding a safety feature that is in critics about opening new space for undefined behavior and adds rules to the user that user can easily get wrong seems problematic.

Adding a safety feature that opens new space for undefined behavior, depends on numerous implicit rules that users must know, and leaves too many aspects implementation-defined seems problematic.

The argument that such issues are 'easily identified during development and testing' feels too 1990s and no longer fits the realities of the second quarter of the 21st century.

Every feature that relies on too many implicit rules or adds new areas of undefined or implementation-defined behavior challenges the long-term stability and clarity of the language and its community.

If such features also provide opportunities for misuse or public criticism, leading to renewed doubts about C++ in professional or managerial circles, the result is not only a technical concern but also a reputational one for the language.

Summary

Based on the remaining questions, the concerns we have heard, and the friction this topic creates both in our group and across WG21, we believe it would be beneficial to give contract assertions more time. Given these circumstances, exploring the white paper route appears to be the most prudent way to pursue the work without jeopardizing the C++26 schedule.

We do not wish to delay C++26, but we want the record to reflect these concerns and to understand whether other national bodies share them.

Conclusion

- We ask WG21 to gather feedback from other national bodies and address the documented concerns before proceeding.
- We ask WG21 to resolve the outstanding discussions constructively, within the provenances they oversee, and to ensure relevant information is discoverable when needed.
- We ask WG21 to explore moving the topic to a white paper and to collaborate with compiler and tool vendors so we gain implementation experience before standardization.