Doc. No.: X3J16/96-0098
WG21/N0916
Date: May 28, 1996
Project: Programming Language C++
Reply To: J. Lawrence Podmolik
STR
podmolik@str.com

Clause 23 (Containers Library) Issues List
Revision 8

Revision History

Revision 1 - January 31, 1995. Distributed in pre-Austin mailing.

Revision 2 - March 2, 1995. Distributed at the Austin meeting.

Revision 3 - May 28, 1995. Distributed in pre-Monterey mailing.

Notes: some discussion was condensed or elided for closed issues to keep the list to a reasonable size. Also, some compound issues were split into several separate issues and some problems with issue numbering were corrected.

Revision 4 - July 11, 1995. Updated and distributed at the Monterey meeting.

Includes several issues generated from the first round of X3J16 public review comments, as well as issues resulting from editorial boxes in the April 28, 1995 version of the WP.

Revision 5 - July 31, 1995. Distributed in post-Monterey mailing.

Updated to reflect issues closed at the Monterey meeting, Also includes several new issues resulting from the X3J16 public review comments and from discussions at Monterey.

Revision 6 - October 29, 1995. Distributed at the Tokyo meeting.

Includes issues that remained open following the Monterey meeting, plus a significant number of new issues. For brevity, this revision lists the full text only of ongoing and new issues; issues closed up to and including the Monterey meeting are summarized below.

Note: Working Paper references in this revision are to the pre-Tokyo draft dated 26 September 1995.

Revision 7 - November 30, 1995. Distributed in the post-Tokyo mailing.

Updated to reflect issues closed at the Tokyo meeting. Also includes new issues raised (but not addressed) at the Tokyo meeting and any issues identified since that meeting.

Revision 8 - May 28, 1996. Distributed in the pre-Stockholm mailing.

Introduction

This document is a summary of the issues identified in Clause 23. For each issue the status, a short description, and pointers to relevant

reflector messages and papers are given. This evolving document will serve as a basis of discussion and historical for Containers issues and as a foundation of proposals for resolving specific issues.

Summary of Open Issues

23-028 Clean up empty sections in Clause 23
23-041 Possible solutions for map::insert()

23-043 Fix container ambiguities when T == size_type
23-044 Inconsistent insert() return types for assoc. containers
23-045 Remove <stdexcept> from <bitset> synopsis
23-046 Clean up bitset element access methods
23-047 Clarify complexity for deque::erase()
23-048 Improve description of list::sort()
23-049 Clarify complexity for vector::insert(p,i,j)
23-050 Add additional constructors to Container requirements
23-051 Fix description of list::unique()
23-052 Fix description of list::merge()
23-053 vector<bool>::const_reference should be bool
23-054 Define vector<bool>::reference::operator==()
23-055 Fix return type of map::operator[]()
23-056 Remove const version of map::operator[]()
23-057 Need semantics for associative containers
23-058 Fix reverse iterator typedef arguments
23-059 Wrong reverse iterator type for associative containers
23-060 Fix postcondition for (&a)->~X() in requirements table
23-061 Reorganize Clause 23 sections
23-062 Remove() algorithm doesn’t work on map/multimap

Summary of Closed Issues

23-001 Add convenience functions to STL containers
23-002 Should some STL members return an iterator?
23-003 Nomenclature problems in STL classes
23-004 Should STL classes have fixed comparator semantics?
23-005 Should some STL members return a size_type?
23-006 Naming inconsistencies in bits<T>
23-007 Adding vector<bool>::flip that toggles all bits
23-008 Add a nested reference class to bits<T>
23-009 Add "default value" arg to map/multimap constructors
23-010 Requirements for type T in template containers
23-011 Bitset inserters/extractors need updating
23-012 Templatize bits members for basic_string
23-013 Return values from library class member functions
23-014 Add hash tables to standard library
23-015 Reference counted strings and begin()/end()
23-016 Adding constructors to nested reference types
23-017 Add clear() to all containers
23-018 Add additional pop() functions to containers
23-019 Make Allocator argument in containers const refs
23-020 Change container adapter interfaces
23-021 Modify complexity of swap() due to allocators
23-022 Add typedef, member to retrieve allocator type
23-023 Specify container iterators as opaque types
23-024 Fix copy constructors w.r.t. allocators
23-025 Remove bitset exposition implementation
23-026 Update vector<bool> with partial specialization
23-027 Make vector<bool> bit ref swap a static member
23-029 Fix vector constructor signatures in description
23-030 Update descriptions of deque operations

23-031 Specialize swap() algorithm for containers
23-032 Non-const top() missing in priority_queue?
23-033 Clean up resize() effects for deque, list and vector
23-034 Reverse iterator types for list
23-035 Correct argument list to vector<bool>::insert
23-036 Need semantics for at() member deque/vector
23-037 Semantics for a.back() in sequence requirements
23-038 Specify iterator properties for Clauses 21 & 23
23-039 Reconsider return type of erase(iterator)
23-040 Need typedefs for map/multimap T type
23-042 Fix default container for priority_queue

Issues

Work Group: Library
Issue Number: 23-028
Title: Clean up empty sections in Clause 23
Sections: 23 (Containers library)
Status: Active

Description:

Clause 23 contains a large number of empty sections with no text, especially in the descriptions of the associative containers. These sections must be reviewed in detail. Either the appropriate text must be added to these sections or the sections should be deleted.

[Note: this problem applies to other library clauses as well, e.g. Clause 24 (Iterators library).]

Proposed Resolution:

Discussed in Monterey but no action was taken. Discussed again by the LWG in Tokyo.

Since most of the empty sections are simply placeholders, they can be removed easily after it is determined that they serve no purpose.

Therefore, leave the empty sections intact for now.

Requestor: Library Working Group
Owner:
Emails: (none)
Papers: (none)

Work Group: Library
Issue Number: 23-041
Title: Possible solutions for map::insert()
Sections: 23 [lib.containers]
Status: Active

Description:

--> Nathan Myers writes in c++std-lib-4239:

The problem with map<>::insert has been kicking around on comp.std.c++ for some time, and has come up here as well. The issue is that (given a map<> instance m and

a map insert iterator i) there is no concise way to
construct a value to pass:

```
m.insert(pair<const int,string>(3,"hi"));
*i = pair<const int,string>(3,"hi");
```

make_pair<>(), whatever its merits, is little help.

The problem is twofold: first, because the required "const"
cannot be deduced, the full type must be specified in the
call -- this repetition of type names is a general nuisance;
second, type deduction may deduce the wrong type anyway.
Any solution offered should solve both.

One approach to the problem would be to provide
a template converting constructor for pair<>:

```
template <class T1, class T2>
struct pair {
  template <class U, class V>
  pair(const pair<U,V>& p) : first(p.first), second(p.second) {}
  ...
};
```

One could then rewrite the above example as

```
m.insert(make_pair(3,"hi"));
*i = make_pair(3,"hi");
```

relying on the implicit conversion (e.g.)
pair<int,char*> --> pair<const int,string>.

A more conservative solution would be to provide a static
member function of map<>:

```
static value_type
  value(const K& k, const T& t) { return pair<const K,T>(k,t); }
```

One could then rewrite the above example as

```
m.insert(m.value(3,"hi"));
*i = m.value(3,"hi");
```

I would consider either of these a satisfactory solution.

--> Sean Corfield replied in c++std-lib-4241:

Of the two solutions, I suspect the converting constructor
will be more useful: it will help people using pair<> in
non-map code (and I have been bitten by this).

For the map-specific solution, what about a two-argument
version of 'insert' that simply constructs the correct
pair<> type and invokes the one-argument version?

Something like:

```
... insert(const T t, U u)
{ return insert(pair<const T, U>(t, u)); }
```

[I'd be quite happy with the static member value() -- this is
just another possible alternative]

Proposed Resolution:

Discussed by the LWG in Tokyo. No clear consensus was

reached. The LWG preferred adding a two-argument overload for insert(), but unfortunately this creates ambiguities with the existing template version of insert() that takes two Iterator arguments.

Requestor: Nathan Myers (myersn@roguewave.com)
Owner:
Emails: c++std-lib-4239, c++std-lib-4241
Papers: (none)

Work Group: Library
Issue Number: 23-043
Title: Fix container ambiguities when T == size_type
Sections: 23 [lib.containers]
Status: Active

Description:

Various types of calls to constructors & member functions are ambiguous for the case that the element of the container is a size_type: as long as C++ does not have constraints, the templates on InputIterator may conflict with the size/value methods.

A note should be added to explain how to disambiguate the constructors (do not default the allocator argument). A solution (possibly involving a defaultable dummy argument?) should be found for assign() and insert().

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579
Papers: (none)

Work Group: Library
Issue Number: 23-044
Title: Inconsistent insert() return types for assoc. containers
Sections: 23.1.2 [lib.associative.reqmts]
Status: Active

Description:

The table in 23.1.2 [lib.associative.reqmts] gives the following signatures:

```
pair<iterator, bool>    a_uniq.insert(t);
iterator                a_eq.insert(t);

iterator                a.insert(p,t);
```

Why is the case with the extra "hint" parameter p treated differently? In other words, in the latter case when inserting into a container with unique keys, there is no way to determine if an insertion actually takes place.

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579

Papers: (none)

Work Group: Library
Issue Number: 23-045
Title: Remove <stdexcept> from <bitset> synopsis
Sections: 23.2 [lib.sequences]
Status: Active

Description:

Remove the header <stdexcept> from the <bitset> header synopsis. It is not needed.

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579
Papers: (none)

Work Group: Library
Issue Number: 23-046
Title: Clean up bitset element access methods
Sections: 23.2.1 [lib.template.bitset],
23.2.1.2 [lib.bitset.members]
23.2.1.3 [lib.bitset.operators]
Status: Active

Description:

Make the following changes to class bitset:

- Add a const version of operator[](size_t) that returns bool.
- Add both const and non-const versions of at() to provide checked access (as is done for the other containers in clause 23).
- Provide semantics for operator[] and at() in 23.2.1.2 [lib.bitset.members] and 23.2.1.3 [lib.bitset.operators].

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579
Papers: (none)

Work Group: Library
Issue Number: 23-047
Title: Clarify complexity for deque::erase()
Sections: 23.2.2.6 [lib.deque.modifiers]
Status: Active

Description:

The complexity given for erase should be labelled as a worst case complexity.

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579
Papers: (none)

Work Group: Library
Issue Number: 23-048
Title: Improve description of list::sort()
Sections: 23.2.3.7 [lib.list.ops]
Status: Active

Description:

Need a more precise specification of the semantics for the list sort() functions.

Note: refer to 25.3 [lib.alg.sorting.] for possible wording to use.

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579
Papers: (none)

Work Group: Library
Issue Number: 23-049
Title: Clarify complexity for vector::insert(p,i,j)
Sections: 23.2.5.6 [lib.vector.modifiers]
Status: Active

Description:

The promise about the complexity if insert(p,i,j) is not compatible with the last sentence of the associated footnote. Change that last sentence to allow for copying the elements of the range before insertion.

In X3J16/95-0195 = WG21/N0795, P.J. Plauger adds:

The vector::insert template cannot meet the stated complexity requirements (originally intended for a random_access_iterator) when the template class parameter InputIterator is truly an input_iterator. They need to be *carefully* rethought. (See 23.2.5.2 for the handling of vector::vector template.)

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-050

Title: Add additional constructors to Container requirements
Sections: 23.1 [lib.container.requirements]
Status: Active

Description:

In section 23.1 [lib.container.requirements], the Container requirements table should also list the required constructors X(al) and X(a, al), for al an object of type Allocator.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-051
Title: Fix description of list::unique()
Sections: 23.2.3.7 [lib.list.ops]
Status: Active

Description:

The Effects section for list::unique() doesn't say what happens with binary_pred in the template form. Should say that the predicate for removal is either operator= or binary_pred.

Also, list::unique() does not apply the binary predicate ''Exactly size() - 1'' times if size() is zero. Should qualify the statement for non-empty lists only.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-052
Title: Fix description of list::merge()
Sections: 23.2.3.7 [lib.list.ops]
Status: Active

Description:

list::merge doesn't state the ordering criteria for either version of the two functions, at least not with sufficient completeness.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-053
Title: vector<bool>::const_reference should be bool
Sections: 23.2.6 [lib.vector.bool]
Status: Active

Description:

The definition for vector<bool, allocator>::const_reference should be bool, not const reference.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-054
Title: Define vector<bool>::reference::operator==()
Sections: 23.2.6 [lib.vector.bool]
Status: Active

Description:

vector<bool>::reference should define operator=(const reference& x) as returning ‘‘*this = bool(x)’’. The default assignment operator is not adequate for this class.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-055
Title: Fix return type of map::operator[]()
Sections: 23.3.1 [lib.map]
Status: Active

Description:

The return type of map::operator[] should be Allocator::types<T>.reference, not T&.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-056
Title: Remove const version of map::operator[]()
Sections: 23.3.1 [lib.map]
Status: Active

Description:

map::operator[](const key_type&) const is an unapproved (and nonsensical) addition. It should be struck.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-057
Title: Need semantics for associative containers
Sections: 23.3.1.1 [lib.map.types] and others
Status: Active

Description:

Much of the description of template classes map, multimap, set, and multiset have no semantics. These must be supplied.

Proposed Resolution:

Requestor: P. J. Plauger
Owner:
Emails: (none)
Papers: X3J16/95-0195 = WG21/N0795

Work Group: Library
Issue Number: 23-058
Title: Fix reverse iterator typedef arguments
Sections: 23.2.2 [lib.deque], 23.2.3 [lib.list]
23.2.5 [lib.vector], 23.2.6 [lib.vector.bool],
23.3.1 [lib.map], 23.3.2 [lib.multimap],
23.3.3 [lib.set], 23.3.4 [lib.multiset]
Status: Active

Description:

The following reverse iterator typedefs are incorrect:

deque::reverse_iterator	23.2.2 [lib.deque]
list::reverse_bidirectional_iterator	23.2.3 [lib.list]
vector::reverse_iterator	23.2.5 [lib.vector]
vector<bool>::reverse_iterator	23.2.6 [lib.vector.bool]
map::reverse_iterator	23.3.1 [lib.map]
multimap::reverse_iterator	23.3.2 [lib.multimap]
set::reverse_iterator	23.3.3 [lib.set]
multiset::reverse_iterator	23.3.4 [lib.multiset]

In each case, the typedefs only specify four template arguments, e.g.

typedef reverse_iterator<iterator, value_type,
const_reference, difference_type> reverse_iterator

However, the definitions of reverse_iterator and reverse_bidirectional_iterator require *five* template

arguments. Each of the above typedefs is missing a "pointer" template argument in the fourth position, after the reference argument but before the difference type.

Each typedefs should be written to read:

```
typedef reverse_iterator<iterator, value_type,
          reference, pointer, difference_type> reverse_iterator;
```

A complicating factor is that none of the containers in Clause 23 currently have a "pointer" typedef. Such a typedef must be introduced for each container, e.g.

```
typedef typename Allocator::types<T>::pointer   pointer;
```

Proposed Resolution:

Requestor: Larry Podmolik (podmolik@str.com)
Owner:
Emails: (none)
Papers: (none)

Work Group: Library
Issue Number: 23-059
Title: Wrong reverse iterator type for associative containers
Sections: 23.3.1 [lib.map], 23.3.2 [lib.multimap], 23.3.3 [lib.set], 23.3.4 [lib.multiset]
Status: Active

Description:

Each of the associative containers (map, multimap, set and multiset) supports only bidirectional iterators, but their reverse_iterator typedefs currently use the regular reverse_iterator adapter, which requires random access iterators. These typedefs should be specified using reverse_bidirectional_iterator instead.

Note: this issue is identical to issue 23-034, which dealt with list only. It was an oversight not to make the same fixes to the associative containers.

Proposed Resolution:

Requestor: Larry Podmolik (podmolik@str.com)
Owner:
Emails: (none)
Papers: (none)

Work Group: Library
Issue Number: 23-060
Title: Fix postcondition for (&a)->~X() in requirements table
Sections: 23.1 [lib.container.requirements]
Status: Active

Description:

In the Container requirements table, the postcondition for the expression (&a)->~X() refers to a.size(). This doesn’t make any sense, as the destructor call deletes the container object.

Proposed Resolution:

Requestor: German delegation comments
Owner:
Emails: c++std-edit-579
Papers: (none)

Work Group: Library
Issue Number: 23-061
Title: Reorganize Clause 23 sections
Sections: 23 [lib.containers]
Status: Active

Description:

The current overall structure of Clause 23 needs some work. In particular, bitset is not a Sequence (in the STL sense) and shuld be moved to its own section. Also, the container adapters belong in a separate section for the same reason (they are currently stuck in between list and vector).

I suggest the following organization for Clause 23:

- Introduction
- Fixed-size containers
  - <bitset>
- Variable-size containers
  - Requirements
  - Sequences
    - <deque>
    - <list>
    - <vector>
  - Associative Containers
    - <map>
    - <set>
  - Container adapters
    - <queue>
    - <stack>

Proposed Resolution:

Requestor: Larry Podmolik (podmolik@str.com)
Owner:
Emails: (none)
Papers: (none)

Work Group: Library
Issue Number: 23-062
Title: Remove() algorithm doesn’t work on map/multimap
Sections: 23 [lib.containers]
Status: Active

Description:

The remove() algorithm doesn’t work on map or multimap. Although remove() is specified to require only forward iterators, and map supports bidirectional iterators, the HP implementation required that the value_type of the collection be assignable. Map::value_type is a typedef for a pair<const Key, value>, therefore the compiler cannot generate asignment to the first member.

John Skaller responds in c++std-lib-4305:

>If the algorithm requires iterators with an mutable/
>assignable value type, then this can simply be added to the
>requirements of the algorithm(s) affected. Almost ALL other
>algorithms are affected -- for example you can't sort a
>constant container, the iterators need to have mutable value
>types.

Skaller further suggests that the iterator tags should be related by an inheritance structure.

Angelica Langer sums up in c++std-lib-4312:

:: We think there are two separate issues here:
:: The one is relating the iterator tags by means of
:: inheritance in order to prevent code duplication.
:: The other is to add new tags to express the difference
:: between constant and mutable iterators.

Proposed Resolution:

Requestor: Angelika Langer (langer@roguewave.com)
Owner:
Emails: c++std-lib-4305, c++std-lib-4308, c++std-lib-4312, c++std-lib-4314
Papers: (none)