Contracts for C++: Prioritizing Safety

Presentation Slides of P2680R0

Gabriel Dos Reis

Software development perspective (P0287)

- “They provide basic mitigation measures for early containment of undesired program behavior”
- “Contracts are requirements that an operation puts on its arguments for successful completion and set of guarantees it provides upon successful completion”
- “structured assert() integrated into the language”
  - Basis for principled program analysis and tooling
- Not:
  - “Contracts are not a general error reporting mechanism, nor are they substitute for testing frameworks.”

Why Prioritize Safety?

Securing existing code and viability of C++ in increasingly unfavorable/hostile environment

Securing existing code and viability of C++ in increasingly unfavorable/hostile environment

- Safety issues in software written in C and C++ are increasingly blamed for why some critical cyberphysical systems are vulnerable
  - Active recommendations by various regulatory bodies and others (NIST, NSA, etc.) to move away from C++
- While the headings start with “memory safety”, technical analysis shows that the entire type system is involved
  - See P2687: Design Alternatives for Type-and-Resource Safe C++

- Upgrade needed to the language to enable **safety by default**
  - Contracts have a key role to play
  - Not just syntactic sugar for things we can easily express today

Design Principles of P2680

Requirements for Contracts

- The **evaluation of contract predicates shall be free of undefined behavior**
  - Key requirement
- They provide basic mitigation framework, they should not themselves be sources of vulnerabilities

- Several ways to get there:
  1. Rewrite the abstract machine specification specifically for contract evaluation
  2. Restrict the set of permitted in contract predicates
  3. ???

Restricted expressions in contract predicates

- Previous efforts (e.g. P0542) choose to specify **side effect** in contract predicates as leading to **undefined behavior semantics**
  - See analyses of intricacies in previous C++20-era contracts, numerous EWG discussions, and papers

- P2680 restricts expressions in contract predicate in order to remove the undefined behavior aspect.
  - Any design that permits undefined behavior in contract predicate evaluation renders the feature unreliable/useless to help bring safety to C++

Design principle of P2680

- Start from a sound logical ground
- The evaluation of a contract predicate can perform side effects between the start and the end of the evaluation of that predicate expression, but the set of such side-effects are not visible from outside the code of evaluation of that predicate.

- Gradually expand without compromising the key requirement of no UB in contract predicate evaluation

Specifics of P2680

- What can we do *without* new annotation?
  - Take a page from constexpr model
    - Don’t confuse with constexpr itself
- A function body can
  - side effects its parameters,
  - local variables,
  - call functions that have same properties
  - The body of a function called in a contract predicate must be available in that same TU
- Starting point to help us provide safety by default in C++

Suggestions/amendments since P2680

- Make it clear that “usable in a contract predicate” is a property of a function
  - (notionally a bit like ‘noexcept’)

- Add ability to annotate functions usable in contract predicates, so their implementations can be separated from their interface
  - However, the implementation shall still be checked for conformance to the restriction so as no to introduce UB
- Add a “`relaxed`” annotation for functions that need side-effects

```
int fizz(string s) [[ pre relaxed: call_mothership(s), not s.empty() ]];
```

Impacts on std library uses

- Q: Which library functions can I use in contract predicates
- A: Any function that we deem appropriate
  - E.g. vec.empty(), str.size(), v.begin(),
  - Etc.

- Q: But those std implementations use techniques that violate the constraints in P2680
- A: Yes, they do, because of lack of contract facilities integrated into the language