Re: RO 2-056 6.11.2 [basic.contract.eval] Make Contracts Reliably Non-Ignorable

Authors: Durlea Andrei (AndreiDurlea)

1. Scope

This paper aims to provide feedback to D3911R1 [1] from the perspective of a first-year student at Politehnica University of Bucharest, Faculty of Computer Science and Automatic Control, with a keen interest in programming language study.

For this feedback I have considered:

- Possibilities enabled by the proposed syntax and how they can be mentioned without paper scope creep;
- Reasoning for the addition of non-ignorable contracts and how that can be expressed further.

2. Suggestion: Integrating future possibilities into the syntax proposals

It is a very clear design choice that the syntax mentioned in Section 5.2 is forward compatible with the upcoming P3400 architecture (for C++29 onwards). However, in my opinion, Section 5.1 also leaves the door open for future "symbol-suffix" syntax making a first-time appearance in C++. Should this bridge be crossed by the syntax proposed in your paper, future additions (C++29 onwards) could utilize similar syntax patterns. Here are some examples:

- `pre?` , `pre!?` etc. - a shorthand symbol addition for each of the four enum values, assuming they all reach a significant usage;
- `array[i]?!` or `array[i]!!` - a Haskell-like way of introducing safe indexing [2] to C++; the behaviour could be similar to contracts, defined by a build-time choice or explicitly in-code, potentially calling a global function;
- `expr?` - syntactic sugar for `std::expected` and `std::optional` .

While including suggestions such as this would definitely introduce scope creep, the design space introduced by such syntax could be elegantly mentioned as follows:

Suggested addition (@ Section 5.1 end)

> ++++

> We also believe that pioneering a syntax like `pre!` and `pre!!` would open the door to a design space where symbol-suffix shorthands become viable in future features from C++29 onwards.

3. Suggestion: Stating more risks introduced by not having a way to reliably write non-ignorable contracts

Although Section 1 clearly states the main usability risk, there are others that could be leveraged as compelling arguments to ensure D3911R1 is adopted. For example:

- Leaving contracts to be defined primarily at build-time increases complexity for downstream users. This breaks the determinism of the language, creating a scenario where semantic meaning changes based on environment flags—something rarely allowed in the C++ ecosystem.
- It contradicts the modern C++ design shift from 'hints' to guarantees. Much like `constexpr` eventually required the addition of `consteval` to satisfy user needs for compile-time guarantees, contracts should launch with both the 'hint' and the guarantee to avoid repeating this evolutionary delay.
- Without clear deterministic support, many libraries may not consider using the contracts feature at all, fearing ABI or behavior fragmentation.

Suggested addition (@ Section 1 1st paragraph end)

> ++++

> We consider that contracts should be launched with support for fully deterministic code, as modern C++ design is shifting from 'hints' to guarantees (drawing a parallel to the evolution of `constexpr` into `consteval`). If released with lack of deterministic reliability,

> contracts risk being under-utilized.

4. Conclusion

As D3911R1 is already a mature paper, it is my belief that EWG will view it favorably. My feedback concerns including the additional risks the contracts feature faces should a reliable way to write non-ignorable contracts not be included in the initial release. Furthermore, I suggest acknowledging the future design space regarding symbol-suffix syntax that the paper basically pioneers.

5. References

[1] D3911R1 — Darius Neațu, Andrei Alexandrescu, Lucian Radu Teodorescu, Radu Nichita, "RO 2-056 6.11.2 [basic.contract.eval] Make Contracts Reliably Non-Ignorable", WG21.

[2] Safe indexing in Haskell - Haskell documentation.