Slay Some Earthly Demon: Storage-Class Specifiers with Compound Literals

Number: n3871
Author: Martin Uecker

Using a storage-class specifier other than constexpr, static, register, or thread_local with an compound literal is specified to be undefined behavior, i.e. for extern, auto, and typedef. But typedef and extern are already ruled out by the constraint that the compound literal can be rewritten into the form

SC typeof(T) ID = { IL };

and this has to be valid, but this is not valid for typedef and extern because of the presence of the initializer. When auto is used for type inference, then this is also not syntactically valid because then the right-hand side is required to be an assignment expression. This leaves the outdated use of auto as a storage-class specifiers as the only remaining case where there is (non-ghost) undefined behavior. It does not seem reasonable to leave this obscure case as an extension point, hence it is suggested to add a constraint to forbid this. For clarity, the constraint explicitly lists all allowed specifiers.

Wording (N3854)

6.5.3.6 Compound literals

Constraints

3 The only storage-class specifiers that shall occur are constexpr, static, register, and thread_local.

Semantics
…

7 ~~**If the storage-class specifiers are absent or contain constexpr, static, register, or thread_local**~~ **T**he behavior is as if the object were declared and initialized in the corresponding scope with ~~**these**~~ **the given** storage-class specifiers~~**; if another storage-class specifier is present, the behavior is undefined**~~. If the storage-class specifier constexpr is present, the initializer is evaluated at translation time. Otherwise, if the storage duration is automatic, the initializer is evaluated at each evaluation of the compound literal; if the storage duration is static or thread the initializer is (as if) evaluated once prior to program startup