Doc.no.: P1250R0
Date: 2018-10-04
Reply-to: Titus Winters (titus@google.com), Ashley Hedberg (ahedberg@google.com), Eric Fiselier (eric@efcs.ca)
Audience: LEWG

Extension by inspecting members of User Defined Types?

Context

In P0921 and SD-8, we enumerate a list of changes that the standard library generally considers "in bounds" when making changes between two language versions. However, in P0919 we are extending the functionality of the standard library in a way that it outside of those bounds: examining how we interpret user-provided types (in this case, hashers) by inspecting them for certain members.

Specifically, P0919 takes user-provided hashers and looks for a member tag type `transparent_key_equal`. In the presence of such a type, additional constraints are placed on the `Eq` template parameter to `std::unordered_map` (and similar unordered associative containers). Additionally, we may start calling `operator()(T)` for `T` that doesn't match the original key; this is the goal in enabling heterogeneous lookup. In theory, there could be existing user provided hashers that define these things but assign different semantics. For those users, this will be a breaking change.

Anecdotally, nobody cares, nor does anyone believe this is likely. However, it does bring up a question: Do we wish to extend SD-8 as we identify more cases like this? Or should we attempt to keep that list fixed and design extensions / future modifications in terms of those operations?

We could imagine instead performing this extension by introducing a new name in `std`, perhaps `std::transparent_hash<Container, HeterogenousType>` and encouraging user-specialization of that template to opt-in to heterogenous lookup for a particular (container/key/value/eq/hash, heterogenous type) mapping. Whenever we want to offer customization points like this, we'd want to ensure that there is no way two users would want to define it for the same type. The coroutines customization points are potentially problematic here, as two users may want a coroutine containing no user-defined types.

Proposal

We do not propose modifying P0919. We do propose that we discuss our policy and behavior going forward. If we believe that this type of extension is safe, we should amend SD-8 accordingly, although describing the conditions under which we would make such a change to how we interpret user-defined types may be difficult.

On the other hand, if we believe this type of extension should be avoided, we should clearly document that, and we should commit to finding extension mechanisms that are more in keeping with the behavior described in SD-8.

Suggested Polls

Do we believe extension points via inspecting members of user-defined types is safe?

Do we believe we should avoid extension points that inspect members of user-defined types?