Document number: P3842R2
Audience: LWG

Ville Voutilainen
2026-03-26

A conservative fix for constexpr uncaught_exceptions() and current_exception()

Abstract

The papers P3818 and P3820 explain the background of the problem we have making uncaught_exceptions() and current_exception(). In short, having those function constexpr is a breaking change in some cases.

Due to timing and lateness thereof, it's deemed prudent to just roll back adding constexpr to those functions, and figure out a solution in C++29 for allowing those functionalities in constant expression evaluation. That was the outcome of an LEWG discussion in a September 2025 teleconference.

Wording

In [exception.syn], remove the constexpr decl-specifiers from these functions:

```
~~constexpr~~ int uncaught_exceptions() noexcept;
using exception_ptr = unspecified ;
constexpr exception_ptr current-exception() noexcept; // exposition only
~~constexpr~~ exception_ptr current_exception() noexcept;
...
template<class T> [[noreturn]] ~~constexpr~~ void throw_with_nested(T&& t);
template<class E> ~~constexpr~~ void rethrow_if_nested(const E& e);
```

Before [uncaught.exceptions]/1, remove the constexpr decl-specifier:

```
~~constexpr~~ int uncaught_exceptions() noexcept;
```

Before [propagation]/9, add a new definition above, and remove the constexpr decl-specifier:

```
constexpr exception_ptr current-exception() noexcept; // exposition only
~~constexpr~~ exception_ptr current_exception() noexcept;
```

In [propagation]/12, edit as follows:

```
template<class E> constexpr exception_ptr make_exception_ptr(E e) noexcept;
```

```
Effects: Creates an exception_ptr object that refers to a copy of e, as if:
  try {
    throw e;
  } catch(...) {
    return ~~current_exception~~current-exception();
  }
```

Strike the note In [propagation]/13:

~~[Note 5: This function is provided for convenience and efficiency reasons. — end note]~~

In [except.nested], remove all constexprs:

```
namespace std {
  class nested_exception {
  public:
    ~~constexpr~~ nested_exception() noexcept;
    ~~constexpr~~ nested_exception(const nested_exception&) noexcept = default;
    ~~constexpr~~ nested_exception& operator=(const nested_exception&) noexcept = default;
    ~~constexpr~~ virtual ~nested_exception() = default;
    // access functions
    [[noreturn]] ~~constexpr~~ void rethrow_nested() const;
    ~~constexpr~~ exception_ptr nested_ptr() const noexcept;
  };
  template<class T> [[noreturn]] ~~constexpr~~ void throw_with_nested(T&& t);
  template<class E> ~~constexpr~~ void rethrow_if_nested(const E& e);
}
```

Before [except.nested]/3, remove the constexpr decl-specifier:

```
~~constexpr~~ nested_exception() noexcept;
```

Before [except.nested]/4, remove the constexpr decl-specifier:

```
[[noreturn]] ~~constexpr~~ void rethrow_nested() const;
```

Before [except.nested]/5, remove the constexpr decl-specifier:

```
~~constexpr~~ exception_ptr nested_ptr() const noexcept;
```

Before [except.nested]/6, remove the constexpr decl-specifier:

```
template<class T> [[noreturn]] ~~constexpr~~ void throw_with_nested(T&& t);
```

Before [except.nested]/9, remove the constexpr decl-specifier:

```
template<class E> ~~constexpr~~ void rethrow_if_nested(const E& e);
```