Document Number: P1448R0

Date: 2019-1-20

Audience: EWG

Reply-to: Nathan Burgers nburgers@bloomberg.net

Simplifying Mixed Contract Modes

Abstract

The working draft for C++20 (*N4791*) provides conditional support for program translation where the contract build level need not be the same for all translation units. The current wording requires that the contract of a function be checked according to the build level in each translation unit where it is odr-used or defined. This can lead to surprising results.

Contents

The Background

According to the wording in (*N4791*) [decl.attr.contract.check]/3, which states:

> A translation may be performed with one of the following build levels: off, default, or audit. A translation with build level set to off performs no checking for any contract. A translation with build level set to default performs checking for default contracts. A translation with build level set to audit performs checking for default and audit contracts. If no build level is explicitly selected, the build level is default. The mechanism for selecting the build level is implementation-defined. The translation of a program consisting of translation units where the build level is not the same in all translation units is conditionally-supported. There should be no programmatic way of setting, modifying, or querying the build level of a translation unit.

Throughout this document, we will refer to a program translation where all translation units have the same build level as a *homogeneous build*, and a program translation where translation units need not all have the same build level as a *heterogeneous build*.

Where translations with heterogeneous build levels are supported, each invocation of a function having contract preconditions and postconditions must check those conditions according to the build level of the translation unit that contains the call site. Additionally, translation units that contain a definition of such a function must check its contract conditions according to their own build modes, respectively.

The Problem

As an example of how this may lead to undesirable situations, suppose one is compiling a program that consists of two modular translation units (according to *P1103R2*), *TU1* and *TU2*, which are defined the following way:

```
// Translation Unit 1
export module foo;
export int f(int x) [[expects audit: x > 0]];
module :private;
int f(int x) [[expects audit: x > 0]] { return x; }

// Translation Unit 2
import foo;
int main() { return f(-1); }
```

Notice that the definition of the function `f` will only be translated in *TU1*, but that `f` is odr-used in *TU2*. The translation of the function definition in *TU1* is required to check the contract according to the build level for its translation, and the use of `f` in *TU2* is required to check the contract according to the build level of its own translation.

In this example, whether `main` will invoke the violation handler depends on the build levels of **both** *TU1* and *TU2*. The set of possible build level configurations and their effect on whether the violation handler is invoked are enumerated below:

TU1 Build Level	TU2 Build Level	Contract Violation Handler is Invoked
off	off	no
off	default	no
off	audit	yes
default	off	no
default	default	no
default	audit	yes
audit	off	yes
audit	default	yes
audit	audit	yes

Whether or not the violation handler is invoked when this program is run is determined by the combination of the build modes of each of its translation units. Reasoning about this behavior is unnecessarily complex.

The Proposed Solution

The proposed solution is to define a single build level that is to be used for the contract conditions of a given function, regardless of the build level of translation units that invoke it. This build level will be the build level of the translation unit that contains the function's definition.

For an inline function `D`, the working draft already specifies,

(*N4791*) [basic.def.odr]/12.6:

> if D invokes a function with a precondition, or is a function that contains an assertion or has a contract condition (9.11.4), it is implementation-defined under which conditions all definitions of D shall be translated using the same build level and violation continuation mode; . . .

Which is compatible with this proposed solution.

Note that this is compatible with the semantics of a homogeneous build. In a homogeneous build, the build level of the translation unit containing the function's definition is the same as the build level of any translation unit where it is used.

The Downside

If the proposal is adopted, implementations have reduced freedom to decide how to lay out contract checking code in a heterogeneous build.

References

[N4791] Richard Smith, Working Draft, Standard for Programming Language C++
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4791.pdf

[P1103R2] Richard Smith, Merging Modules
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1103r2.pdf