2. German public review comments

No.	Comment	Author	incoming Date	WP chapter	DIN WG Topic leader	related Document	German WG vote must,shall,nice,apposed, abstain	Re-commen-dation for ISO/ANSI [status]
1	Derived Classes comments	Ulrich Eisenecker	26.6.1995	10	Eisenecker	Email: 26.6.95 X3J16: edit-561	not voted on	**yes**, [open]
2	Numerics Library	Ulrich Eisenecker	26.6.1995	26	Eisenecker	Email:26.6.95 X3J16: edit-564	not voted on	**yes**, [open]
3	class auto_ptr: the copy ctor should be private to forbid parameter passing of auto_ptr per value.	Uli Breymann	30.6.1995	20.4.5	Bormann, Breymann	Email: 21.5.95 X3J16: edit-566	not voted on	**yes**, [open]
4	What is „m“ ?	Uli Breymann	30.6.1995	23.3.1 p. 23-32	Bormann, Breymann	Email: 21.5.95 X3J16: edit-568	not voted on	**yes**, [open]
5	Logic error: replace !(*i > *j) with !(*i < *j)	Uli Breymann	30.6.1995	25.3.2 p. 25-20	Bormann, Breymann	Email:21.5.95 X3J16:lib-3822	not voted on	**yes**, [open]
6	Findings in chapter 20,23,24,25	Carsten Bormann	30.6.95	20	Bormann, Breymann	Fax: 30.6.95 X3J16: edit-579 lib-3829	not voted on	**yes**, [open]
7	Layout compatible types	Manfred Lichtmannegger	30.6.95	3.9	Unruh	X3J16: edit-576	not voted on	**yes**, [arrays still open]
8	pop() should return a value	Nicolai Josuttis	6.12.95	23.2.4.1 23.2.4.2 23.2.4.3	Kiefer	X3J16: lib-4496	pop_value()	**yes** [open]
9	Mandate for bad_alloc	Ulrich Eisenecker	7.12.95	26.3	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
10	Can the class instantiated by the user ?	Ulrich Eisenecker	7.12.95	26.3.2.4 footnote 219	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
11	Design of basic_istream and basic_ostream	Udo Mueller	8.12.95	27	Kiefer, Lichtmann-egger	X3J16: lib-?	note that op << with int is not possible	**yes** [open]
12	Editorial issues	Udo Mueller	8.12.95	20.3.6.1 20.4.1.1 20.4.1.3	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
13	Editorial issues	Udo Mueller	8.12.95	27	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
14	Wrong Specifications for Logical Iterator Conditions	Ulrich Breymann	6.1.96	25.3.3.1	Kiefer	X3J16: lib-?	not voted on	**yes** [open]
15	Missing iterator parameter	Ulrich Breymann	24.1.96	24.3.2.6.5	Kiefer	X3J16: edit-?	not voted on	**yes** [open]
16	Instantiation of virtual functions in templates	Udo Müller	13.2.96	14.3.2, §6	Unruh		**apposed** (4:1)	**no**
17	Minor Typos	Nicolai Josuttis	8.12.95	21.1.1.8.3 21.1.1.9.1 23.2.1 23.2.1.2	Kiefer	X3j16: edit-624	not voted on	**yes** [open]
18	Remove find_first, rfind and find_last	Nicolai Josuttis	8.12.95	21.1.1.9.3 21.1.1.9.4	Kiefer	X3j16: edit-624	WG recommends an informative vote	**yes** [open]
19	Introduce const & bitset::operator[] () const	Nicolai Josuttis	8.12.95	23.2.1	Kiefer	X3J16: lib-4496	not voted on	**yes** [open]
20	Specify #include <iterator> for containers	Nicolai Josuttis	24.1.96	21.1 23.2 23.3	Kiefer	X3J16: lib-4496	not voted on	**yes** [open]
21	capacity() and reserve()	Nicolai Josuttis	24.1.96	21.1.1.6 23.2.5.4	Kiefer	X3j16: lib-4496	not voted on	**yes** [open]
22	Move min(), max() swap() into utility part	Nicolai Josuttis	24.1.96	25 -> utility	Kiefer	X3j16: lib-4496	not voted on	**yes** [open]
23	Move bitset section	Nicolai Josuttis	19.2.96	23.4	Kiefer	X3J16: edit-624	not voted on	**yes** [open]

24	Distance n=0	Ulrich Breymann	19.2.96	24.1.6	Kiefer	X3J16: edit-?	not voted on	**yes** [open]
25	Access to Exception Object's Type Information	Roland Hartinger	6.3.96	5.2.7 15.5.4 18.5.3	Hartinger	X3J16/96-0052 WG21/N0870 core-6404	not voted on	**yes** [open]

1) Comment from the German WG member Ulrich Eisenecker:

10.3, paragraph 5:

The first sentence is much too long and needs to be
cut in at least to or more sentences.

General comment:
I missed a hint in this chapter, that a pure virtual
function may be fully implemented.
---
2) Comment from the German WG member Ulrich Eisenecker:

26 Numeric Library:
First sentence: IMO "seminumerical" operations is a
very unhappy expression.
It should be better turned into "numerical".

26.2.5
"template<class T>
istream& operator>>(istream& is, complex<T>& x);"

require: ... (which _may_ throw ios::failure)

How is the user instructed about the specific behavior ?
It is not described, how that is done, nor that it must done.
Of minor importance may be the aspect of portability (but I
criticized that already earlier). I critized as well, that I
miss any exception specifications in the functions, even such
ones, that the function will throw no exceptions. (Old stuff,
I know and I don't want to continue repeating it.)

26.2.6, 6th line:
For what stands the abbrivation TBS ?

26.3, 4th line:
for what stands the abbrivation BLAS-like ?

26.3.2.3, paragraph 2:
There is a superfluous dot in the "a..max()".

26.3.3, first paragraph:
Again: what means the abbrivation BLAS-like ?
---
3) Comment from the German WG member Uli Breymann:

Error report/Comment to CD 20.4.5 Template class auto_ptr
The copy c'tor is wrong. Consider:

```
int f(auto_ptr<myType> Ptr)  // copy c'tor called
{
   return Ptr->getSomeInfo();
}      // Error: Ptr goes out of scope and deletes object

int main()
{  auto_ptr<myType> myAutoPtr(new myType);

   cout << "Info:"
   << f(myAutoPtr);      // oops!

   myAutoPtr->doSomeThing(); // Error: Objekt doesn't exist any ´more
```

}

The copy c'tor should be private to forbid its use, so that ˝parameter passing of auto_ptrs per value is not possible. Auto_ptr objects should be passed only by reference. Of course, the concept of ˝strict ownership (page 20-16, bottom) has to be modified: Ownership ˝cannot be transferred as in operator=().

Proposal
========

```
public:
 void operator=(const X*); is public and allowed if the auto_ptr
                    is equal NULL, otherwise runtime error
private:
 auto_ptr(auto_ptr&);        // copy c'tor : use is not allowed
 void operator=(const auto_ptr&); // private: use is not allowed
```

New semantic of 'strict ownership':
If an auto_ptr P once has got ownership of an object, it can get rid of it only by calling P.release() or by going out of scope (i.e. destroying the object).

As the public operator=(const X*) is defined in this proposal, ˝there is no need any more for the method reset(X* p=0).

---

4) Comment from the German WG member Uli Breymann:

Error report/Comment to CD 23.3.1 Containers Library, p. 23-32

The last line on p. 23-32 is wrong and does not explain anything.
1. What is "m"? It was not mentioned before.
2. Does access with operator[] always mean to add an element constructed with T() if the key x cannot be found? (If x ˝is already there, the insert-operation is a no-op, anyway)

Proposal
========
What is meant is clear by definition of a map, but it should be written down exactly.

---

5) Comment from the German WG member Uli Breymann:

Error report/Comment to CD 25.3.2 Algorithms, p. 25-20
Logic error in last line.
Replace !(*i > *j)
with !(*i < *j)

---

6) Comments from the German WG member Carsten Bormann to Chapter ˝20, 23, 24, 25:

1. 20.1

This subclause is not consistent with the new kind of allocator ˝with a types<t> member type (see 20.4).

2. 20.1-Table 33

Define ``reference''.

3. 20.3.5-1

Change the first ``Returns:'' to ``operator() returns''.

4. 20.3.6.4-1

Change ``not greater'' to ``less'' in the example.

5. 20.4.1

Explain ``for use as the this value in a constructor or ˝destructor".

6. 20.4.1-Note 169

Change ``In'' to ``An''.

7. 20.4.1.1

The semantics given for allocate() are clearly wrong, as no ˝construction takes place.

8. 20.4.1.2

Fix font of .allocate<char,void>.

9. 20.4.3.1-Note 170

This will, in general, not be possible for memory models with the ˝same distance type. Alternatively, change the example to say ˝``...allocate(long long n, T huge *)'' and change the text accordingly.

10. 20.4.3.5

Explain the role of the second parameter of ˝return_temporary_buffer().

11. 20.4.5.2

Explain ``->m'' notation.

12. 20.4.5.2

What does release() return?

13. 20.4.5.2

What does reset() return? Does a reset imply a release or is ˝the old contents destroyed?

14. 23: lib.containers

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().

15. 23.1-Table 50

For (&a)->~X(), it is strange to call a method in the ˝definition of a postcondition.

16. 23.1-4

It would be useful to define that begin() returns an iterator ˝that can be used to enumerate all elements of the container in sequence and˝that is equal to end() if incremented beyond the last element (or if˝the con-

tainer is empty).

17. 23.1.1-6-Table 53

For a.back(), change *a.end() to *--a.end().

18. 23.1.1-6-Table 53

Add at().

19. 23.1.2-7-Table 54

Explain that the complexity of a.key_comp() and a.value_comp() ˝is *by definition* constant (this is not a statement about their ˝complexity, but a statement about what constant complexity means here).

20. 23.1.2-7-Table 54

a.insert(t) should be a_eq.insert(t).

21. 23.1.2-7-Table 54

Explain why a.insert(p,t) does not return a pair.

22. 23.1.2-7-Table 54

Explain that and when a.lower_bound() and a.upper_bound() can ˝return a.end().

23. 23.2

<bitset> Synopsis: Why include stdexcept?

24. 23.2.1

Add a const_reference type and associated methods.

25. 23.2.1-1

The comments for operator~(), operator bool() and flip() are˝misaligned.

26. 23.2.1

Bitsets should have at(i) methods. Bitsets should be consistent˝to other containers with respect to checked and unchecked member ˝access.

27. 23.2.1.2

Define the semantics of operator[].

28. 23.2.2

The typedef for iterator and const_iterator, if taken seriously, ˝would disallow common implementation techniques.

29. 23.2.2.1

Subclause is empty.

30. 23.2.2.3

Subclause is empty.

31. 23.2.2.4

While the standard generally does not have the right tools to talk ˝about memory consumption, it would be useful to know whether a ˝resize(0) is supposed to free storage or not (cf. 23.2.5.4).

32. 23.2.2.5

Subclause is empty. This should, in particular, define the ˝difference between a[i] and a.at(i).

33. 23.2.2.6

Change ``references to the deque'' to ``references to elements of ˝the deque''.

34. 23.2.2.6

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

35. 23.2.3

The typedefs for iterator and const_iterator do not make sense.

36. 23.2.3.1

Subclause is empty.

37. 23.2.3.2

Define semantics of constructors and assignment (e.g., by ˝reference to sequence requirements).

38. 23.2.3.3

Subclause is empty.

39. 23.2.3.7-2

splice(pos, x, i): change ``unchanged is'' to ``unchanged if''.

40. 23.2.3.7-2

remove/remove_if: Explain ``the list iterator i''.

41. 23.2.3.7-2

sort: Explain ``the operator<''.

42. 23.2.4.3-2

Define operator<.

43. 23.2.5

The typedef for iterator and const_iterator, if taken seriously, ˝would disallow some alternative implementation techniques.

44. 23.2.5.1

Subclause is empty.

45. 23.2.5.3

Subclause is empty.

46. 23.2.5.6

The promise about the complexity if insert(p,i,j) is not ˝compatible with the last sentence of note 192. Change that last sentence to allow ˝for copying the elements of the range before insertion.

47. 23.2.6

The typedefs for iterator and const_iterator do not make sense.

48. 23.2.6

Say explicitly that the specialization is intended to have the ˝same semantics as the general case.

49. 23.3.1, 23.3.2, 23.3.3, 23.3.4

The typedefs for iterator and const_iterator do not make sense.

50. 23.3.1.1-23.3.1.4

Subclause is empty.

51. 23.2.1.5

Explain m.

52. 23.3.1.6-23.3.1.8

Subclause is empty.

53. 23.3.3.1-23.3.3.7

Subclause is empty.

54. 24.1.4-Table 60

Explain ``--r == --r implies r == s".

55. 24.1.6-2

``can be defined", i.e., it is the user's responsibility? ˝Explain that this is part of <iterator>.

56. 24.1.6-5

``may define"? Repeat language from 20, ``for all memory models,˝..."

57. 24.1.6-11

Header <iterator>: Drawing iostream into an implementation that ˝just needs iterators is most unfortunate. The contents of the header ˝<iterator> should be confined to those operations that do not need ˝iostream; the rest should be put into a separate header.

58. 24.4

This should be part of the iostream library clause. In this ˝context, it

should be decided whether this subclause needs to be templatized together with the rest of iostream.

59. 24.3.1.2.5

Returns: *this

60. 24.3.1.2.6

Returns: x.base() == y.base(); (There is no conversion from a ˝reverse iterator to its base.)

61. 24.3.1.3-1

The note seems misplaced, but does also apply here analogously.

62. 24.3.1.4.5

Returns: *this

63. 24.4.3.5

Change ``iterator over'' to ``iterate over''.

64. 25-5, 25-6

Predicate and BinaryPredicate are type parameters, not classes.

65. 25-5, 25-6

Predicate and BinaryPredicate are not constrained to return a ˝Boolean value. Unfortunately, most of the descriptions that follow use ˝pseudo-code such as ``pred(*i) == true''. This does not have identical˝seman-tics with the description here that the predicate is used in an ˝if(){} context. Change all further occurrences of ``== true'' into ``!= false''.

66. 25-7

Explain X and Distance.

67. 25.1.3, 25.1.4

Align names of find_end and find_first_of.

68. 25.1.4

Returns: Change pred(i, first2+n) to pred(*i, *(first2+n)).

69. 25.1.4

Complexity: explain how a forward iterator can be used as a ˝measure for the number of applications.

70. 25.1.5

Explain ``value''.

71. 25.1.9

Explain why variant 3 uses a ``const T&'' while variant 4 uses a ˝``T'' as its third parameter.

72. 25.2.7, 25.2.8

Explain assignment complexity of remove, remove_if, unique (i.e., ˝whene-
ver the term ``Eliminates'' is used).

73. 25.2.8

Explain that *(first-1) is not accessed.

74. 25.2.9, 25.2.10

Correctly indent the assignment under Effects of X_copy.

75. 25.3.2-1

Change ``comp(*i, *j)'' to ``comp(*j, *i)''.

76. 25.3.3

Add the assumption that the sequences are sorted.

77. 25.3.3.3

Add ``without violating the ordering'' to the first sentence of ˝effects.

78. 25.3.5.4, 25.3.5.5

Define the terms ``difference'' and ``symmetric difference" ˝unambi-
guously.

79. 25.3.8

Unambiguously define ``lexicographically less than''.

80. 25.4

A mandate (``shall'') cannot be in a note.

---

7) Comment from the German WG member Manfred Lichtmannegger:

Layout-Compatible Types
=======================

Chapter [basic.types] 3.9 Type paragraph 10 defines ˝layout-compatibility

If two types T1 and T2 are the same type, then T1 and T2 are ˝layout-compatible
types. [Note: layout-compatible enumerations are described in ˝7.2.
layout-compatible POD-structs and POD-unions are described in ˝9.2.]

I am not sure about cv-qualifier with respect to "T1 and T2 are ˝the same type",
but I would have expected that pointer types to layout-compatible ˝types would
be layout-compatible and that array types of layout-compatible ˝types with
the same number (!) of elements would be layout-compatible. But ˝as there is
no such definition these types are only layout-compatible when ˝they are
exactly the same types.

---

8) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
Introduce pop_value() for container adaptors

Introduction:
Due to time penalties, the STL container
adaptor classes have no function
that removes the next element AND returns it.

Instead two different functions top() and pop()
have to get called.
As the normal interaction with stacks and queues
is to process the next element I suggest
to introduce as add on a function pop_value() that
does the job.

Proposed Changes:
Insert into class definition of
queue (23.2.4.1):

```
value_type pop_value() {
    T tmp(front());
    pop();
    return tmp;
}
```

Insert into class definition of
priority_queue (23.2.4.2):

```
value_type pop_value() {
    T tmp(top());
    pop();
    return tmp;
}
```

Insert into class definition of
stack (23.2.4.3):

```
value_type pop_value() {
    T tmp(top());
    pop();
    return tmp;
}
```

---

9) Comment from the German WG member Ulrich Eisenecker:

26.3, paragraph 3:
Note that the bad_alloc exception is not mandated.

---

10) Comment from the German WG member Ulrich Eisenecker:

26.3.2.4, footnote 7:
If this class can be instantiated from the programmer, then it
should be explicitly mentioned in the text of that paragraph.

---

11) Comment from the German WG member Udo Mueller

Design of basic_istream and basic_ostream

Problem:
If my understanding of typedef in the draft is correct (namely
that a member typedef in a template still only defines an
alias for a type rather than a type), it is not possible
to instantiate a basic_istream or a basic_ostream with an int
data type, or more general any elementary type other than char.
I do not like that, e.g. on my machine wide characters are ˝typedefed
to int.
This "feature" may be intended, as there are requirements for the
default constructor of the character container. If so, there ˝should
at least be a remark about this in the introduction.

Solution:
If the problem exists and the need for having int wide characters
(without a class wrapper) is accepted, there are several ˝solutions, e.g.

- removing the shift operator for the parameter itself. As this has to be a specialization anyway, it should not be too much additional work, to define it as a global operator.
- making all the shift operators for the elementary types global. May be a good thing anyway, to reduce the differences between the shift operators for elementary types and classes.

---

12) Comment from the German WG member Udo Mueller
Editorial issues chapter 20

20.3.6.1 ff; p. 20-11
Why is the protected section the first section int the declaration of class binderlist ? Would it not be better to generally start with the public section?

20.4.1.1; p. 20-15;
The argument "hint" in the description of the allocator member function is not explained. In the meantime a box has been added, which at least gives an idea, and the name has disappeared from 20.4.1.1, but still is present in 20.4.1.

In the former version there was a difference between n==1 and ˝n>1, which fortunately has disappeared. Nevertheless this function should also be allowed to use new[] to allocate an array.

20.4.1.3; p. 20-16;
class heap_allocator is not declared correctly.

---

12) Comment from the German WG member Udo Mueller
Editorial issues chapter 27

27.1.2.6; p. 27-3
What does the word "compatible" mean in this context? It might help to give a complete definition.

27.2; p. 27-4
The forward declaration of basic_ios, basic_istream and ˝basic_ostream leaves out the second template argument, namely the traits. In my understanding of template default arguments this is an error. If my understanding is wrong, this is at least bad style.

27.3.1, p. 27-5
What does the term "associate" mean? The relation between cin, ˝cout, etc. and the corresponding stdio should be defined more ˝precisely.

27.3.1, p. 27-5, footnote 224
Just for the sake of completeness, would a rewording like the following not be better?

Constructors and destructors for static objects can access these objects to read from cin and stdio or write to cout, cerr, clog, stdout and stderr.

27.4; p. 27-6

The forward declaration of ios_traits seems wrong to me:
It has the form of the forward declaration of a specialization.

27.4.2; p. 27-8
If streampos has been moved to Annex D, why not wstreampos also? Anyway Box 125 states a more important inconsistency: A deprecated feature used throughout the standard.

27.4.2.3; p. 27-10
The return value of get_pos is pos_type(pos); I guess it might say fpos. But what is the meaning of the argument state?

27.4.3; p. 27-10
The meaning of T1...T4 should be explained in a similar way as in ˝the iomanip synopsis in 27.6

27.4.3.3; p. 27-17
It might help understandability to add a reference to 22.1.1.5 for the term classic "C" locale.

27.4.3.4; p. 27-17
The explanation of xalloc is very short and uses the term index, which was only explained in note 27.4.3 as: for the sake of exposition. It should be made clear, that this is also used to define the semantics of this function (and the following two functions), so the word exposition may not be appropriate.

27.4.3.4; p. 27-17
Footnote 226 is dangling

27.4.2.5; p. 27-18
There seems to be one auxiliary too much in the description of ˝the effects of the constructor ios_base().

27.4.4; p. 27-18
The operators bool and ! have no semicolon at the end of the line; the same applies to their description in 27.4.4.3.

27.4.4.1; p. 27-19
Why must the members be initialized using init? Would it not be better to leave the place of the initialization to the ˝implementor:

Default constructor and a call to init must have the same postcondition as the constructor taking an argument.

27.4.4.2;p. 27-20
The term "synchronized with" used to explain "tied to" does still not seem to be precise enough to me. It may be worth explicitly defining the "syncronisation of an input and an output sequence". There is a definition of that term in ˝27.5.2.4.2 on page 27-31, which still is incomplete(c.f. the box).

27.4.5.3; p. 27-24
Footnote 229 is dangling.

27.5.2.3.1; p. 27-30
If in gbump we have xnext>xend-n+1, the result of gbump may be undefined with the straight forward implementation.
It seems worth to me, to state that explicitly.

27.6.1.1; p. 27-35
The first two shift operator >> in the "Formatted input" section have no semicolon at the end; the same applies to the first operator in 27.6.1.2.2

What will happen, if char_type == int? Or more general, any sort
of typedef to an elementary type?
As far as my understanding goes the code will break. See problem
above.

The declaration of the last shift operator is not consistent with
the explanation in 27.6.1. The header uses

```
basic_istream<charT,traits>& operator>>
  (basic_streambuf<char_type,traits>* sb);
```

while the explanations uses

```
basic_istream<charT,traits>& operator>>
  (basic_streambuf<charT,traits>* sb);
```

27.8.1.5; p. 27-70
The openmode in the headers should be iosbase::openmode,
shouldn't it? The same applies to the explication in 27.8.1.6
---
14) Comment from the German WG member Ulrich Breymann
Wrong Specifications for Logical Iterator Conditions

25.3.3.1 "lower_bound"

To be corrected:
1. The case i == first leads to a contradiction. (range [first, ˝i) ?)
2. i == last is not specified.
Answer to Box 116: yes! or to be more exact:

Proposal:
========
lower_bound returns either
a) first, such that for any iterator j in the range (first, ˝last) the
following conditions hold:
(value <= *j) // '<=' introduced for readability
resp. if only T=LessThanComparable is required:
(value < *j) || !(*j < value)
or
comp(value, *j) || !comp(*j, value)
or

b) last, such that for any iterator j in the range [first, last)˝the
following conditions hold:
(*j < value) or comp(*j, value) == true
or

c) the furthermost iterator in the range (first, last) such ˝that for
any iterator j in the range (first, i) the following.... " ˝etc.
(rest as before)

BETTER Proposal:
===============
The specifications of logical conditions have been proved as ˝error
prone. The statement:
"Effects: lower_bound returns the first position into which ˝value
can be inserted without violating the ordering"
is clear, sufficient, and easy to understand. Therefore the
specifications for logical iterator conditions should simply be
deleted. They do not contain any new infomation and bloat the ˝standard
document unnecessarily.

25.3.3.2 "upper_bound"

To be corrected:
1. The case i == first leads to a contradiction. (range [first, ˝i) ?)
2. The case i == last is not specified, but must be allowed!

Answer to Box 117: yes!

In addition, the conditions specified are too weak and could be ˝more precise.

Proposal:
========

upper_bound returns either

a) first, such that for any iterator j in the range (first, ˝last) the following conditions hold:
(value < *j) or comp(value, *j) == true
or

b) last, such that for any iterator j in the range [first, last)˝the following conditions hold:
(*j <= value) // '<=' introduced for readability
resp. if only T=LessThanComparable is required:
(*j < value) || !(value < *j)
or
comp(*j, value) || !comp(value, *j)
or

c) the furthermost iterator i in the range
(first, last) such that for any iterator j in the range (i,˝last)
the following corresponding conditions hold:
(*j > value) // '>' introduced for readability
resp. if only T=LessThanComparable is required:
!(*j < value) && (value < *j)
or
!comp(*j, value) && comp(value, *j)

BETTER Proposal:
================
The specifications of logical conditions have been proven as ˝error prone. The statement:
"Effects: upper_bound returns the furthermost position into ˝which value can be inserted without violating the ordering"
is clear, sufficient, and easy to understand. Therefore the specifications for logical iterator conditions should simply be deleted. They do not contain any new infomation and bloat the ˝standard document unnecessarily.

Ref: WP Draft of Sept 26th, 1995
Subject: Wrong Table
20.1.1 Equality Comparison, p. 20-1
'post-condition' is not suitable here as headline. Because ˝the statement does not imply any change, there is no need to differentiate between pre- and post-condition
Proposal: replace 'post-condition' by 'comment'
This is true for a lot of other tables, too, e.g. Table 40, p. ˝20-2, where sometimes the meanings of column items are 'post-condition' and somtimes rather 'comment'

20.1.2 Less than comparison

1. see 20.1.1
2. replace '==' by '<'

---

15) Comment from the German WG member Ulrich Breymann
Missing iterator parameter

24.3.2.6.5 inserter (p. 24-23) (editorial)
Missing iterator parameter
correct writing see 24.3.2.5 on the same page.

---

16) not given to ANSI/ISO since the German WG were apposed.

---

17) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
Minor typos

21.1.1.8.3
> basic_string<charT,traits,Allocator>& assign(const charT* ˝s);
>
> Returns: assign(basic_string(s)).

it should return: ˝"assign(basic_string<charT,traits,Allocator>(s))"

21.1.1.9.1
>Returns: ˝find(basic_string<charT,traits,Allocator>(s,n)pos);

"," is missing between "(s,n)" and "pos"

23.2.1
in the declaration of the string constructor for bitsets,
the template format of the definition later on is missing ˝(typo)

23.2.1.2
> bool operator == ...
and
> bool operator != ...
both:
> Returns: "A nonzero value if ..."

should be: Returns: "true if ..."

---

18) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
Editorial note on find_first_of()/find_last_of() instead of ˝find()/rfind()

Introduction:
string::find_first_of(charT,size_type=0)
and find(charT,size_type=0)
do exaclty the same.
Same for find_last_of(charT,size_type=0)
and rfind(charT,size_type=0).
So I propose to introduce a editorial hint
about that.

Proposed Changes:
Introduce in 21.1.1.9.3 after the return statement of
find_first_of(charT,size_type=0):
Notes: find(c) and find(c,pos) do exactly the same.
Introduce in 21.1.1.9.4 after the return statement of
find_last_of(charT,size_type=0):
Notes: rfind(c) and rfind(c,pos) do exactly the same.

---

19) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
Introduce const reference bitset::operator[] () const

Introduction:
For constant bitsets a possibility to use operator[]
for element acces is missing.
So I propose to introduce one.

Proposed Changes:
Insert into class definition of
bitset (23.2.1):
const reference operator[](size_t pos) const; // for b[i];

---

20) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
Specify "#include <iterator>"

Introduction:
For all container their header synopsis specifies which
additional header files get included.
E.g. in 23.3:
> Header <map> synopsis
>
> #include <memory> // for allocator
> ...
As all container work with iterators, they should also
include <iterator>.

So I propose for all container headers which use iterators
to specify:
#include <iterator> // for iterators

This belongs to:
21.1: Header <string> synopsis
23.2: Header <deque> synopsis
23.2: Header <list> synopsis
23.2: Header <queue> synopsis
23.3: Header <map> synopsis
23.3: Header <set> synopsis

---˝

21) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
capacity() and reserve()

Introduction:
In the C++ library exist two "container" which have the member
functions cpacity() and reserve(), namely string and vector.
Their meaning is unclear described or even not
senseful described:
- capacity() returns
"the size of allocated storage"
- reserve() enlarges capacity if necessary and ensures that
"It is guaranteed that no reallocation takes place during
the insertion that happend after reserve() til the time
when the size of the string/vector reaches the size
spezified by reserve".

The meaning seems not to be quite clear:
- What does the return value of capacity() and the parameter
for reserve() mean ? Is it the size of the storage or the
logical number of elements/chars ?
- If it is the size of the storage, does it include eos for ˝strings ?
- What does "reaches the size" mean? This seems not to be exact,
because it has to become greater than the specified ˝size
for reserve().
So i propose to describe exactly what is meant.

Proposed Changes:
Change the return value description for capacity()
into the following:
For string (21.1.1.6):
Returns:
the number of char-like objects in the string,
for which the size of the allocated storage is enough for.
For vector (23.2.5.4):
Returns:
the number of elements in the vector,
for which the size of the allocated storage is enough for.

Change the last sentence of the description for reserve()
as follows:
For string (21.1.1.6):
It is guaranteed that no reallocation during a insertion
that happens after reserve() takes place until the ˝time
when the size of the string becames greater than the size
specified by reserve().
For vector (23.2.5.4):
It is guaranteed that no reallocation during a insertion
that happens after reserve() takes place until the ˝time
when the size of the vector becames greater than the size
specified by reserve().

---

22) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
move min(), max(), swap() into utility part

Introduction:
STL did introduce some utility functions
for maximum, minimum, and swapping.
As this functions are not algorithms but
"only" general helpful functions,
I propose to move them to into the proper places.

Proposed Changes:
Move
template<class T> const T& min(const T& a, const T& b);
template<class T, class Compare>
const T& min(const T& a, const T& b, Compare comp);
template<class T> const T& max(const T& a, const T& ˝b);
template<class T, class Compare>
const T& max(const T& a, const T& b, Compare comp);
from 25 into the utility section.

Move
template<class T> void swap(T& a, T& b);
from 25 into the utility section.

Move
template<class ForwardIterator1, class ForwardIterator2>
void iter_swap(ForwardIterator1 a, ForwardIterator2 ˝b);
from 25 into the iterator section.

---

23) Comment from the German WG member Nicolai Josuttis ˝(nico@bredex.de):
Move bitset section

Introduction and Proposed Changes:
The bitset section is placed in the manual inside
the sequences section. This seems to say that it
is a sequenceable STL container.
As it is not a STL container I suggest to move the section
after all other containers as section number 23.4.

---

24) Comment from the German WG member Ulrich Breymann:
Ref: WP Draft of 26th January 1996
Subject: Wrong Example / Editorial issue

24.1.6 Iterator tags,
implementation of function __reverse (top of page 24-7)

wrong:
```
  Distance n;
```

Proposal:
```
  Distance n=0;
```

Reason: n is *undefined* in the example, because
distance(first, last, n) *adds* something to n

---

25) Proposal from the German WG member Roland Hartinger

Access to Exception Object's Type Information
see Doc No: X3J16/96-0052, WG21/N0870, core-6404
