Make predicate exceptions propagate by default

Timur Doumler (papers@timur.audio)

Document #: P3626R0
Date: 2025-02-11
Project: Programming Language C++
Audience: EWG

Abstract

One of the concerns over the Contracts for C++ proposal [P2900R13] raised in [P3573R0] is that if a checked contract predicate exits via an exception, that exception is caught and forwarded to the contract-violation handler, which leads to overhead. The suggestion was made that such an exception should instead unconditionally propagate up the stack. This paper provides the necessary wording changes for such a modification.

The proposed wording is relative to D2900R14, the draft version that is in CWG and LWG wording review at the time of writing.

Modify [basic.contract.eval] as follows:

A *contract violation* occurs when

— B is `false`,

— ~~the evaluation of the predicate exits via an exception, or~~

— the evaluation of the predicate is performed in a context that is manifestly constant-evaluated ([expr.const]) and the predicate is not a core constant expression.

[*Note:* If B is true, no contract violation occurs and control flow continues normally after the point of evaluation of the contract assertion. ~~The evaluation of the predicate can fail to produce a value without causing a contract violation, for example, by calling `longjmp` ([csetjmp.syn]) or terminating the program.~~ — *end note*]

If the evaluation of the predicate of a function contract assertion ([dcl.contract.func]) exits via an exception, the behavior is as if the function body exits via that same exception. [*Note*: A *function-try-block* ([except.pre]) is the function body when present and thus does not have an opportunity to catch the exception. If the function has a non-throwing exception specification, the function `std::terminate` is invoked ([except.terminate]). — *end note*] If the evaluation of the predicate of an *assertion-statement* ([stmt.contract.assert]) exits via an exception, the search for a handler continues from the execution of that statement.

[*Note:* There are other circumstances in which the evaluation of the predicate can fail to produce a value without causing a contract violation, for example, by calling `longjmp` ([csetjmp.syn]) or terminating the program. — *end note*]

[...]

~~If the contract violation occurred because the evaluation of the predicate exited via an exception, the contract-violation handler is invoked from within an active implicit handler for that exception ([except.handle]). If the contract-violation handler returns normally and the evaluation semantic is observe, that implicit handler is no longer considered active.~~

~~[*Note:* The exception can be inspected or rethrown within the contract-violation handler. — *end note*]~~

If the contract-violation handler returns normally and the evaluation semantic is enforce, the program is contract-terminated~~; if violation occurred as the result of an uncaught exception from the evaluation of the predicate, the implicit handler remains active when contract termination occurs~~.

Modify [except.terminate] as follows:

Some errors in a program cannot be recovered from, such as when an exception is not handled or a `std::thread` object is destroyed while its thread function is still executing. In such cases, the function `std::terminate` ([exception.terminate]) is invoked. [*Note:* These situations are:

— ...

— when evaluating the predicate of a function contract assertion ([dcl.contract.func]) or a contract-violation handler ([basic.contract.handler]) invoked from evaluating a function contract assertion on a function with a non-throwing exception specification exits via an exception, or

— ...

— *end note*]

Modify [contracts.syn] as follows:

```
~~enum class detection_mode : unspecified {~~
~~  predicate_false = 1,~~
~~  evaluation_exception = 2~~
~~};~~
```

[...]

```
class contract_violation {
  // no user-accessible constructor
public:
  contract_violation(const contract_violation&) = delete;
  contract_violation& operator=(const contract_violation&) = delete;

  /* see below */  contract_violation();

  const char* comment() const noexcept;
  ~~detection_mode detection_mode() const noexcept;~~
  ~~exception_ptr evaluation_exception() const noexcept;~~
  bool is_terminating() const noexcept;
  assertion_kind kind() const noexcept;
  source_location location() const noexcept;
```

```
    evaluation_semantic semantic() const noexcept;
};
```

Remove [support.contract.enum.detection] entirely.

Modify [support.contract.violation] as follows:

~~`contracts::detection_mode detection_mode() const noexcept;`~~

~~*Returns:* The enumerator value corresponding to the manner in which the contract violation was identified.~~

[...]

~~`exception_ptr evaluation_exception() const noexcept;`~~

~~*Returns:* If the contract violation occurred because the evaluation of the predicate exited via an exception, an `exception_ptr` object that refers to that exception or a copy of that exception; otherwise, a null `exception_ptr` object.~~

Bibliography

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

[P3573R0] Michael Hava, J. Daniel García Sanchez, Ran Regev, Gabriel Dos Reis, John Spicer, Bjarne Stroustrup, J.C. van Winkel, and Daveed Vandevoorde. Contract concerns. https://wg21.link/p3573r0, 2025-01-12.