P2680R0 2022-10-15 Reply-To: gdr@microsoft.com
Audience: SG21

Contracts for C++: Prioritizing Safety

Gabriel Dos Reis
Microsoft

1 INTRODUCTION

A programming language is a set of responses to challenges of its time. What are the contemporary problems that the next version of C++ needs to address? Undoubtedly, there are many. Safety should be listed among the very top. By this, I mean ***safety by default***. While it can be argued that it is possible to build – with enough care and expertise - C++ programs that are both type and resource safe, the challenge here is to provide mechanisms that sustain scalable sound programming techniques promoting safety. Such techniques are to be practiced by millions of C++ programmers, and not require diplomas of advanced studies in language subtleties. It should take extraordinary steps to write a program that violates memory, type, and resource safety. The notion of contracts is a tool that can help the C++ community get there. But only if we can design them well enough to prioritize safety, and to underpin robust code analysis tools (including static code analysis, runtime checks, etc.)

At its core, the notion of contracts is simple. And we must keep it that simple at the language support level, so that they are recognizable by ordinary programmers – not just something we call "contracts". A contract (G. Dos Reis 2016) is generally a pre-condition on a function, or a post-condition on a function. A pre-condition is a **predicate** that expresses the expectations of a function on its arguments. Similarly, a post-condition is a **predicate** that expresses the guarantees of a function immediately following a successful execution. It is customary to extend these aspects to include assertions inside the definition of a function: again, a **predicate** that expresses an invariant at a given program execution point.

2 SIDE EFFECTS

It is easy to design a syntactic sugar template that can expand to unstructured, arbitrary code that we would call contracts. However, to enable robust code analysis tools at scale, and scalable practice of contracts, it is necessary to put structures in place. In particular, it is necessary to take the notion of predicate more seriously, something more than an unrestricted arbitrary code block with possibly type bool. **I suggest that we make each of a pre-condition and a post-condition, a self-contained expression** (their free variables being function parameters and constants), **and side-effect free when seen from the outside of each of their cone of evaluation**. That means for example that I should not be allowed, in a pre-condition, to phone home, chat with aunties and uncles, write a log of the conversation to disk, and in the post-condition share the log with friends. If those activities are to happen as part of the proper execution of a function, then they belong in the function body proper – as we do today. Contracts are summaries of expectations of and guarantees expressed as predicates. This is not just a matter of "well, if you don't like it, don't do it". To provide safety by default, we need to go beyond the usual "live, and let live". We add programming language features to promote certain programming styles. Here, we want

to promote safe programming by default using contracts. Not supporting side effects visible outside the cone of evaluation of a contract that does not limit expressivity of the language. We can already express those side effects today, easily in the body of the functions; and we do so routinely.

3 Semantics Model

It has been suggested that, in terms of code generation, a pre-condition is a prolog to a function, and that a post-condition is an epilog. While that analogy holds at the lower implementation level, it is important – from language design perspective – that not all prologs are pre-conditions, and not all epilogs are post-conditions. At the day-to-day programming level, a pre-condition is the expression of the expectations of a function. It is fundamental that such an expectation can be evaluated (symbolically if possible) by the compiler and code analysis tools, and code generated as appropriate.

I suggest to abandon the approach that a pre-condition is just a prolog, and that a post-condition is just an epilog. Those are implementation details that do not help us design a language support in the language that empowers scalable code analysis such as those based on SAL (Gabriel Dos Reis 2014), as deployed in-the-field for over two decades.

Does that mean no side effects in contracts? The answer is: No! What I am suggesting is to take a page from the constexpr semantics model (Smith 2013). We can do as much side effects as we want inside the cone of constexpr evaluation, as long as those side effects are not visible from the outside, when evaluation is finished. We wouldn't require that you call only constexpr functions, but we would require that you call only functions whose definitions are reachable from the point of the contract and whose side effects stay inside the cone of evaluation of that contract. We know this model works, and we have had successful experience with it over a decade of modern C++ programming. It is easier to start from a sound solid logical ground and expand from there (as proven by the constexpr semantics model and technology) than to try to issues patches to an unprincipled and unstructured arbitrary code evaluation model. To have the hope of pretending to bring increased safety to C++, it is imperative to operate from logically sound grounds.

4 References

G. Dos Reis, J. D. Garcia, J. Lakos, A. Meredith, N. Myers, B. Stroustrup. 2016. *A Contract Design.* July 11. https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0380r1.pdf.

Gabriel Dos Reis, Bjarne Stroustrup. 2010. *General Constant Expressions for System Programming Languages.* March 22. https://www.stroustrup.com/sac10-constexpr.pdf.

Gabriel Dos Reis, Shuvendu Lahiri, Francesco Logozzo, Thomas Ball, Jared Parsons. 2014. *Contracts for C++: What Are the Choices?* November 23. https://open-std.org/JTC1/SC22/WG21/docs/papers/2014/n4319.pdf.

Smith, Richard. 2013. *Relaxing constraints on constexpr functions.* April 18. https://open-std.org/JTC1/SC22/WG21/docs/papers/2013/n3652.html.