# Why Contracts?


| | |
|---|---|
| Document #: | P3204R0 |
| Date: | 2024-11-07 |
| Project: | Programming Language C++ |
| Audience: | SG21 (Contracts) |


Reply-to: Joshua Berne <berne@notadragon.com>



**Abstract**

Questions often arise regarding why Contracts should be part of the C++ language and why the solution proposed in [P2900R10] should be the basis of that feature. As a co-author of that proposal and an active participant in SG21 (and the Standardization of Contracts since 2019), this paper is my attempt to answer some of those questions.


## Contents

## Revision History

Revision 0

- Original version of the paper

# 1 Introduction

Questions have been asked about why we should have Contracts in C++, whether [P2900R10] is the correct solution, and what should give us faith that the right path is being followed in the design of Contracts.

I would not be writing papers to explain each bit of reasoning behind the design in [P2900R10] if I did not firmly believe that the answers to these questions are all in favor of the design we are pursuing. In this paper, I will attempt to capture my own answers to these questions.

# 2 Questions

## 2.1 A "Rude" Question From Outside WG21

As [P2900R7] was being presented for the first time to a wider audience beyond SG21, Herb Sutter presented two questions to the group that he labelled the "50,000ft-view" questions. Herb emphasized that this question is not *his* question but one that he has repeatedly received from engineering executives and one for which EWG would ideally approve and publish its answer so he can point to it when he gets this question again in the future. On the surface, this question can appear to be fairly rude, but if we can't address the supposedly rude questions, then we have failed to build a useful, powerful, effective, and robust solution.

| Question 1: "Rude" Q | Herb Sutter, 2024-03-20 |
|---|---|
| Contracts have been available for decades (e.g., SAL, C# Code Contracts) but aren't widely used in the mainstream. What can make us confident that this one will succeed?<br>*Mattermost a few minutes ago: 'What are contracts for?'* | |

At its heart, the Contracts facility proposed in [P2900R10] is, primarily, an evolution of existing practice within the C++ community, starting with the `assert()` macro provided by the C Standard Library and extending through the many replacement macros that libraries have provided to fill the gaps between `assert()` and a facility that can be used at scale.

Bloomberg has `bsls_assert`, libstdc++ has `_GLIBCXX_DEBUG_ASSERT`, Google has `DCHECK`, Boost has `BOOST_ASSERT`, and many other publicly and privately available libraries have their own homegrown replacements for `assert()`. Functionality ranges from simply being a different spelling to a wide range of customizability through the preprocessor and even, in some cases, power runtime configuration options. These replacements are widely used, and all have features that the SG21 Contracts proposal aims either to support out of the box or to provide the foundation for supporting in the future.

More importantly, the SG21 Contracts proposal fixes many of the issues with macro-based solutions that cannot be addressed in a user-provided library and will thus be a highly appealing transition

for all the existing assertion libraries as soon as feature parity becomes a possibility.[1] Contracts, as proposed in [P2900R10], not only fixes the issues related to ODR violation and the syntactic messiness of using the preprocessor, but also integrates better with the language itself to ensure that newly introduced assertions can verify the validity of the programs to which they are added.

This last quality, which was first introduced in [P2834R1] and is explained further in [P2900R10], stems from what we have come to call the Prime Directive of Contracts.

**Principle 1: Prime Directive**

The presence or evaluation of a contract assertion in a program should not alter the correctness of that program (i.e., the property that evaluation of the program does not violate any provisions of its plain-language contract).

This principle and its ramifications are a key part of why Contracts will be easy to adopt into any system: They have been designed to maximize the ability to introduce them into a program without fundamentally altering the program itself. Thus, a program built with all contracts *ignored* can rely on the contract assertions themselves not altering the program's semantics. This principle maintains one of the longest-held tenets of the design of the C++ programming language: You don't pay for what you don't use.

The Contracts facility proposed by SG21 also aims for another long-term goal beyond improving and consolidating all of the assertion libraries that are currently in heavy use throughout the C++ community: to provide a strong foundation for annotating what makes a program correct and to thus facilitate robust static analysis of that correctness. By being able to leverage the full power of the C++ programming language to express contract annotations, we ensure that no naturally expressible checks are left unsaid in a program. This range of applicability maximizes the amount of information that can be then processed by formal correctness checkers and static analyzers to determine at compile time whether a program will actually run correctly. Rather than limit what can be expressed to only the small set of expressions that tools of today might understand, we maximize what can be expressed to avoid any chance that users will feel hindered from capturing what they consider to be the correct states of their program.[2]

Put more succinctly, Contracts in C++ will be a success because contract-checking facilities in C++ are already a success, and this feature has been designed from the start to (eventually) provide a superior replacement for all those facilities via integration with existing and future tools in a powerful fashion.

## 2.2 A "Pragmatic" Question

In his presentation, Herb presented a second question that is much more specifically targeted at this proposal and where it is in its lifetime.

[1] Even partial integration with the contract-violation handler, such as the library API proposed by [P3290R2], will be a powerful change in the ability to improve the correctness of real C++ programs.

[2] Of course, being a minimal viable product does mean that many conditions cannot be checked and that many should not yet be expressed as contract assertions due to their potentially destructive nature. The MVP is a space that has significant room for future growth by adopting features to give in-source mechanisms to express these properties, and more details of at least one potential future direction can be found in [P2755R1].

| Question 2: "Pragmatic" Q | Herb Sutter, 2024-03-20 |
|---|---|

C++0x concepts looked great, until we tried to apply them to the standard library. Will standardizing this feature be successful without first specifying how it would be used in the Standard Library?

*If we aren't ready for that: This is what TSes are for...*

Yes, Contracts will be successful because the C++ community has extensive experience using contract-checking facilities, like the proposed SG21 Contracts, in existing libraries, including most modern Standard Library implementations. This experience, combined with the Standard Library specification having been written with a facility like this in mind for many years (going back to when Contracts were in the Draft C++20 Standard; see [P0788R0]), means that the SG21 Contracts facility is well suited for capturing the preconditions and postconditions of functions in the Standard Library.

However, we do have some caveats.

- The Standard Library, while very foundational and ubiquitously deployed, is also one of the most complex generic libraries that the average C++ developer will encounter as part of their daily routine. Therefore, many of the contract checks the library might want to encode are conditional on element types or on expressing concerns that are highly nontrivial to encode without the addition of new features. The potential contract assertions that could be placed on `std::sort` are explored in [P3212R0]. A more forward-looking exploration of this can be found in Section 4.4 of [P2755R1] where we show both the simplest and many of the more complex contract assertions that could be written for `std::vector`. The algorithms and containers of the Standard Library seem simple on the surface yet have incredible complexity and depth and will require a wide range of powerful tools to be able to fully express checks of all the guarantees they make to clients.
- As mentioned above, most (or possibly all) Standard library implementations already make use of their own macro-based facilities to provide contract-checking. Supporting these macros as diagnostic tools is not something that these Libraries are going to stop doing any time soon and is even less likely when the core-language Contracts facility is not yet robust enough to provide complete feature parity with the macro-based solutions. Thus, putting contract assertions into the specification knowing that Libraries are from from ready, willing, or able to migrate to the core-language facility would be irresponsible of the Standard. For the foreseeable future, and possibly indefinitely, whether a library implementation chooses to use contract assertions, its own mechanisms, or nothing at all to check for correctness is a matter of implementation QoI, and implementations should not be forced to perform such checks in a proscribed manner.
- Even with no specification to use contract assertions, Standard Library vendors have already shown interest in adapting their current macro-based solutions to leveraging contract assertions when they are available, and users have opted into that functionality. This interest goes even beyond the checking of use of the Standard Library and to the checking of core-language undefined behavior such as pointer dereferences, as described in [P3191R0].

The C++ community should not fear that the Contracts facility will be unable to support the Standard Library but should be encouraged that the library groups have been planning for years

to use a facility very similar to what SG21 is proposing, and Standard Library implementations are poised to take advantage of the Contracts facility as much or as little as they wish, when the core-language support is available to them.

### 2.3 A More Fundamental Question

One last question comes up so often that it is the title of this paper.

> Question 3: Fundamental Q
>
> With so many potential safety features competing for the Committee's focus in the current environment, why should Contracts take priority, i.e., why Contracts?

Today's political environment puts every decision the Committee makes under a microscope regarding how a feature will not only perform but will be *perceived* by both the C++ community and those who are antagonistic toward the language and the C++ community. The Contracts facility proposed by SG21 is aimed squarely at this problem and intends to provide the foundation for how the problem can be solved.

One fundamental problem with addressing safety issues is that solutions often focus entirely on the *symptoms* of the problem rather than on the fundamental causes of the problem. CVE[3] issues that stem from taking advantage of undefined behavior are, almost tautologically, a result of software bugs that enable those problems to occur; no reasonable developer intends for a program to allow an attacker to take over a system, and any developer with malicious intent is not going to be stopped by any programming language change we could provide. Preventing undefined behavior from allowing exploitation treats the *symptom*, but a program that has a bug in it is the *actual problem*.

Contracts identify such bugs and provide a way to mitigate them at their source, including a wide variety of options for how contracts will be applied to programs throughout their entire lifecycle — from development to production systems.

Contracts can also be leveraged to improve correctness incrementally in a way that no other proposed safety-improving feature can do; the addition of a single contract assertion into an already-existing C++ application will help improve its correctness and stability. Each new assertion will add layers to this benefit. Importantly, this improvement does not depend on adopting a new coding style or refactoring old code. The Contracts facility proposed by SG21 has been built to have the fewest possible impediments to its adoption in any codebase of any quality.

That, fundamentally, is the problem any safety-focused feature must address. We have, as a community, built a huge number of features that allow people to write better code, but we have also sadly witnessed a community that resists adopting many of those improvements. Billions of lines of C++ have been written without many modern language and library features that would let that code be safer, and no engineering effort is going to rewrite it to more modern standards. Contracts, however, can help incrementally improve every single one of those programs. Even better, one of the biggest detriments to fixing legacy software is the lack of an understanding of what that software should do, let alone whether it is doing so correctly. By introducing contract assertions into a program, a developer can not only learn how the program behaves, but they can also document

[3] Common Vulnerability and Exposures, see https://www.cve.org

that behavior and capture, in an ongoing and continuous fashion, when those expectations fail. All of these improvements can be made with no semantic changes to the program as deployed (until actual bugs are found and then fixed) and no need to rewrite any existing code (until understanding has grown enough to be able to provide modern replacements that meet the same contractual requirements).

Contract checking is not the end for these programs but is a gateway to improved correctness that is accessible to any C++ program that can be compiled with a modern compiler. These improvements can then be used as a foothold to pave the way for actually using the safer, modern language that we all know everyone should be using.

## 3 Conclusion

I hope that, after reading this, you find the questions Herb posed to SG21 far from surprising, new takes on the problem space but well-worn issues to which developers have devoted years of thought, which have culminated in the proposal SG21 has put forth in [P2900R10].

What we cannot forget as we delve into every corner case and nuance to make this proposal as robust as possible is that we will begin to see the benefits as soon as a single contract assertion is added to a developer's library. With each additional assertion, these benefits will do nothing but compound. That compounding will not happen if we do not come to a consensus on the foundation proposed by [P2900R10] and adopt it into the C++ Standard.

## Bibliography


[P0788R0] Walter Brown, "Standard Library Specification in a Concepts and Contracts World", 2017
http://wg21.link/P0788R0

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

[P2834R1] Joshua Berne and John Lakos, "Semantic Stability Across Contract-Checking Build Modes", 2023
http://wg21.link/P2834R1

[P2900R10] Joshua Berne, Timur Doumler, and Andrzej Krzemieński, "Contracts for C++", 2024
http://wg21.link/P2900R10

[P2900R7] Joshua Berne, Timur Doumler, and Andrzej Krzemieński, "Contracts for C++", 2024
http://wg21.link/P2900R7

[P3191R0] Louis Dionne, Yeoul Na, and Konstantin Varlamov, "Feedback on the scalability of contract violation handlers in P2900", 2024
http://wg21.link/P3191R0

[P3212R0] Andrzej Krzemieński, "The contract of sort()", 2024
http://wg21.link/P3212R0

[P3290R2] Joshua Berne, Timur Doumler, and John Lakos, "Integrating Existing Assertions With Contracts", 2024
http://wg21.link/P3290R2

## Acknowledgements

Thanks to Herb Sutter for being willing to have his questions quoted (and I hope I have done so in sufficient context to avoid misunderstandings).

Thanks to all of SG21 for working diligently to help produce a Contracts proposal that is not only useful on its own but will be capable of evolving to one that is ubiquitously useful and, more importantly, that will provide a fundamental improvement in others' and my own daily work.

Thanks to Lori Hughes for reviewing this document and making it significantly more readable; any remaining grammatical issues are the author's fault and not hers.