C++ coroutine gotcha: Falling off the end of a function-level catch

devblogs.microsoft.com/oldnewthing/20220718-00

July 18, 2022

Raymond Chen

Allowing execution to flow off the end of a coroutine is equivalent to performing a `co_return` with no operand, assuming the coroutine completes with no value. Otherwise, the behavior is undefined.

This is the same rule as with regular functions, just with the letters “co” in front.

`void f1()`
`{`
`  DoSomething();`
`  // implicit "return"`
`}`
	`simple_task<void> f1co()`
	`{`
	`  co_await DoSomething();`
	`  // implicit "co_return"`
	`}`

`int f2()`
`{`
`  DoSomething();`
`  // illegal fall-of-the-end`
`}`
	`simple_task<int> f2co()`
	`{`
	`  co_await DoSomething();`
	`  // illegal fall-of-the-end`
	`}`

Unfortunately, many compilers (as of this writing) aren’t consistent in diagnosing this type of programming error.

```
simple_task<int> f2co()
{
    co_await DoSomething();
    // illegal fall-of-the-end
}
```

```
// gcc 11.3 -std=c++20 -Wall
(no errors or warnings)

// clang 14.0.0 -std=c++20 -Wall
warning: non-void coroutine does not return a value

// msvc 19.31 -std:c++20 -W4
warning C4033: 'f2co' must return a value
note: Flowing off the end of a coroutine results in undefined behavior when promise type
'std::coroutine_traits::promise_type' does not declare 'return_void'
```

In this case, clang and msvc notice that you forgot to return a value, but gcc doesn't notice.

If we tweak the function slightly, we get different results:

```
simple_task<int> f3co()
{
    if (Maybe()) co_return 1;
    // illegal fall-of-the-end
}

// gcc 11.3 -std=c++20 -Wall
(no errors or warnings)

// clang 14.0.0 -std=c++20 -Wall
warning: non-void coroutine does not return a value in all control paths

// msvc 19.31 -std:c++20 -W4
(no errors or warnings)
```

Adding a `co_return` on one branch of an `if` statement is enough to fool msvc; it doesn't notice that there's still a code path that fails to `co_return` something.

And then there's this wrinkle:

```
simple_task<int> f4co() try
{
    co_return 1;
}
catch (...)
{
    // illegal fall-of-the-end
}
```

```
// gcc 11.3 -std=c++20 -Wall
(no errors or warnings)

// clang 14.0.0 -std=c++20 -Wall
warning: non-void coroutine does not return a value in all control paths

// msvc 19.31 -std:c++20 -W4
(no errors or warnings)
```

The `catch` block fails to `co_return` anything, which makes it an illegal fall-off-the-end, but gcc and msvc fail to detect it.

This particular failure is easy to miss if you use the WIL exception handling macros like `CATCH_LOG`:

```
simple_task<int> f4co() try
{
    co_return 1;
}
CATCH_LOG(); // invisible fall-off-the-end
```

The `CATCH_LOG` macro catches all exceptions, logs them through the WIL infrastructure, and then falls off the end of the function. It is intended to be used only in cases where falling off the end is allowed (namely, function returning `void` or coroutine completing with `void`). If you use it in a coroutine that has a completion value, then you will just fall off the end, and if you're unlucky, the error will go undiagnosed, and you're off in undefined territory.

Bonus chatter: But really, what happens when you fall off the end of the coroutine without `co_return`ing a value? As I noted, it's technically undefined behavior, but in practice what happens is that the promise's `return_value` method is never called before reaching `final_suspend`. What happens next depends on how the promise is implemented.

In our `simple_task`, it means that the promise state remains `empty`, and then when you try to `co_await` the `simple_task`, the `get_value` method hits an assertion failure and then forcibly `std::terminate`s the program.

For PPL tasks, the promise implementation just returns a default-constructed result; For hat-types, that means a null pointer. For C++/WinRT asynchronous operations, the promise implementation returns an empty result: For value types, you get a default-constructed value type; for reference types, you get a null pointer.

This can lead to a good amount of head-scratching when you `co_await` the task or asynchronous operation and get an empty result / null pointer even though you go back to the code and see that at no point does it ever `co_return nullptr;`.

From what you can tell, the compiler appears to have lost its mind, but really, you're the one who went crazy. You just didn't realize it.

Raymond Chen

Follow