When writing a specialization,
be careful about its location;
or to make it compile
will be such a trial
as to kindle its self-immolation.

Only one namespace can contain any given `inline` namespace

Unlike conventional `using` directives, which can be used to generate arbitrary many-to-many relationships between different namespaces, `inline` namespaces can be used only to contribute names to the sequence of enclosing namespaces up to the first non`inline` one. In cases in which the names from a namespace are desired in multiple other namespaces, the classical `using` directive must be used, with the subtle differences between the two modes properly addressed.

As an example, the C++14 Standard Library provides a hierarchy of nested `inline` namespaces for **literals** of different sorts within namespace `std`.

- `std::literals::complex_literals`
- `std::literals::chrono_literals`
- `std::literals::string_literals`
- `std::literals::string_view_literals`

These namespaces can be imported to a **local scope** in one shot via a `using std::literals` or instead, more selectively, by `using` the nested namespaces directly. This separation of the types used with **user-defined literals**, which are all in namespace `std`, from the user-defined literals that can be used to create those types led to some frustration; those who had a `using namespace std;` could reasonably have expected to get the user-defined literals associated with their `std` types. However, the types in the nested namespace `std::chrono` did *not* meet this expectation.[11]

Eventually *both* solutions for incorporating literal namespaces, `inline` from `std::literals` and non`inline` from `std::chrono`, were pressed into service when, in C++17, a `using namespace literals::chrono_literals;` was added to the `std::chrono` namespace. The Standard does not, however, benefit in any objective way from any of these namespaces being `inline` since the artifacts in the `literals` namespace neither depend on ADL nor are templates in need of user-defined specializations; hence, having all non`inline` namespaces with appropriate `using declarations` would have been functionally indistinguishable from the bifurcated approach taken.

[11] CWG issue 2278; **hinnant17**