Doc No: N3301-11-0071
Date: 2011-09-04
Author: Pablo Halpern
Intel, Corp.

phalpern@halpernwightsoftware.com

Defect Report: Terminology for Container Element Requirements

Contents

Document Conventions

All section names and numbers are relative to the April 2011 FDIS, N3290 as modified by the proposed resolution for LWG 2033.

Existing working paper text is indented and shown in dark blue. Edits to the working paper are shown with ~~red strikeouts for deleted text~~ and green underlining for inserted text within the indented blue original text.

Comments and rationale mixed in with the proposed wording appears as shaded text.

Requests for LWG opinions and guidance appear with light (yellow) shading. It is expected that changes resulting from such guidance will be minor and will not delay acceptance of this proposal in the same meeting at which it is presented.

National Body comments and issues

This defect report describes an omission in N3173, which resolved comment US 115 to the July, 2010 FCD. The proposed wording in this paper interacts with the resolution of LWG 2033. The wording here assumes that the resolution of LWG 2033 has been applied.

Description of Defect

Adoption of N3173 corrected the misuse of the terms CopyConstructible and MoveConstructible and the phrase "constructible with *args*" in the containers section of the FCD. Unfortunately, the paper missed a few incorrect uses of CopyConstructible and failed to

correct similar misuses of the term DefaultConstructible. These errors persist now in the IS and should be corrected by a TC.

The nature of the terminology misuse is that elements of a container are never constructed directly within the container (except in the case of `array`), but rather are constructed by calling the `construct` member function of the container's allocator. The allocator is not required to call the element's constructor with exactly the list of arguments supplied to `construct`. The `scoped_allocator_adaptor` is an example of an allocator that modifies the `construct` argument list before calling the element's constructor. Thus, saying that a container's `value_type` is DefaultConstructible is neither necessary nor sufficient for specifying the requirements on that type. The proposed wording below defines a precise replacement for the term DefaultConstructible in the containers section just as N3173 did for CopyConstructible and MoveConstructible. The wording also replaces any incorrect uses of DefaultConstructible with the new term and corrects any remaining incorrect uses of CopyConstructible.

Proposed Resolution (formal wording)

Add a new bullet to 23.2.1 [container.requirements.general], paragraph 13 and add a `destroy` requirement to each of the existing bullets as follows:

Given a container type X having an `allocator_type` of A and a `value_type` of T and given an lvalue `m` of type A, a pointer `p` of type `T*`, a value v of type T, or a value `rv` of type rvalue-of-T, the following terms are defined. (If X is not allocator-aware, the terms below are defined as if A were `std::allocator<T>`.):

— T is ***DefaultInsertable*** *into X* means that the following expressions are well formed:

```
allocator_traits<A>::contruct(m, p);
allocator_traits<A>::destroy(m, p);
```

— T is ***CopyInsertable*** *into X* means that the following expressions are~~is~~ well-formed:

```
allocator_traits<A>::contruct(m, p, v);
allocator_traits<A>::destroy(m, p);
```

— T is ***MoveInsertable*** *into X* means that the following expressions are~~is~~ well-formed:

```
allocator_traits<A>::contruct(m, p, rv);
allocator_traits<A>::destroy(m, p);
```

— T is ***EmplaceConstructible*** *into X from args*, for zero or more arguments, *args*, means that the following expressions are~~is~~ well-formed:

```
allocator_traits<A>::contruct(m, p, args);
allocator_traits<A>::destroy(m, p);
```

[*Note:* A container calls `allocator_traits<A>::contruct(m, p, args)` to construct an element at `p` using *args*. The default of `contruct` in `std::allocator` will call `::new((void*) p) T(`*args*`)` but specialized allocators may choose a different definition. – *end note*]

There are no incorrect uses of `DefaultConstructible`, `CopyConstructible`, `MoveConstructible`, or *constructible from* in section 23.2, including Tables 96 through Tables 103.

In sections 23.3.3 [deque] through 23.5 [unord], make the following text replacements:

Original text, in FDIS	**Replacement text**
`T` shall be `DefaultConstructible`	`T` shall be `DefaultInsertable` into `*this`
`key_type` shall be `CopyConstructible`	`key_type` shall be `CopyInsertable` into `*this`
`mapped_type` shall be `DefaultConstructible`	`mapped_type` shall be `DefaultInsertable` into `*this`
`mapped_type` shall be `CopyConstructible`	`mapped_type` shall be `CopyInsertable` into `*this`
`mapped_type` shall be `MoveConstructible`	`mapped_type` shall be `MoveInsertable` into `*this`
`Key` shall be `CopyConstructible`	`Key` shall be `CopyInsertable` into `*this`
`value_type` is constructible from	`value_type` is `EmplaceConstructible` into `*this` from

Notes to the editor: The above are carefully selected phrases that can be used for global search-and-replace within the specified sections without accidentally making changes to correct uses of `DefaultConstructible` et. al.. Please ensure that the resolution of 2033 is applied before applying these changes, otherwise, the use of `DefaultConstructible` in that resolution will be incorrect.

Separable issue: In 23.4.4.2 map constructor `map(first, last)`, has an incomplete *requires* clause. It describes what the requirement is *if* `*first` is `pair<key_type,mapped_type>` but doesn't say what requirement is otherwise. What should the requirement be? Does `*this` have to be a pair, or merely pair-like? What are the actual requirements on `first->first` and `first->second`? I believe that the requirement should be fairly broad but

complex: the iterator's value type must have members `first` and `second`, where `key_type` is EmplaceConstructible into `*this` from `first->first` and `mapped_type` is EmplaceConstructible into *`this` from `first->second`. However, it might be sufficient and simplest to say that `value_type` is EmplaceConstructible into `*this` from `*first`. The same issue applies to the `insert` member 23.4.4.4 [map.modifiers]. In the latter case, the range insert version should probably be separated from the other two and each one's requirements precisely described (some use of forward<> might be needed). It is also confusing that the requirements for `insert` describes things that are *not* required. Same issue for multimap (23.4.5.3).

Separable issue: `operator[](key_type&&)` is missing a requirement that `key_type` be MoveInsertable into `*this`.

References

N3290: Final Draft International Standard: Programming Languages C++, 2011-04-11

N3102: ISO/IEC FCD 14882, C++0X, National Body Comments

N3173: Terminology for constructing container elements

LWG 2033: Preconditions of `reserve`, `shrink_to_fit`, and `resize` functions