JTC1/SC22/WG14 N3144
Title: **Underspecified Declarations**
Author: **Martin Uecker**
Date: **2023-06-22**

Suggested alternative text changes to address DE-49 and GB-50:

6.7 Declarations
Constraints

~~**5 In an underspecified declaration all declared identifiers that do not have a prior declaration shall be ordinary identifiers.**~~

Semantics

12 A declaration such that the declaration specifiers contain no type specifier or that is declared with constexpr is said to be underspecified. If such a declaration is not a definition, if it declares no or more than one ordinary identifier, if the declared identifier already has a declaration in the same scope, ~~*or*~~ if the declared entity is not an object, **or if anywhere within the sequence of tokens making up the declaration identifiers that are not ordinary are declared,** the behavior is ~~**undefined**~~ **implementation-defined.XXX)**

XXX) It is recommended that implementations that accept such declarations follow the semantics of the corresponding feature in ISO/IEC 14882.

6.7.9

NOTE 1 ~~**Such**~~ **A declaration that also defines a structure or union type has implementation defined behavior** ~~**violates a constraint**~~**. Here, the identifier x which is not ordinary but in the name space of the structure type is declared.**

```
auto p = (struct { int x; } *)0;
```

Even a forward declaration of a structure tag

```
struct s;
auto p = (struct s { int x; } *)0;
```

would not change that situation. A direct use of the structure definition as the type specifier ensures ~~**the validity**~~ **portability of the declaration. The following is also implementation-defined:**

```
auto alignas (struct s *) x = 0;
```

6.7.1

22 EXAMPLE 6 Because declarations using constexpr are underspecified, the following is implementation-defined because tokens within the declaration declare s which is not an ordinary identifier:

```
constexpr typeof (struct s *) x = 0;
```