Planning to Revisit the Lakos Rule

The Lakos Rule is Foundational for Contracts

Document #:	P2837R0
Date:	2023-05-10
Project:	Programming Language C++
Audience:	LEWG

Reply-to:	Alisdair Meredith <ameredith1@bloomberg.net> Harold Bott Jr. <hbott1@bloomberg.net>

Contents

1 Abstract **1**

2 Revision History **1**

2.1 2023 May mailing 1

3 Introducing the Lakos Rule **2**

4 The Lakos Rule Since 2011 **2**

5 The Lakos Rule for C++26 and Onward **2**

6 Conclusion **2**

7 References **2**

1 Abstract

C++11 introduced the `noexcept` specifier and the `noexcept` operator to retain the existing strong exception-safety guarantees promised by various types within the Standard Library when adding support for move semantics. With minimal opportunity to use a working implementation of such new language features, the Lakos Rule — described in [N3279] and [N3248] — was adopted as a simple, minimal set of design rules to follow until we gained sufficient practical experience. We now have over a decade of such experience, and our library designers have frequently suggested revising those guidelines.

With the pending arrival of Contracts in C++26, the need to revisit the Lakos Rule is clear. We make our case to defer such a review until that feature has landed.

2 Revision History

2.1 2023 May mailing

Initial draft of this paper.

3 Introducing the Lakos Rule

When C++11 introduced the `noexcept` specifier to support the strong exception-safety guarantees in the Standard Library in the presence of move semantics, we had no experience to guide our usage. The Lakos Rule was adopted as a simple set of design rules to follow — which would not impact client code written using the Standard Library — until we gained sufficient practical experience.

4 The Lakos Rule Since 2011

[N3279] introduced some essential vocabulary for talking about contracts on functions and the Lakos Rule to guide the Library Working Group in its adoption of this new feature. After more than a decade of practical experience, [P2861R0] provides an update to the original paper. It revisits the motivations, restates the guidelines, and illustrates up-to-date usage of these guidelines. [P2831R0] is an independent affirmation of the importance of the Lakos Rule in practice.

5 The Lakos Rule for C++26 and Onward

Now that we have over a decade of such experience, our library designers have frequently suggested revising these library guidelines (e.g., see [P1656R2]). Meanwhile, we anticipate that C++26 will include a Contracts facility being developed by SG21, incorporating *contract checking annotations* (CCAs) for narrow contracts.

Once the initial Contracts proposal is complete and ready to merge into the C++ working draft, we will need to address whether vendors have freedom to add CCAs to their library implementations in 16.4.6 [conforming]. We will also want a design guideline about how we would use CCAs in the library specification itself, much like Walter Brown's [P1369R0], which guides the modern specification for library clauses, with the trade-off between concepts in signatures and *Constraints:* clauses.

Meanwhile, each vendor can decide whether they adopt this rule in their implementation. We note that writing non-Lakos-Rule software on top of a library that adheres to the Lakos Rule is simple; writing software that adheres to the Lakos Rule on top of a library that does not is impossible. Since the Standard Library is fundamental to software written in C++, we have reinforced our belief that the Standard Library specification should continue to follow this fundamental design rule.

The current Lakos Rule has preserved the baseline behavior of the C++ Standard Library. Given the inevitable need to readdress this design space in the presence of Contracts, we should defer revisiting this rule until we have the complete context, expected in the next 12–18 months.

6 Conclusion

The original Lakos Rule was deliberately conservative so that programs built on top of the Standard Library would not be constrained against its use by external factors, such as use in a nonterminating production environment. Those principles hold as strongly today and will become only more relevant as C++26 looks to support a contract checking facility.

When Contracts are added to the Standard, we **expect** to codify something like the Lakos Rule that recognizes the incompatibility of narrow contracts and contract checking annotations. We should defer further discussion of such rules until either the Contracts facility lands in 12–18 months or we clearly see it will not land in this Standards cycle.

7 References

[N3248] A. Meredith, J. Lakos. 2011-02-28. noexcept Prevents Library Validation.
https://wg21.link/n3248

[N3279] A. Meredith, J. Lakos. 2011-03-25. Conservative use of noexcept in the Library.
https://wg21.link/n3279

[P1369R0] Walter E. Brown. 2018-11-25. Guidelines for Formulating Library Semantics Specifications.
https://wg21.link/p1369r0

[P1656R2] Agustín Bergé. 2020-02-14. "Throws: Nothing" should be noexcept.
https://wg21.link/p1656r2

[P2831R0] Timur Doumler, Ed Catmur. 2023-05-15. Functions having a narrow contract should not be `noexcept`.

https://wg21.link/p2831r0

[P2861R0] John Lakos. 2023-05-15. Narrow Contracts and `noexcept` are *Inherently* Incompatible.
https://wg21.link/p2861r0