Containers and pointers

Andrew Koenig

As the Working Paper stands today, it is not possible for a portable program to have, say, an object of class `std::set<int*>` unless all its elements point to elements of the same array. This is because the `std::set` template replies on `<` being appropriately defined on the values placed in a `set` and `<` is undefined on pointers except when both pointers being compared point to elements of the same array.

The obvious solution, namely decreeing that `<` must be a (strong) partial ordering over pointers even if the precise ordering is undefined, fails for the same reason `<` is not fully defined to begin with: on machines with segmented architectures it is desirable for performance reasons to implement `<` by comparing only the low-order parts of the pointers. This works whenever it is possible to guarantee that every array will fit entirely with a single segment; such guarantees are often possible on such implementations.

Here is a sketch of a solution to the problem, offered in the hope that others will care enough about the problem to fill in the details. The idea is to extend the definition of `less` (defined in [lib.comparisons]) by partial specialization so that for any type `T`, `std::less<T*>` is guaranteed to yield a strong total order relation even if the built-in `<` operator does not do so. For completeness, the same should be done for `std::le`, `std::gt`, and `std::ge`. A quick check shows that `std::set` already relies on `std::less` as its default comparison instead of the built-in `<` operator; if the other containers do as well, that should solve the problem.