Clause 24 (Iterators) Issues List (Rev. 3)

David Dodgson
dsd@tr.unisys.com
UNISYS

The following list contains the issues for Clause 24 on Iterators. The list is divided based upon the status of the issues. The status is either *active* - under discussion, *resolved* - resolution accepted but not yet in the working paper, *closed* - working paper updated, or *withdrawn* - issue withdrawn or rejected. They are numbered chronologically as entered in the list. Only the active and resolved issues are presented here. Those wishing a complete list may request one.

The proposed resolutions are my understanding of the consensus on the reflector.

1. Revision History

Revision 0 -	5/26/95	pre-Monterey	N0702/95-0102
Revision 1 -	9/25/95	pre-Tokyo	N0773/95-0173
Revision 2 -	11/30/95	pre-Santa Cruz	N0832/96-0014
Revision 3 -	5/23/96	pre-Stockholm	N0915/96-0097

2. *Active Issues*

Work Group: Library Clause 24
Issue Number: 24-021
Title: Separate Header for Stream Iterators
Section: 24.4
Status: active
Description:
From public review:
Drawing iostream into an implementation that just needs iterators is most unfortunate.

The current iterator header includes headers <ios> and <streambuf> to handle the stream iterators in 24.4. This requires all of I/O to be included in the iterators header. Yet I/O only needs this if the iterators are used.

If a new header is used should it be in clause 24 or in clause 27?
Is <iositer> a good name for the new header?
Should the stream iterators be incorporated into current I/O headers?

From Nathan Myers:
Message c++std-lib-4174
There are natural places for each of these iterator templates.
Move istream_iterator<> to <istream>.
Move ostream_iterator<> to <ostream>.
Move istreambuf_iterator<> and ostreambuf_iterator<> to <streambuf>.
Add forward declarations of all four to <iosfwd>.

Proposed Resolution:
Move the stream iterators into the I/O headers.

Remove #include's for iosfwd, ios, and streambuf from 24.1.6 [lib.iterator.tags] Header <iterator> synopsis and tags for subclause 24.4.

Move istream_iterator to <istream>, ostream_iterator to <ostream>, and the streambuf iterators to <streambuf>. Add forward declarations of all four to <iosfwd>. Add #include <iterator> in these headers.

Requestor: Public Review & Library WG
Owner: David Dodgson (Iterators)
Emails: lib-4174,4186,4191,4199,4202
Papers:

Work Group: Library Clause 24
Issue Number: 24-024
Title: Operator ->* Issues for Iterators
Section: 24.1.3, 24.1.1
Status: active
Description:

24.1.1, 24.1.3 p24-2,4:

Should operator->* be added for iterators?

Section 14.3.3 [temp.opref] specifically allows operator-> to appear in a template where its return type cannot be dereferenced if it is not used. No such guarantee is made for operator->*. If operator->* is desired, the same guarantee should be made.

A proposal to change the core language to have operator->* work in a similar fashion to operator-> was rejected at the Santa Cruz meeting.

Including operator->* in an iterator (or auto_ptr) requires a series of member templates, helper classes, and partial specialization.

Does operator-> work correctly for input iterators? (*a can return an rvalue).

Resolution:
Requestor: Library WG
Owner: David Dodgson (Iterators)
Emails: lib-4301,4559-4560
Papers:

Work Group: Library Clause 24
Issue Number: 24-028
Title: Const Attribute in Iterator Requirements
Section: 24.1
Status: active
Description:

24.1:

The tables in Clause 24 of Iterator Requirements include mutative operations such as ++ and =, but make no mention of constness. We should distinguish which operations require a non-const operand and which can be performed on a const operand. (e.g. is *a allowed on a const iterator?)

Proposed Resolution:

All operations found in the tables can be applied to const operands except: ++a a++ --a a-- a= a+= a-=.

Requestor: Nathan Myers
Owner: David Dodgson (Iterators)
Emails: lib-4172
Papers:

Work Group: Library Clause 24
Issue Number: 24-029
Title: Streambuf Iterator Issues

Section: 24.1.6 and 24.4
Status: active
Description:

24.1.6 header and 24.4 on streambuf iterators:
These issues are raised by P.J. Plauger in N0795:

24.1.6:
Class istreambuf_iterator should be declared with public base class input_iterator. There is then no need to add a special signature for iterator_category (which is missing from the <iterator> synopsis).

24.1.6:
Template operator==(istreambuf_iterator) should not have default template parameters.

24.1.6:
Template operator!=(istreambuf_iterator) is ambiguous in the presence of template operator!=(const T&, const T&). It should be struck.

24.1.6:
Class ostreambuf_iterator should be declared with public base class output_iterator. There is then no need to list a special signature for iterator_category.

24.1.6: (Done 1/96)
Template operator==(ostreambuf_iterator) and corresponding operator!= are nonsensical and unused. They should be struck.

24.4.3: (Withdrawn 3/96)
istreambuf_iterator should have a member ``bool fail() const'' that returns true if any extractions from the controlled basic_streambuf fail. This is desperately needed by istream to restore its original approved functionality when these iterators are used with facet num_get.

24.4.3.2:
istreambuf_iterator(basic_istream s) should construct an end-of-stream iterator if s.rdbuf() is null. Otherwise, it should extract an element, as if by calling s->rdbuf()->sgetc(), and save it for future delivery by operator*(). (Lazy input, however, should be permitted.)

24.4.3.2: (Done 1/96)
istreambuf_iterator(basic_streambuf *) has no description

24.4.3.3: (Done 1/96)
istreambuf_iterator::operator*() should deliver a stored element, or call sgetc() on demand, then store the element. It should *not* extract a character, since this violates the input_iterator protocol.

24.4.3.4: (Done 1/96)
istreambuf_iterator::operator++() Effects should say that it alters the stored element as if by calling s->snextc(), where s is the stored basic_streambuf pointer.

24.4.3.7:
template operator==(istreambuf_iterator&, istreambuf_iterator&) should have const operands.

24.4.3.8:
template operator!=(istreambuf_iterator&, istreambuf_iterator&) should have const operands. It also is ambiguous in the presence of template<class T> operator!=(T, T) (as are many operators in the library).

24.4.4: (Done 1/96)
ostreambuf_iterator::equal is silly, since output iterators cannot in general be compared. It should be struck.

24.4.4: (Done 1/96)
ostreambuf_iterator should remove all references to equal, operator==, and operator!=. Output iterators cannot be compared.

24.4.4: (Done 1/96)
ostreambuf_iterator should have a member ``bool fail() const'' that returns true if any insertions into the controlled basic_streambuf fail. This is desperately needed by ostream to restore its original approved functionality when these iterators are used with facet num_put.
ostreambuf_iterator should add the member `bool failed() const', which returns true only if an earlier insertion failed. It is needed by num_put in 22.2.2.2 to communicate insertion failures to inserters in 27.6.1.2. With this change, I believe the following example inserter from basic_ostream satisfies all the exception-handling requirements in the current draft:

```
Mytype& operator<<(long X)
 {iostate stat = goodbit;
 if (opfx())
  {const Myfacet& fac = use_facet<Myfacet>(getloc());
  try {
   if (fac.put(Myiter(rdbuf()), Myiter(0), (ios_base&)*this,
    stat, X).failed()
    stat |= badbit; }
  catch (...) {
   setstate(badbit, Rethrow); }} // added argument
 osfx();
 setstate(stat);
 return (*this); }
```

24.4.4.1: (Done 1/96)
ostreambuf_iterator::ostreambuf_iterator() produces a useless object. It should be struck.

24.4.4.1:
ostreambuf_iterator;:ostreambuf_iterator(streambuf *) should require that s be not null, or define behavior if it is.

24.4.4.2: (Done 1/96)
ostreambuf_iterator::equal is not needed and should be struck.

24.4.4.3: (Done 1/96)
ostreambuf_iterator::operator== is silly, since output iterators cannot in general be compared. It should be struck.

24.4.4.3: (Done 1/96)
ostreambuf_iterator::operator!= is silly, since output iterators cannot in general be compared. It should be struck.

Resolution: Resolution as suggested in N0795
Requestor: Bill Plauger
Owner: David Dodgson (Iterators)
Emails: lib-4299,4404,4406-4407,4409-4412
Papers: pre-Tokyo N0795

Work Group: Library Clause 24
Issue Number: 24-038
Title: Removal of proxy class
Section: 24.4.3 [lib.istreambuf.iterator]
Status: active
Description:
24.4.3:

The changes to input iterator semantics make the proxy class an implementation detail. It should not be required as part of the standard.

From P.J. Plauger in N0795:
24.4.3:
istreambuf_iterator should remove all references to proxy, whether or not Koenig's proposal passes to make more uniform the definition of all input iterators. It is over specification.

24.4.3.1:
istreambuf_iterator::proxy is not needed (once istreambuf_iterator is corrected as described below). It should be removed.

24.4.3.2:
istreambuf_iterator(const proxy&) should be removed.

24.4.3.4:
istreambuf_iterator::operator++(int) Effects should say that it saves a copy of *this, then calls operator++(), then returns the stored copy. Its return value should be istreambuf_iterator, not proxy.

Editorial box 69 suggests that proxy be replaced by an opaque unnamed type.

Resolution:
Requestor: David Dodgson
Owner: David Dodgson (Iterators)
Emails:
Papers: N0795, Updated Issues List for Library, pre-Tokyo
N0833, Proposed Iterators Changes, pre-Santa Cruz

Work Group: Library Clause 24
Issue Number: 24-039
Title: Return Type of operator* in istreambuf_iterator
Section: 24.4.3
Status: active
Description:
24.4.3 24.4.3.3:
The istreambuf_iterator operator* function is declared as returning char T and its description says it returns the result of sbuf_->sgetc(). However sgetc() returns int_type, so the risk of data truncation exists.

Proposed Resolution:
Change the return type for operator* in 24.4.3 and 24.4.3.3 to traits::int_type.
Requestor: Cathy Kimmel (kimmel@decc.enet.dec.com)
Owner: David Dodgson (Iterators)
Emails:
Papers:

Work Group: Library Clause 24
Issue Number: 24-040
Title: Header Synopsis Includes
Section: 24.1.6 [lib.iterator.synopsis]
Status: active
Description:
24.1.6 p11:
From lib-4691:
Several public comments pointed out that the C++ header synopsis #includes of other C++ headers were not correct.

In the table below, included headers are marked with a single + where Judy proposes adding them to the #includes for the indicated header. A double ++ indicates that a German public comment also proposed the addition. Includes for headers marked with a - are proposed for removal, while those with neither a + or - are to remain unchanged.

24 iterator
+ istream

+ ostream
+ functional
- iosfwd
- ios
- streambuf
cstddef

Proposed Resolution:
+ istream
basic_istream is referenced in istream_iterator and istreambuf_iterator
+ ostream
basic_ostream is referenced in ostream_iterator and ostreambuf_iterator
+ utility
utility is used for char_traits and for the library != operator (note: functional is not used)
- iosfwd
not referenced because of the istream/ostream references
- ios
the stream iterators now reference char_traits
streambuf
istreambuf_ and ostreambuf_ iterators both reference basic_streambuf ???

Requestor: Judy Ward / Beman Dawes
Owner: David Dodgson (Iterators)
Emails: lib-4691
Papers:

Work Group: Library Clause 24
Issue Number: 24-041
Title: Distance Type for Output Iterators
Section: 24.1.6
Status: active
Description:
24.1.6:
All iterators except output iterator use distance type. Input iterators have no distance per se but the distance_type is used as a count type for certain algorithms. A count type would also be useful for certain algorithms using output iterators. The proposal is to define distance_type for output iterators.

The recent addition of the iterator_traits proposal defines a separate template iterator for definition of the types associated with an iterator. Currently category, value, and distance are the types defined. To have distance not defined for output iterators will require a partial specialization. It is a cleaner and more consistent interface to allow distance to be specified for all types of iterators.

Resolution: See paper 96-0091/N909
Requestor: Angelika Langer
Owner: David Dodgson (Iterators)
Emails:
Papers: X3J16/96-0091 WG21/N0909

3. *Resolved Issues*

Work Group: Library Clause 24
Issue Number: 24-003
Title: const operation for iterators
Section: 24.3
Status: resolved
Description:
24.3.1 p24-13 Box 108
Suggest that the operator *() for STL iterators be made into a const operation.

The function
```
  void fn (const ReverseIterator & x) {
   ...
     y = x*;
   ...
  }
```
shows that the operation * is not defined as const in the reverse_iterator (DRAFT 20 Sept 1994, 24.2.1.2). However, the body of the function does not modify the iterator object.

Of course, const Iterator is different from const_iterator and from

This change was accepted in Monterey (see N740). However, in box 108, Corfield says it seems wrong to have const member functions return a reference or a pointer to non-const T. He believes this should be reconsidered for operator* and operator->.

It has been further suggested in public review that const should also be used the descriptions in 24.3.1.2.2 and 24.3.1.2.3. (Editorial if accepted.) Also, the same decisions should be made for reverse_iterator in 24.3.1.3, 24.3.1.4.2, and 24.3.1.4.3.

The changes to make const uniform were accepted in Santa Cruz. Sean Corfield has withdrawn his comments.

Proposed Resolution:
Both base() and operator*() should be const.
Accepted in Monterey - N740

As stated above, there is a difference between const iterator and const_iterator. The template parameters must specify const if const T is desired.

Reverse_iterator should be treated the same as reverse_bidirectional_iterator.

Further changes were accepted in Santa Cruz - N0833

Requestor: Bob Fraley <fraley@porter.hpl.hp.com>
David Olsen (public review comment #17)
Owner: David Dodgson (Iterators)
Emails: c++std-lib-3135
Papers: N740 - Small Changes
N833 - Proposed Iterators Changes

Work Group: Library Clause 24
Issue Number: 24-006
Title: Relaxing Requirement on Iterator++ Result
Section: 24.4.3
Status: resolved
Description:
24.4.3 p24-23
The return type of operator++ for istreambuf_iterator is listed as 'proxy'. This suggestion is to make the return type an object which is "convertible to const X&" rather than "X&".
Resolution: accepted in Austin
Requestor: Nathan Myers
Owner: David Dodgson (Iterators)
Emails:
Papers: 95-0021/N0621 (Pre-Austin mailing)

Work Group: Library Clause 24
Issue Number: 24-007
Title: Fixing istreambuf_iterator
Section: 24.4.3
Status: resolved
Description:

24.4.3 p24-23:
Proposes the addition to istreambuf_iterator of
inline istreambuf::proxy::operator istreambuf_iterator()
{ return sbuf_; }
to better conform to the Forward Iterator specification.
Resolution: accepted in Austin
Requestor: Nathan Myers
Owner: David Dodgson (Iterators)
Emails:
Papers: 95-0022/N0622 (Pre-Austin mailing)

Work Group: Library Clause 24
Issue Number: 24-013
Title: Const declaration of operator[]
Section: 24.3.1.3 [lib.reverse.iterator]
Status: resolved
Description:
24.3.1.3 p24-15.16: [Box 109] and
24.3.1.4.11 p24-19: [Box 110]
Should operator[] of reverse_iterator be specified as const?
Proposed Resolution:
Same resolution as issue 3 (Box 108 in lib.reverse.bidir.iter
section 24.3.1.1 for reverse_bidirectional_iterator)

This was accepted and added to the working paper. Box 109 from Corfield states that he thinks it is wrong to return a non-const T from a const member function.

Again, this should be resolved as issue 3.

This was accepted in Santa Cruz

Resolution: specified as const - See N740
Requestor: Editorial box
Owner: David Dodgson (Iterators)
Emails:
Papers: Small Changes, 95-0140/N0740, David Dodgson, post-Monterey
Proposed Iterators Changes, 96-0015/N0833R1, post-Santa Cruz

Work Group: Library Clause 24
Issue Number: 24-030
Title: Distance Requirement
Section: 24.2.6
Status: resolved
Description:
24.2.6 p24-12 [lib.operator.operations]:

24.2.6:
Template function distance should have the requirement that last is reachable from first by incrementing first.

Resolution: As suggested
Requestor: Bill Plauger
Owner: David Dodgson (Iterators)
Papers: N0833R1 - Proposed Iterators Changes, post - Santa Cruz

---

Work Group: Library Clause 24
Issue Number: 24-032
Title: Insert Iterator Issues
Status: resolved
Description:
24.3.2 p24-18 [lib.insert.iterator]:

24.3.2.3:
Template class front_insert_iterator should not have a Returns clause.

24.3.2.5:
insert_iterator::operator++(int) returns a reference to *this, unlike in other classes. Otherwise, the update of iter by operator= gets lost.

24.3.2.6.5:
Declaration for template function inserter is missing second template argument, class Iterator. It is also missing second function argument, of type Iterator.

Resolution:
Requestor: Bill Plauger
Owner: David Dodgson (Iterators)
Emails:
Papers: N0833R1 - Proposed Iterators Changes, post Santa Cruz

---

Work Group: Library Clause 24
Issue Number: 24-033
Title: Iterator Category Defintion
Section: 24.1.6 [lib.iterator.tags]
Status: resolved
Description:
24.1.6:
Iterator tags could be related by inheritance. Doing so would allow a more generic solution to algorithms which are multiply defined based on iterator category. For example, it might be possible to define to versions of an algorithm, one based on output_iterator and one based on forward_iterator. Iterator categories which inherit from forward_iterator could use the second algorithm. If the categories are inherited, then the based classes should use inheritance.

It may also be desirable to provide a mechanism to indicate whether an iterator is constant or mutable. Different algorithms on iterators could be used if this information was available.

Resolution: Inheritance in iterator tags accepted in N833R1 accepted in Santa Cruz.
input -> forward -> bidirectional -> random
Requestor: Angelika Langer
Owner: David Dodgson (Iterators)
Emails: lib-4305,4308,4312,4315
Papers: N0833, Proposed Iterators Changes, pre-Santa Cruz

Work Group: Library Clause 24
Issue Number: 24-034
Title: Reverse Iterator Description [Box 107]
Section: 24.3.1 [lib.reverse.iterators]
Status: resolved
Description:
24.3.1 p24-13 p3:
Box 107 (from Corfield) states that the description for a reverse iterator return type should specify the return type, not a reference. The Reference and Pointer parameters include the appropriate type definitions.
This paragraph appears to be a holdover from before the parameters for the template were reworded. This paragraph should be reworded to conform to the new parameters.

Requestor: Editorial Box
Owner: David Dodgson (Iterators)
Emails:
Papers: N833R1 - Proposed Iterators Changes, post Santa Cruz

Work Group: Library Clause 24
Issue Number: 24-035

Title: Typos in 26 Sept. 95 Draft
Section: 24.1.6, 24.3.1.4.15
Status: resolved
Description:

24.1.6 p11:

After paragraph 11 the following title appears:

Header <iterator> synopsislib.iterator.synopsis

<- bold - - - - -><- normal - - - ->

Either the additional wording was added unintentionally or an attempt was made to add a header. Since the synopsis does not belong in the previous section (Iterator tags), a new header should be added. Other clauses seem to have a separate header before the synopsis, perhaps "Iterator classes" would serve?

24.3.1.4.15:

The header for 24.3.1.4.15 [lib.reverse.iter.opsum] states it is "operator==" when it should be "operator+". Operator== has already been defined and the code in this section is for operator+.

Resolution:

Add a new header before the synopsis.

Change the header for 24.3.1.4.15 to operator+.

Requestor: David Dodgson
Owner: David Dodgson (Iterators)
Emails:
Papers: N833R1 - Proposed Iterators Changes, post Santa Cruz

Work Group: Library Clause 24
Issue Number: 24-037
Title: Iterator Traits
Section: 24.
Status: resolved
Description:

24.:

Define the types governing iterators in an iterator_traits class.

```
template <class Iterator>  struct iterator_traits {
  typedef Iterator::distance_type      distance_type;
  typedef Iterator::value_type         value_type;
  typedef Iterator::iterator_category  iterator_category; }
```

The types for any iterator could then be referenced as:

```
iterator_traits<Iter>::distance_type ...;
```

Partial specialization would be used for pointer types:

```
template <class T>  struct iterator_traits<T*> {
  typedef ptrdiff_t  distance_type;
  typedef T  value_type;
  typedef random_access_iterator_tag  iterator_category; }
```

Additionally, the current base classes for iterators would be replaced by:

```
template <class Category, class T, class Distance=ptrdiff_t >
struct iterator {
  typedef Distance distance_type;
  typedef T        value_type;
  typedef Category iterator_category; }
```

which would be used as:

```
class MyIter:public iterator<bidirectional_iterator_tag,
                             double, long>  { ... }
```

Resolution: Accepted in Santa Cruz
Requestor: Bjarne Stroustrup, Alex Stepanov, Matt Austern
Owner: David Dodgson (Iterators)
Emails:
Papers: N847,Bring Back the Obvious Definition of Count, pre-Santa Cruz