The Use and I: Transitivity of Module Uses and its Impact

Lydia Duncan (Chapel Implementer, Cray Inc.)

Early in Chapel's history it was decided that a 'use' of a module would be transitive, meaning that using a module (B) which used another module (A) would bring A's symbols into scope in addition to the symbols defined by B. Such a nested series of 'use's is referred to as a 'use' chain. This decision has directly affected the new namespace control features added in the last year - limited uses via the 'except' and 'only' keywords, the ability to rename symbols imported via a 'use' statement, and private symbols – and sparked a desire for control over the transitivity of individual 'use' statements. This talk will explore the impact on these new features, both in implementation and from a user's perspective.

The 1.13 release of Chapel will allow the user to limit the symbols brought into scope via a 'use' statement, either by specifying which symbols to ignore via an 'except' list or by specifying the symbol to include via an 'only' list. Because uses are transitive, these lists must affect all the deeper 'use's in a particular 'use' chain, or the conditions they attempt to guarantee would not be honored. For instance, if the aforementioned 'use' of module B specified an 'except' list, it would be incorrect to find symbols from A with names present in the 'except' list. This adds some complexity to the analysis of whether a symbol is visible in a given scope, and this complexity was by necessity included in part of our current resolution implementation. Further complexity is added when one considers 'use' chains which contain more than one such list.

The ability to rename a symbol in an 'only' list will also be added in the 1.13 release, allowing the user to avoid naming conflicts between 'use's and to prevent the shadowing of symbols at an outer scope to the 'use' statement. Similarly to plain 'only' lists, renaming a symbol during a 'use' must affect all deeper 'use's in a chain, so that if the symbol 'foo' was renamed to 'bar' and the module A also defined a symbol 'foo', this symbol should be likewise not be accessible in the scope of the 'use' by the name of 'foo'. The complexity added to our implementation was very similar to that of the 'only' and 'except' lists.

Introduced in Chapel 1.12, the ability to define a symbol as public or private allows the user to control the visibility of symbols that are only intended for access within their defining module. Those 'use's of a module which defines private symbols must not allow access to these symbols from scopes where they would not otherwise be visible. With transitivity involved, the visibility of a private symbol can change along a 'use' chain – specifically, modules that are children of the module where the private symbol was defined should be able to access this symbol, and if a 'use' would bring it into a nearer scope the privacy should not change the presence of the symbol.

With such a variety of ways for a 'use' to be impacted by transitivity, it seems natural to give the user control over when these considerations must be taken into account. By allowing the user to specify whether a 'use' is transitive or private to that module, we will also provide them with a greater understanding and awareness of the places from which their symbols are visible.