`noexcept` policy for SD-9 (The Lakos Rule)

Timur Doumler (papers@timur.audio)
John Lakos (jlakos@bloomberg.net)

Document #:	P3155R0
Date:	2024-02-15
Project:	Programming Language C++
Audience:	LEWG

0 Motivation and context

A long-standing design principle in the C++ Standard Library has been that a function having a narrow contract — that is, a function specified to have *preconditions* — should not be declared `noexcept`, even if it is known to never throw an exception when called in-contract (without violating the preconditions). When we are convinced that a function having a narrow contract cannot throw when called in-contract, it should instead be specified as *Throws: nothing*. This design principle is also known as the *Lakos Rule*.

The Lakos Rule was first proposed in [N3248] and adopted with [N3279]. An updated version of the rule was codified into policy in [P0884R0]. See [O'Dwyer2018] for a more detailed summary.

More recently, [P1656R2] argued that the Lakos Rule should be abandoned as a design principle. According to this paper, functions that are known to never throw an exception for a valid combination of input values and accessible state should always be declared `noexcept`, regardless of whether they have a wide or a narrow contract. Further, [P2148R0] proposed adopting a set of design policies for the evolution of the C++ Standard Library that moves away from the Lakos Rule. While such a policy has not been formally adopted, LEWG recently moved away from adhering to the Lakos Rule, and instead considers whether a function having a narrow contract should be `noexcept` on a case-by-case basis.

We believe that we should consider retaining the Lakos Rule as a design policy for the C++ Standard Library. There are a number of important engineering reasons why the Lakos Rule is still useful and important today; abandoning it would inflict avoidable damage to the C++ language. To argue this case, we published two papers, [P2831R0] and [P2861R0], in the May 2023 committee mailing, to be presented at the Varna meeting in June 2023. However, neither paper was scheduled for discussion in LEWG at the Varna meeting, nor at the following meeting in Kona in November 2023.

Instead of considering our papers, LEWG has decided that we should first generally agree on a process for adopting design policies for the C++ Standard Library. Most recently, LEWG has agreed on a new framework to adopt such design policies (see [P2267R1]) and to publish them in a new standing document, SD-9. This framework allows us to finally decide on actual design policies, which are meant to be adopted via policy papers. [P2267R1] sets out a number of requirements that such policy papers must satisfy in order to be considered by LEWG. [P2831R0] and [P2861R0] have not been written with those requirements in mind, because they have been published before [P2267R1] was adopted. The purpose of the present paper is to satisfy the requirements in [P2267R1] and serve as an "envelope paper" for [P2831R0] and [P2861R0] to make them admissible for discussion in LEWG.

Most recently, another paper [P3005R0], rather than proposing a concrete `noexcept` policy, proposed a *process* how to agree on such a policy. [P3005R0] considers seven possible candidate `noexcept` policies, six design questions that would need to be answered to choose one of those policies, and lists 20 design principles that can help us find the correct answers to those questions by following the Principled-Design methodology. We believe that the process described in [P3005R0] is an excellent way to resolve the current disagreements on a `noexcept` policy in LEWG. However, since [P3005R0] proposes a process for decision-making rather than a single concrete policy, it also does not fit into the requirements for a policy paper in [P2267R1]. We therefore consider the present paper to be an "envelope paper" for [P3005R0] as well.

1 Rationale

The rationale for retaining the Lakos Rule as a design policy has been thoroughly explored in our previous papers, [P2831R0] and [P2861R0], and is further analysed in detail in [P3005R0]. We therefore refer the reader to those papers and references therein.

In short, the Lakos Rule is essential for implementing non-terminating recovery (an important requirement in some systems) and negative testing (the only known alternative to exception-based negative testing, death tests, is neither scalable nor portable). Independently from that, in API design, adhering to the Lakos Rule is a requirement for being able to widen the contract of a function without breaking backwards-compatibility. Finally, the Lakos Rule is foundational for Contracts, a new language feature targeting C++26 (see [P2900R5]) which is currently in design review. In addition, nearly 15 years of experience with `noexcept` have shown that applying it superfluously in the Library, beyond its original intended use, can be highly detrimental to writing safe and reliable software, while the claimed benefits of doing so generally do not hold up to scrutiny.

2 Prior art in the C++ Standard

The C++ Standard Library has been mostly following the Lakos Rule as a design principle since C++11, the standard that added `noexcept` to the language, but exceptions do exist. A detailed survey of the prior art in the C++ Standard will be provided in a future revision of this paper.

3 Status quo in the wider C++ community

Community surveys such as the JetBrains Developer Ecosystem survey, the Meeting C++ Community Survey, and the Standard C++ Foundation's Annual C++ Developer Survey consistently show that about half of C++ developers work on codebases where exceptions are either partially or completely banned from use; for the latter in particular, the `noexcept` policy is of little relevance.

Codebases that do use exceptions fall into different camps: many C++ libraries generously sprinkle `noexcept` all over their code, while codebases that portably use techniques like non-terminating recovery and negative testing adhere to the Lakos Rule in their own code (see [P2831R0] for case studies of several companies with such codebases). We expect the use of such techniques to become significantly more widespread with the adoption of Contracts for C++ ([P2900R5]). The three major implementations of the C++ Standard Library (GCC, Clang, and Microsoft) do not use the Lakos Rule and generally tighten *Throws: nothing* to `noexcept`, making them incompatible with these techniques; other implementations of the Standard Library, or parts of it, such as Bloomberg's BDE, follow the Lakos Rule.

4 Proposed policy

We propose to retain the policy we already have: the one described in [P0884R0], refining the previous rules in [N3279]. The policy consists of the following rules, which includes the Lakos Rule:

a) No library destructor should throw. They shall use the implicitly supplied (nonthrowing) exception specification.

b) If a library function has a wide contract (i.e., does not specify undefined behavior due to a precondition), it should be marked as unconditionally `noexcept` if it cannot throw; if it has a narrow contract, it should be specified as *Throws: Nothing* if it cannot throw when called in-contract.

c) If a library swap function, move-constructor, or move-assignment operator is conditionally-wide (i.e. can be proven to not throw by applying the `noexcept` operator) then it should be marked as conditionally `noexcept`.

d) If a library type has wrapping semantics to transparently provide the same behavior as the underlying type, then default constructor, copy constructor, and copy-assigment operator should be marked as conditionally `noexcept` matching the underlying exception specification.

e) No other function should use a conditional `noexcept` specification.

f) Library functions designed for compatibility with C code (such as the atomics facility) may be marked as unconditionally `noexcept`.

Note that this set of rules corresponds to Policy C in [P3005R0], which considers seven possible `noexcept` policies B-H in addition to the null policy A ("no policy"). We believe that all these policies should be considered, via the process proposed in [P3005R0], in order to be confident that we are adopting the correct one.

5 Rationale for adopting a policy

The placement of `noexcept` in the C++ Standard Library has been a hotly debated topic in recent years, taking up a considerable amount of Committee time. Adopting a formal policy on this matter would put the debate to rest and help maintain a coherent design for the C++ Standard Library going forward. Adopting the policy sooner rather than later would reduce the amount of newly introduced Standard Library functions incompatible with the Lakos Rule that would have to be retroactively fixed via Defect Reports.

6 Proposed wording

Append to "List of Standard Library Policies" section of SD-9 the items a) — f) enumerated in Section 4 above.

Acknowledgements

We wish to express our gratitude to the late Ed Catmur, co-author of [P2831R0], for his valuable contributions to this effort until his tragic death during a fell run on 31 December 2023.

References

[N3248] Alisdair Meredith and John Lakos. `noexcept` Prevents Library Validation. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3248.pdf, 2011-02-28.

[N3279] Alisdair Meredith and John Lakos. Conservative use of `noexcept` in the Library. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3279.pdf, 2011-03-25.

[O'Dwyer2018] Arthur O'Dwyer. The Lakos Rule. https://quuxplusone.github.io/blog/2018/04/25/the-lakos-rule/, 2018-04-25.

[P0884R0] Nicolai Josuttis. Extending the noexcept Policy, Rev0. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0884r0.pdf, 2018-02-10.

[P1656R2] Agustín Bergé. "Throws: Nothing" should be `noexcept`. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p1656r2.html, 2020-02-11.

[P2148R0] CJ Johnson and Bryce Adelstein Lelbach. Library Evolution Design Guidelines. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p2148r0.pdf, 2020-09-23.

[P2267R1] Inbal Levi, Ben Craig, and Fabio Fracassi. Library Evolution Policies — The rationale and process of setting a policy for the Standard Library. https://wg21.link/p2267r1, 2023-11-23.

[P2831R0] Timur Doumler and Ed Catmur. Functions having a narrow contract should not be `noexcept`. https://wg21.link/p2831r0, 2023-05-15.

[P2861R0] John Lakos. Narrow Contracts and `noexcept` Are Inherently Incompatible: The Lakos Rule. https://wg21.link/p2861r0, 2023-05-15.

[P2900R5] Joshua Berne, Timur Doumler, and Andrzej Krzemieński. Contracts for C++. https://wg21.link/p2900r5, 2024-02-15.

[P3005R0] John Lakos, Joshua Berne, Harold Bott, Mungo Gill, Alisdair Meredith, Bill Chapman, Mike Giroux, and Oleg Subbotin. Memorializing Principled-Design Policies for WG21. https://wg21.link/p3005r0, 2024-02-15.