Doc. No. P3651R0
Date: 2025-03-06
Audience: All
Author: Bjarne Stroustrup
Reply to: bjarne@stroustrup.com

I sent this note to reflectors. It was written for WG21 members. Someone leaked it to the Web where it caused discussions without context. I therefore feel that I have to make it public (unchanged). I am of course disappointed that WG21 couldn't focus on this. At least they didn't do nothing. C++26 will have a hardened standard library and the community now have a good design with implementations to works from.

Note to the C++ standards committee members

Bjarne Stroustrup (bjarne@stroustrup.com)

Executive summary

- Profiles are essential for the future of C++
- Profiles will not break your existing code
- Profiles will not prevent your favorite new feature
- Profiles are part of a long tradition of C++ evolution
- The alternative is incompatible, ad hoc restrictions

The rest is details. Don't get lost in the details.

Support initial profiles for C++26. Support more profiles later.

Why this note

This is clearly not a traditional technical note proposing a new language or library feature. It is a call to urgent action partly in response to unprecedented, serious attacks on C++. I think WG21 needs to do something significant and be seen to do it. Profiles is a framework that can do that. For technical discussions, see my papers in the Reference section and especially the papers recursively referenced from there. Profiles is not a conventional language feature but a framework for offering guarantees by imposing restrictions on existing features.

As I have said before, this is also an opportunity because type safety and resource safety (including memory safety) have been keys aim of C++ from the very start.

I feel strongly about this. Please don't be fooled by my relatively calm language.

Profiles are essential for the future of C++

C++, usually under the misleading label C/C++, is under attack from the US and EU regulatory bodies, the managements of several powerful commercial organizations, and just about everybody promoting a different language for anything C++ is used for or might be used for (e.g., see references). In particular, the US government demands a plan for achieving memory safety by 2026 and several important organizations will follow that. Without the support from the committee this will imply

- Disuse of C++ for many important new projects
- Decreased funding for future development of C++ language, libraries, and tools

This is unprecedented and ignores C++ strengths. This has already started.

We – the C++ standards committee – have an obligation to help the C++ community at large to address these challenges. We do not have much time. Fortunately, we have a plan (backed by repeated SG23 and EWG votes) and significant practical experience, but if we waste our time on inessential details rather than approving the Profiles framework and a couple of significant profiles for C++26, it will be too late and will adversely affect our users.

I have heard said that the US government mandate and the related "safety concerns" will affect only security sensitive and network-facing components. That's not the way things work in many organizations and tool chains. If a language is labeled "dangerous" or "unsafe" (the way that seems to be popular for "C/C++"), it will (where possible) be kept out of potentially sensitive code and not included in tool chains unless necessary. Not using C++ will be seen as the safe bet.

The demands for memory safety are not unreasonable, in fact, I consider them too feeble for the long term, so responding to the demands is in the interest of C++.

Profiles will not break your existing code

Profiles are opt-in and also offer strong compatibility:

- No code will have a different meaning with or without a profile.
- Unless you enable a profile, no code without UB will behave differently.
- Eventually, most modern C++ code will run with its current meaning with most profiles enabled.

It is impossible (not just hard) to get both complete compatibility with every old style of coding together with safety for any reasonable definition of "safety." Much old-style code cannot be statically proven safe (for some suitable definition of "safe") or run-time checked. Such code will not be accepted under key profiles (addressing the safety requirements from powerful organizations).

Note that the safety requirements insist on guarantees (verification) rather than just best efforts with annotations and tools. The initial profiles that we can get ready for C++26 won't deliver as strong guarantees as we would like, but our plans for C++26 attack the major sources of errors/insecurities as part of a framework that will get us stronger guarantees in the future. And, of course, profiles don't stop people from applying added checks where they have the tools for such.

Profiles will not prevent your favorite new feature

Importantly, this applies to proposed/future features and libraries also. Profiles do not constrain the evolution of C++. Features that will help the community significantly tend not to be significantly affected by Profiles. However, features that cannot be verified as safe tend to be out of the tradition of C++ improvements as well as banned by the government safety requirements.

Profiles is a framework that restricts what part of the language can be used if a profile is explicitly requested. It is not an attempt to impose a novel and alien design and programming style on all C++ programmers or to force everyone to use a single tool. The evolution of C++ can continue unhampered. Personally, I hope for static reflection, functional-programming style pattern matching, contracts, and better concurrency support. As ever, there are significant constraints on what new features can be accepted (performance, compatibility, generality, ease of use, etc.) but those are independent of profiles.

The main constraint on new features is the same as for existing features, if some use cannot be proven safe for some profile's notion of safety, the feature cannot be used together with that profile (or the profile must simply be suppressed for essential uses of the "unsafe" feature).

Profiles are part of a long tradition of C++ evolution

C++ was initially derived from C by adding type-checking guarantees and libraries to C. With the exception of requiring declaration and type checking of function arguments (a restriction), it left C alone. That was a major success and eventually C followed suit, but for years complaints about type safety being restrictive were common. Profiles are opt-in, so there should be no compatibility problems. The heavy-handed government and corporate approaches will of course lead to loud complaints, but the best WG21 can do is to mitigate that. That's what we are doing with the flexible Profiles framework. We are not trying to force a single idea of safety or a single programming style onto everybody.

From roughly day #1 there have been requests for support for some sort of subsetting of C++, but these ideas failed to get traction because the features people wanted eliminated (e.g., pointer arithmetic) were essential for implementing the features that people wanted instead (e.g., range-checked vectors). Instead of simple subsetting, a general approach, known as "subset of superset" has been used for decades.

- First, superset by adding a ideal domain abstractions
- Next, subset by restricting language use of features deemed undesirable

This was the approach of the early evolution from C to C++ (e.g., a vector class allowed us to avoid many range errors). This was the basis for Lockheed-Martin and the US Airforce's flight software coding guidelines (JSF++ of which I was a major author), and of the C++ Core Guidelines (which has been deployed for many years with various degrees of library and static analysis support). The standard library hardening is in the same tradition, strengthening the guarantees offered by the standard library.

For initial profiles that translates into:

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

Complete type safety and resource safety were the Ideals from the start of C++. For example, see "The Design and Evolution of C++" and/or any of my papers and talks relating to safety and reliability. Profiles will give us that – and more – where requested.

The alternative is incompatible, ad hoc restrictions

If the C++ community does not adopt profiles, the likely outcome is a fragmented C++ community and ecosystem:

- Developers will rely on ad hoc tools and a variety of proprietary and non-proprietary safety mechanisms that lack interoperability. These tools, libraries, and language features may very well not fit into the desired future evolution of C++.
- Organizations will impose incompatible restrictions, making it harder to share libraries, code, and developer expertise.
- C++ loses ground to languages perceived as safer and more modern yet lacking some of C++'s strength and its ecosystem. Interoperability among an open set of languages could become a significant burden on the use of those languages.

Profiles provide a unifying framework that allows the C++ community to address safety challenges without compromising C++'s core strengths.

Providing a common framework and common facilities is an essential part of the task of a language standards committee.

Conclusion

Back the profiles proposed for C++26. That way we all win. It’s in the best tradition of C++ responding to feedback from real-world use.

The sky isn’t falling, but unless we act now and get C++ onto a track supporting a flexible framework of profiles (supporting various forms of safety), we risk a painful decline.

References

- A note containing links to current Profile documents “Dealing with pointer errors: Separating static and dynamic checking”: https://isocpp.org/files/papers/P3611R0.pdf .
- An article placing profiles in a historical context “21st Century C++”: https://cacm.acm.org/blogcacm/21st-century-c/ .
- A call for standard memory safety “It Is Time to Standardize Principles and Practices for Software Memory Safety”**:** https://cacm.acm.org/opinion/it-is-time-to-standardize-principles-and-practices-for-software-memory-safety/ and https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-996.pdf.
- A large group of security agencies call for memory safety “The Case for Memory Safe Roadmaps - Why Both C-Suite Executives and Technical Experts Need to Take Memory Safe Coding Seriously”: https://media.defense.gov/2023/Dec/06/2003352724/-1/-1/0/THE-CASE-FOR-MEMORY-SAFE-ROADMAPS-TLP-CLEAR.PDF .
- Unsafe languages as a risk to national security “Product Security Bad Practices“: https://www.cisa.gov/resources-tools/resources/product-security-bad-practices .
- EU “Cyber Resilience Act” https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act