Disambiguating Nested-Requirements

Motivation

A requires-expression currently has the following grammar:

\begin{verbatim}
requires-expression :
    requires requirement-parameter-list_opt requirement-body
requirement-parameter-list :
    ( parameter-declaration-clause_opt )
requirement-body :
    { requirement-seq }
requirement-seq :
    requirement
    requirement-seq requirement
requirement :
    simple-requirement
    type-requirement
    compound-requirement
    nested-requirement
simple-requirement :
    expression ;
type-requirement :
    typename nested-name-specifier_opt type-name ;
compound-requirement :
    { expression } noexcept_opt return-type-requirement_opt ;
return-type-requirement :
    -> type-constraint
nested-requirement :
    requires constraint-expression ;
\end{verbatim}

with no specific additional disambiguation rules.
Of note here, is that expression in a simple-requirement can currently be a requires-expression itself. So, in the example:

```cpp
template<typename T>
concept C = requires {
    requires (T v) { v.foo(); }; // (1)
    requires T{};             // (2)
}
```

requirement (1) is a simple-requirement and requirement (2) is a nested-requirement. However, (1) is almost certainly unintentional: It has essentially no effect. (In particular, whether v.foo() is valid has no effect on whether C<T> evaluates to true.) More likely the programmer intended it to be a nested-requirement with that same expression.

Besides the ease of confusion, this ambiguity also makes it more difficult to make typename prefixes optional in a requirement-parameter-list. E.g.:

```cpp
template<typename T>
concept K = requires (typename T::Type X) { // (3)
    X.next();
}
```

It would be nice to make typename optional in line (3) (like it is, e.g., in lambda parameter lists), but requiring that would impose a more difficult look-ahead scheme than other contexts in which typename has been made optional.

**Proposal**

Let's introduce a rule that disallows simple-requirements that start with a requires keyword, and let's make typename optional in the parameters of requires-expressions. That achieves two goals:

1. reduce the likelihood of an unintended simple-requirement instead of a nested requirement
2. allow a more concise declaration for certain requires-expression parameters

Note that this does not materially hamper the programmer who really wants to include a requires-expression as a simple-requirement: Such an expression can still be parenthesized, or the same effect can be obtained by surrounding it with braces (making it a compound-requirement). We also do not think that this will break existing code since GCC does not today implement the disambiguation needed to implement the rules of the current working paper (N4842). In that sense, we're proposing existing practice.

The first part of this proposal makes example (1) above ill-formed. That example does not require that v.foo() be well-formed; instead, it is a requirement that the whole
requires-expression be well-formed, which it always is. It’s far-fetched for that to ever make sense, so it seems reasonable to ban such barely-ever-intentional, confusing and error-prone requirements.

In other words, at the cost of a somewhat late specification change, we avoid utterly confusing semantics by banning such simple-requirements.

The second part — which is made technically possible through the first part — makes the language more consistent, by allowing the omission of typename in a requirement-parameter, which is consistent with functions and lambdas, and in line with the evolutionary direction of “Down with typename!” (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0634r3.html).

**Wording**

Add a new paragraph after [expr.prim.req.simple]/1:

> A requirement that starts with a requires token is never interpreted as a simple-requirement. [Note: This simplifies distinguishing between a simple-requirement and a nested-requirement. —end note]

Change [temp.res] sub-bullet (5.2.5) as follows:

> — parameter-declaration in a lambda-declarator or requirement-parameter-list, unless that parameter-declaration appears in a default argument, or

**Acknowledgments**

Richard Smith, Andrew Sutton, and Ville Voutilainen contributed materially to this paper — thanks!