Do not promise support for function syntax of operators

Document: P3392R0
Date: 2024-09-07
Main Author: Corentin Jabot (corentin.jabot@gmail.com)
Co-Author: Inbal Levi (sinbal2l@gmail.com)
Audience: Library Evolution Working Group (LEWG)

With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody. - Hyrum's Law

This paper is a follow-up to P1732 - initially proposed by CJ Johnson. We appreciate the original author's initiative and would like to continue the work as we consider it useful.

Motivation

It was agreed upon in Kona 2019 that the standard Library reserves the right to change the way operators are implemented (switching between member and nonmember overloading). We would like to ensure we convey this information to our users to avoid unexpected behavior in code relying on the current/future behavior that doesn't match the user's expectations.

Design

Following LEWG guidance, we made the following tweaks:

- Carve out an exception for `operator->`. This is because calling `->` will call `operator->` recursively on the return value of the called operator, a behavior that can only be opt-out from by calling `operator->` directly
- Carve out exceptions for `operator new`, `operator new[]` and `operator delete`, `operator delete[]`, as according to [expr.new]/p12 calling them directly can be used to allocate/deallocate storage.

Wording

Modify SD-8 by adding a new bullet under “Rights the Standard Library Reserves for Itself”:

Primarily, the standard reserves the right to:

[...]

- Assume that all 'operator functions' will not be called with function syntax (`a.operator@(b)` or `operator@(a, b)`), with the exception of `operator->`. This includes overloaded operators and user-defined literal functions. The exceptions are `operator->`, which the programmer is permitted to invoke via the member syntax `a.operator->()`, `operator new`, `operator new[]`, `operator delete` and `operator delete[]`.

[...]

Acknowledgment

Thanks to CJ Johnson for the initial paper, and Ben Craig for the Wording suggestion