```
    struct Z<double> { };          // Error, outer::Z or outer::inner::Z?
}
```

Reopening namespaces can reopen nested `inline` ones

Another subtlety specific to **`inline`** namespaces is related to reopening namespaces. Consider a namespace `outer` that declares a nested namespace `outer::m` and an **`inline`** namespace `inner` that, in turn, declares a nested namespace `outer:inner::m`. In this case, subsequent attempts to reopen namespace `m` cause an ambiguity error:

```
namespace outer
{
    namespace m { }      // opens and closes ::outer::m

    inline namespace inner
    {
        namespace n { }  // opens and closes ::outer::inner::n
        namespace m { }  // opens and closes ::outer::inner::m
    }

    namespace n          // OK, reopens ::outer::inner::n
    {
        struct S { };    // defines ::outer::inner::n::S
    }

    namespace m          // Error, namespace m is ambiguous.
    {
        struct T { };    // with clang defines ::outer::m::T
    }
}

static_assert(std::is_same<outer::n::S, outer::inner::n::S>::value, "");
```

In the code snippet above, no issue occurs with reopening `outer::inner::n` and no issue would have occurred with reopening `outer::m` but for the `inner` namespaces having been declared **`inline`**. When a new namespace declaration is encountered, a lookup determines if a matching namespace having that name appears anywhere in the ***`inline`*** *namespace set* of the current namespace. If the namespace is ambiguous, as is the case with `m` in the example above, one can get the surprising error shown.[3] If a matching namespace is found

[3]Note that reopening already declared namespaces, such as `m` and `n` in the `inner` and `outer` example, is handled incorrectly on several popular platforms. Clang, for example, performs a name lookup when encountering a new namespace declaration and give preference to the outermost namespace found, causing the last declaration of `m` to reopen `::outer::m` instead of being ambiguous. GCC, prior to 8.1 (c. 2018), does not perform name lookup and will place *any* nested namespace declarations directly within their enclosing namespace. This defect causes the last declaration of `m` to reopen `::outer::m` instead of `::outer::inner::m` and the last declaration of `n` to open a new namespace, `::outer::n`, instead of reopening `::outer::inner::n`.