Doc. No. P3611R0
Date: 2025-2-1
Audience: SG23, EWG, LEWG
Reply to: Bjarne Stroustrup (bjarne@stroustrup.com)

Dealing with pointer errors:
Separating static and dynamic checking

To address serious concerns about safety from important users of C++, we must do something urgently. That is, we need to include something significant in C++26. SG23 and EWG have voted in favor of profiles as a general framework, for specific profiles, and for library hardening. What we put in needs to be based on experience and work together in a principled way. This note outlines how.

We are following the subset-of-superset philosophy of profiles, the C++ core guidelines, the JSF++, and arguable C++'s evolution from C:

- First, we superset by adding a hardened standard library (dynamic checking)
- Next, we subset by restricting language use (static checking)

We are focusing on pointer errors because the data shows that's where the root of the majority of safety-related problems is, because we have experience dealing with that, and because SG23 and EWG have expressed support for that:

- The hardened standard library catch range and nullptr errors
- The static parts of the type and bounds profiles prohibit uses that cannot be run-time checked.

Users have choices that can ease gradual adoption:

- Using both together gives the strongest guarantees. This is recommended.
- Using only hardening allows users to preserve code that has not yet been made safe.
- Using only the static parts of the type and bounds profiles allow users to eliminate traditional pointer code while using a standard library that has not yet been hardened.

We are responding to serious comments from library and core language implementers and from users. Please consult the latest version of documents for specifics:

- Gabriel Dos Reis: C++ Profiles: The Framework. P3589.
- Herb Sutter: Core safety profiles for C++26. D3081.

- Konstantin Varlamov, Louis Dionne: Standard library hardening. P3471.
- Ville Voutilainen, Jonathan Wakely, and Gabriel Dos Reis: Contracts and profiles: what can we reasonably ship in C++26. D3608.

The profiles framework provides a standard way of requesting these profiles in code. In addition, implementers can offer default enabling where needed.

Obviously, these initial profiles don't address all problems. We need to and can provide profiles addressing further problems (e.g., dangling pointers: the lifetime profile), but this will get us started in C++26. Gradual improvement is one important thing that the profiles framework offers.

Note that in this particular case, the static and dynamic parts of the implementation are carefully separated to simplify implementation and to rely on existing experience.