Make contracts reliable by default

Document #:	P3640R0
Date:	2025-03-17
Project:	Programming Language C++
Audience:	EWG, SG21, SG23

Reply-to:	Jonas Persson <jonas.persson@iar.com>

Contents

1 The safety problem with Observe and Ignore

This should be a reasonable way to use Contracts:

```
int ddref(int** p) pre(p) pre(*p) { return **p; }
```

But in [P2900R13] this is not safe. A call outside of a contract in *observe* or *ignore* mode is not an error.

This is the correct code:

```
int ddref(int** p)
    pre(p)
    pre(p && *p)
{
    if(!p) throw std::logic_error("precondition failed");
    if(!*p) throw std::logic_error("precondition failed");
    return **p;
}
```

It's too easy to assume that what a contract promises is true.
We should not trick users into writing code with undefined behavior.

Observe and *ignore* have their place, but they are fundamentally unsafe. It should be obvious from the code when it is safe to assume that the contract holds. Having to read related build scripts does not count as obvious. Hiding the semantic selection in the build system, or in some separately declared label definition, will not clearly enough tell what is being applied.

1.1 Labels

We might eventually get functionality to make it possible to attach labels to contracts that only allow checked modes [P3400R0]. This may sound good, but it might not be enough.

```
int ddref(int** p)
    pre <safe>(p)
    pre <safe>(*p)
{
    return **p;
}
```

We don't yet have a finished labels design, but it is reasonable to expect labels to be numerous and user-defined in different places.
It might not be clear from the local context which are reliable and which aren't.

1.2 A safe keyword quick_enf

A keyword to separate safe and unsafe contracts is probably better:

```
int ddref(int** p) pre safe(p) pre safe(*p) { return **p; }
```

But this another example of bad defaults. The short and obvious syntax should be the safe one, and the one where the contract cannot be trusted should be the explicit one.

1.3 Safe by default

Limit undecorated contracts to *enforce* and *quick_enforce*. Add a keyword to the other contracts, to make it clear that they might not be checked.

```
pre maybe(stmt)
post maybe(stmt)
contract_assert maybe(stmt)
```

```
int ddref(int** p)
    pre maybe(p)
    pre maybe(p && *p)
{
    if(!p) throw std::logic_error("precondition failed");
    if(!*p) throw std::logic_error("precondition failed");
    return **p;
}
```

Our original attempt now works as expected and always gives us a valid pointer to dereference.

```
int ddref(int** p) pre(p) pre(*p) { return **p; }
```

For *enforce* and *quick_enforce* semantics both basic and annotated contracts behave the same. Same thing if a label evaluates to any enforced semantics.

The *ignore* and *observe* modes only apply to the `maybe` annotated contracts. In its basic form it is always some variant of enforced.

The initial contract proposal only have a single global evaluation semantic, and a semantic must be chosen to apply when the global one is not applicable.
In for *ignore* mode, contracts with a `maybe` annotation will be ignored, but an undecorated will be evaluated with the enforced equivalent *quick_enforce*.
In *observe* mode, `maybe` annotated contracts will be observed while undecorated will be *enforced*.

When we in the future get labels additional evaluation semantics, the safe contracts might be enforced in different ways, but they can never be ignored.

If the proposed changes in this document are implemented, an undecorated contract can always be trusted to have established what its precondition promises.

Other possible names are “pre optional”, “pre unsafe”, “pre?” …

1.4 The ODR issue

While this enforcement control does not solve the ODR issue, it mitigates it to the level that we will not get our non-optional contracts unexpectedly removed by the linker. They will always be enforced, even if we may lose or gain some logging.

1.5 Acknowledgement

Thanks to David Sankel for bringing this issue up in a way that inspired this paper.

1.6 Summary

We propose to require the Contracts feature in its basic form to always be enforced. A new annotation is proposed for contracts, to allow them to be ignored depending on compilation flags or labels.

This is a very late realization, but I’m afraid we are going to regret it if we ship it like this.

Differentiating between guaranteed contracts and suggested contracts will make a huge difference.

It is not yet too late to fix this for C++26.

2 References

[P2900R13] Joshua Berne, Timur Doumler, Andrzej Krzemieński. 2025-01-13. Contracts for C++. https://wg21.link/p2900r13

[P3400R0] Joshua Berne. 2025-01-09. Specifying Contract Assertion Properties with Labels. https://wg21.link/p3400r0