Library WP changes for London

diff -rc copyof-1/lib-algorithms librar-1/lib-algorithms
*** copyof-1/lib-algorithms Tue Jul 15 17:24:18 1997
***************
*** 555,561 ****
 .CW "if (\f6pred\fP(*\f6first\fP)){...}" .
The function object
 .CW pred
! is assumed not to apply any non-constant
function through the dereferenced iterator.
This function object may be a pointer to function,
or an object of a type with an appropriate function call operator.
--- 555,564 ----
 .CW "if (\f6pred\fP(*\f6first\fP)){...}" .
The function object
 .CW pred
! ." 25 [lib.algorithms]
! ." CD2 comment UK 687
! ." Steve Rumsby
! shall not to apply any non-constant
function through the dereferenced iterator.
This function object may be a pointer to function,
or an object of a type with an appropriate function call operator.
***************
*** 575,588 ****
 as its argument and \f6first1\fP and \f6first2\fP as
its iterator arguments, it should work correctly in
the construct
! .CW "if (\f6pred\fP(*\f6first1\fP, *\f6first2\fP)){...}" .
 .CW BinaryPredicate
always takes the first iterator
type as its first argument, that is, in those cases when
 .CW "T \f6value\fP"
is part of the signature, it should work
correctly in the context of
! .CW "if (\f6pred\fP(*\f6first1, \f6value\fP)){...}" .
\f6binary_pred\fP shall not
apply any non-constant function through the dereferenced iterators.
 .P
--- 578,594 ----
 as its argument and \f6first1\fP and \f6first2\fP as
its iterator arguments, it should work correctly in
the construct
! ." 25 [lib.algorithms]
! ." CD2 comment UK 688
! ." Steve Rumsby
! .CW "if (\f6binary_pred\fP(*\f6first1, *\f6first2\fP)){...}" .
 .CW BinaryPredicate
always takes the first iterator
type as its first argument, that is, in those cases when
 .CW "T \f6value\fP"
is part of the signature, it should work
correctly in the context of
! .CW "if (\f6binary_pred\fP(*\f6first1, \f6value\fP)){...}" .
\f6binary_pred\fP shall not apply any non-constant function through the dereferenced iterators.

---

\**\ **

*** 594,600 ***

they do not have to be defined.

In these cases the semantics of

\CW a+n

! is the same as that of

\Cb

\{
    X tmp = a;
    advance(tmp, n);
\}

---

*** 600,609 ***

they do not have to be defined.

In these cases the semantics of

\CW a+n

! \" 25 [lib.algorithms]

! \" CD2 comment UK 689

! \" Steve Rumsby

! is the same as that of

\Cb

\{
    X tmp = a;
    advance(tmp, n);
\}

---

*** 605,614 ***

\CW a-b

is the same as of

\Cb

\{
    Distance n;
    distance(a, b, n);
    return n;
\}

\Ce

\"----------------------------------------------------------------------

---

\H2 "Non-modifying sequence operations" lib.alg.nonmodifying

--- 614,627 -----

\CW a-b

is the same as of

\Cb

\{
    Distance n;
    distance(a, b, n);
    return n;
\}

\Ce

\"----------------------------------------------------------------------

---

\H2 "Non-modifying sequence operations" lib.alg.nonmodifying

---

*** 1491,1505 ***

such that for each non-negative integer

\CW \" i < (\f6last\fP - \f6first\fP)\" the following assignment takes place:

\Cb

\* (\f6first\fP + i) = *(\f6result\fP + (i + (\f6middle\fP - \f6first\fP)) % (\f6last\fP - \f6first\fP))

\Ce

\eN

! Should this be:

\Cb
*(\f4result\fP + i) = *(\f4first\fP + (i + (\f6middle\fP - \f6first\fP)) \% (\f6last\fP - \f6first\fP))
\Ce
! .nE
.La Returns:
.CW "result + (\f6last\fP - \f6first\fP)"
.La Requires
--- 1504,1521 ----

such that for each non-negative integer
.CW "i < (\f6last\fP - \f6first\fP)"

the following assignment takes place:
! ./" [lib.alg.rotate] 25.2.10
! ./" Library issue 25-001
! ./" Steve Rumsby
! ./".Cb
! ./" **(\f6first\fP + i) =  *(\f6result\fP + (i + (\f6middle\fP - \f6first\fP)) \% (\f6last\fP - \f6first\fP))
! ./" .Ce
! ./" .eN
! ./" Should this be:
.Cb
*\(\text{result + (middle - first)}\mod(last - first)\)
\Ce
! ./".nE
.La Returns:
.CW "result + (last - first)"
.La Requires
***************
*** 1542,1557 ****

(where \f6n\fP is a positive argument of type
.CW iterator_traits<RandomAccessIterator>::difference_type )
returns a randomly chosen value
! of type
.CW iterator_traits<RandomAccessIterator>::difference_type )
in the interval
.CW "[0, \f6n\fP)"
! .eN
! Can it accept an argument that yields a result of a type
! that, although different from
! .CW RandomAccessIterator::difference_type ,
! can be converted to it?
! .nE
./"===
.H3 "Partitions" lib.alg.partitions
./"-----
--- 1558,1576 ----

(where \f6n\fP is a positive argument of type
.CW iterator_traits<RandomAccessIterator>::difference_type )
returns a randomly chosen value
! ./" [lib.alg.random.shuffle] 25.2.11
! ./" Library issue 25-002
! ./" Steve Rumsby
! ./".Cb
! ./" Can it accept an argument that yields a result of a type
! ./" that, although different from
! ./" .CW RandomAccessIterator::difference_type ,
! ./" can be converted to it?
! ./" .nE
./"===
The furthermost iterator in the range `\[first, last\)`
such that for any iterator \(j\) in the range `\[first, last\)`
\[1899,1905\]

--- 1918,1927 ----
The furthermost iterator
\(i\) in the range
\[first, last\]
\[1908,1919\]

--- 1930,1944 ----
Complexity:
At most \(\log(last - first) + 1\)
\[\#j < value\]
or
\[\text{comp}(j, value) \neq \text{false}\]

--- 1931,1937 ----
Complexity:
At most \(\log(last - first) + 1\)
\[\#j < value\]
or
\[\text{comp}(j, value) \neq \text{false}\]

--- 1938,1944 ----
Complexity:
At most \(\log(last - first) + 1\)
\[\#j < value\]
or
\[\text{comp}(j, value) \neq \text{false}\]

--- 1938,1944 ----
Complexity:
At most \(\log(last - first) + 1\)
\[\#j < value\]
or
\[\text{comp}(j, value) \neq \text{false}\]

--- 1938,1944 ----
Complexity:
At most \(\log(last - first) + 1\)
\[\#j < value\]
or
\[\text{comp}(j, value) \neq \text{false}\]

--- 1940,1944 ----
Complexity:
At most \(\log(last - first) + 1\)
\[\#j < value\]
or
\[\text{comp}(j, value) \neq \text{false}\]

--- 1943,1944 ----
+ pre: \f5i\fP,\f5j\fP are not iterators into \f5a\fP.
+ .br
inserts the elements from the
range \&\f5[i,\&\f5]\fP\& into the container.
T} T{
***************
*** 1050,1057 ****
." 94-0171/N0558
 template <class InputIterator>
     void assign(InputIterator first, InputIterator last);
!     template <class Size, class T>
!       void assign(Size n, const T& t = T());
 allocator_type get_allocator() const;
.Ce
.Cb
--- 1071,1080 ----
." 94-0171/N0558
 template <class InputIterator>
     void assign(InputIterator first, InputIterator last);
! ." CD2 comment Sweden _23/b
! ." CD2 comment Sweden _23/c
! ." Steve Rumsby
!     void assign(size_type n, const T& t);
! allocator_type get_allocator() const;
.Ce
.Cb
***************
*** 1091,1097 ****
 void push_back(const T& x);
.Ce
.Cb
! iterator insert(iterator position, const T& x = T());
!     insert(iterator position, size_type n, const T& x);
 template <class InputIterator>
     void insert (iterator position, InputIterator last);
--- 1114,1122 ----
 void push_back(const T& x);
.Ce
.Cb
! ." CD2 comment Sweden _23/b
! ." CD2 comment Sweden _23/c
! ." Steve Rumsby
!     iterator insert(iterator position, const T& x);
!     insert(iterator position, size_type n, const T& x);
 template <class InputIterator>
     void insert (iterator position, InputIterator last);
***************
*** 1189,1195 ****
 insert(begin(), first, last);
 .Ce
 .Pb
! template <class Size, class T> void assign(Size n, const T& t = T());
 .Pe
 .La Effects:
 .Cb
--- 1214,1223 ----
 insert(begin(), first, last);
 .Ce
 .Pb
! ." CD2 comment Sweden _23/b
! ." CD2 comment Sweden _23/c
! ." Steve Rumsby
! void assign(size_type n, const T& t);
Effects:

--- 1217,1223 ----
"[deque] [insert]

iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);

--- 1245,1253 ----
"[deque] [insert]

! iterator insert(iterator position, const T& x);
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x);
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x);
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x);
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x);
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, iterator insert(iterator position, const T& x);
void insert(iterator position, size_type n, const T& x);

--- 1324,1331 ----

" 94-0171/N0558
template <class InputIterator>
void assign(InputIterator first, InputIterator last);
! template <class Size, class T>
! void assign(Size n, const T& t = T());
allocator_type get_allocator() const;

--- 1354,1363 ----
" 94-0171/N0558
template <class InputIterator>
void assign(InputIterator first, InputIterator last);
! CD2 comment Sweden _23/b
! CD2 comment Sweden _23/c
! Steve Rumsby
! void assign(size_type n, const T& t);
allocator_type get_allocator() const;

--- 1362,1368 ----

void pop_back();

--- 1394,1402 ----

void pop_back();

! CD2 comment Sweden _23/b
! CD2 comment Sweden _23/c
! Steve Rumsby
!

--- 1403,1412 ----
**template <class InputIterator>**

```cpp
void insert(iterator &position, InputIterator first, last);
```

**Effects:**

```cpp
void assign(size_type n, const T& t);
```

**Complexity:**

Constant time.

---

```
void assign(size_type n, const T& t);
```

and

```
insert(iterator &position, size_type n, const T& x = T());
```

---

```
insert(iterator &position, const T& x);
```

---

```
template <class Size, class T> void assign(Size n, const T& t = T());
```

Invalidates all iterators and references to the list

```
and
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
Invalidates only the iterators and references to the spliced element.

La Requires:

\texttt{\textbf{CW} i}

is a valid dereferenceable iterator of

\texttt{\textbf{La Complexity:}}

Constant time if

\texttt{\textbf{CW} \textbackslash f6x\textbackslash fP == this} ;

***************

*** 2065,2071 ****

\textbackslash f6// \_lib\_vector\_modifiers\_modifiers: \textbackslash fP\&

void push\_back(const T\& x);
void pop\_back();

\textbackslash ! iterator insert(iterator position, const T\& x = T());
void insert(iterator position, size\_type n, const T\& x);

\textbackslash template <class InputIterator>
\textbackslash void insert(iterator position, InputIterator first, InputIterator last);

\textbackslash ! \textbackslash CD2 comment Sweden_23/b
\textbackslash ! \textbackslash CD2 comment Sweden_23/c
\textbackslash ! \textbackslash Steve Rumsby

\textbackslash ! \textbackslash CD2 comment Sweden_23/b
\textbackslash ! \textbackslash Steve Rumsby

***************

*** 2080,2089 ----

\textbackslash f6// \_lib\_vector\_modifiers\_modifiers: \textbackslash fP\&

template <class InputIterator>
void assign(InputIterator first, InputIterator last);

\textbackslash ! template <class Size, class U> void assign(Size n, const U\& u = U());
alloc\_type get\_allocator() const;

\textbackslash Ce
\textbackslashCb

--- 2123,2131 ----

\textbackslash f6// \_lib\_vector\_modifiers\_modifiers: \textbackslash fP\&

void push\_back(const T\& x);
void pop\_back();

\textbackslash ! \textbackslash CD2 comment Sweden_23/b
\textbackslash ! \textbackslash Steve Rumsby

***************

*** 1568,1573 ****

--- 1613,1623 ----

\textbf{CW} "position == i"
or
\textbf{CW} "position == ++i" .
+ ." 23.2.2.4 [lib.list.ops]
+ ." CD2 comment Sweden_23224
+ ." Steve Rumsby
+ .P

" 23.2.2.4 \[lib.list.ops\]
" CD2 comment Sweden_23224
" Steve Rumsby
+ .P

Invalidates only the iterators and references to the spliced element.

La Requires:

\textbf{CW} \textit{i}

is an iterator in the range
\textbf{CW} "[first, last)" .
+ ." 23.2.2.4 [lib.list.ops]
+ ." CD2 comment Sweden_23224
+ ." Steve Rumsby
+ .P

Invalidates only the iterators and references to the spliced elements.

La Complexity:

\textit{Constant time if}

\textbf{CW} \textbackslash f6x\textbackslash fP == this" ;
iterator insert(iterator position, const T& x);
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, InputIterator first, InputIterator last);

***************
*** 2150,2156 ****
insert(begin(), first, last);
.*
Pb
! template <class Size, class U> void assign(Size n, const U& u = U());
.*
Pb
." CD2 comment Sweden _23/b
!" CD2 comment Sweden _23/c
!" Steve Rumsby
! void assign(Size n, const T& t);
.*
Pb
." 94-0171/N0558
template <class InputIterator>
void assign(InputIterator first, InputIterator last);
!" 94-0171/N0558
template <class InputIterator>
void assign(InputIterator first, InputIterator last);
.*
Pb
!" CD2 comment Sweden _23/b
!" CD2 comment Sweden _23/c
!" Steve Rumsby
! iterator insert(iterator position, const T& x = T());
void insert(iterator position, size_type n, const T& x);
template <class InputIterator>
void insert(iterator position, InputIterator first, InputIterator last);
!" 94-0171/N0558
template <class InputIterator>
void assign(InputIterator first, InputIterator last);
! template <class Size, class U> void assign(Size n, const U& u = U());
allocator_type get_allocator() const;
.*
Pb
!" CD2 comment Sweden _23/b
!" CD2 comment Sweden _23/c
!" Steve Rumsby
! void assign(Size n, const T& t);
allocator_type get_allocator() const;
// modifiers:
void push_back(const bool& x);
void pop_back();
!
iterator insert(iterator position, const bool& x = bool());
void insert (iterator position, size_type n, const bool& x);
template <class InputIterator>
    void insert (iterator position, InputIterator first,
                  InputIterator last);
--- 2424,2432 ----
// modifiers:
void push_back(const bool& x);
void pop_back();
!
iterator insert(iterator position, const bool& x);
void insert (iterator position, size_type n, const bool& x);
template <class InputIterator>
    void insert (iterator position, InputIterator first,
                  InputIterator last);
***************
*** 3214,3220 ****
set(InputIterator first, last,
    const Compare& comp = Compare(), const Allocator& = Allocator());
.Pe
! ." Effects: Constructs an empty
 .CW set
    using the specified comparison object and allocator,
    and inserts elements from the range
--- 3284,3292 ----
set(InputIterator first, last,
    const Compare& comp = Compare(), const Allocator& = Allocator());
.Pe
! ." Editorial
! ." Steve Rumsby
! .La Effects: Constructs an empty
 .CW set
    using the specified comparison object and allocator,
    and inserts elements from the range
diff -rc copyof~1/lib-diagnostics libr~1/lib-diagnostics
*** copyof~1/lib-diagnostics Tue Jul 15 17:24:24 1997
--- libr~1/lib-diagnostics Tue Jul 15 22:17:08 1997
***************
*** 75,81 ****
namespace std {
    class logic_error : public exception {
    public:
        logic_error(const string& what_arg="","FP\fP\fwhat_arg\&\fP\f5");
    };
}
.Ce
--- 75,84 ----
namespace std {
    class logic_error : public exception {
    public:
        ." 19.1.1 Class logic_error [lib.logic.error]
        ." CD2 comment Germany _191
        ." Steve Rumsby
        explicit logic_error(const string& what_arg="","FP\fP\fwhat_arg\&\fP\f5");
namespace std {
    class domain_error : public logic_error {
    public:
        domain_error(const string& what_arg = "");
    }
}
---
namespace std {
    class domain_error : public logic_error {
    public:
        explicit domain_error(const string& what_arg = "");
    }
}
---
namespace std {
    class invalid_argument : public logic_error {
    public:
        invalid_argument(const string& what_arg = "");
    }
}
---
namespace std {
    class invalid_argument : public logic_error {
    public:
        explicit invalid_argument(const string& what_arg = "");
    }
}
---
namespace std {
    class length_error : public logic_error {
    public:
        length_error(const string& what_arg = "");
    }
}
---
namespace std {
    class length_error : public logic_error {
    public:
        explicit length_error(const string& what_arg = "");
    }
}
namespace std {
  class out_of_range : public logic_error {
public:
  out_of_range(const string &what_arg);
  }
};

namespace std {
  class runtime_error : public exception {
public:
  runtime_error(const string &what_arg);
  }
};

namespace std {
  class range_error : public runtime_error {
public:
  range_error(const string &what_arg);
  }
};

namespace std {
  class overflow_error : public runtime_error {
public:
overflow_error(const string& \&fP\f6what_arg\&\fP\f5);
};

--- 281,290 ----
namespace std {
  class overflow_error : public runtime_error {
    public:
! " 19.1.8 Class overflow_error [lib.overflow.error]  
! " CD2 comment Germany _191
! " Steve Rumsby
! explicit overflow_error(const string& \&fP\f6what_arg\&\fP\f5);
  }
}

***************
*** 284,290 ****
namespace std {
  class underflow_error : public runtime_error {
    public:
! underflow_error(const string& \&fP\f6what_arg\&\fP\f5);
  }
}

--- 308,317 ----
namespace std {
  class underflow_error : public runtime_error {
    public:
! " 19.1.9 Class underflow_error [lib.underflow.error]  
! " CD2 comment Germany _191
! " Steve Rumsby
! explicit underflow_error(const string& \&fP\f6what_arg\&\fP\f5);
  }
}

***************
*** 1,12 ****
.H1 "Library introduction" lib.library 17
.P
This clause describes the contents of the
! .I "\*C Standard library" ,
! .ix "\*C Standard library"
how a well-formed \*C program makes use of the library, and
how a conforming implementation may provide the entities in the library.
.P
The \*C Standard library provides an extensible framework, and contains
components for:
  language support,
  diagnostics, general utilities, strings, locales,
  containers, iterators, algorithms, numerics, and input/output.
--- 1,18 -----
.H1 "Library introduction" lib.library 17
.P
This clause describes the contents of the
! ." 17 Library Introduction [lib.library]
! ." CD2 comment UK 682
! ." Steve Rumsby
! .I "\*C Standard Library" ,
! .ix "\*C Standard Library"
how a well-formed \*C program makes use of the library, and
how a conforming implementation may provide the entities in the library.
The C Standard Library provides an extensible framework, and contains components for:

- language support,
- diagnostics, general utilities, strings, locales,
- containers, iterators, algorithms, numerics, and input/output.

### 17.1 Definitions

A function, specified as part of the C Standard library, that must be defined by the implementation.

If a C program provides a definition for any reserved function, the results are undefined.

Clause _intro.defs_ defines additional terms used elsewhere in this International Standard.
This subclause describes several editorial conventions used to describe the contents of the \*C Standard library.

These conventions are for describing implementation-defined types (_lib.type.descriptions_), and member functions (_lib.functions.within.classes_).

--- 367,376 ----

This subclause describes several editorial conventions used to describe the contents of the \*C Standard Library.

These conventions are for describing implementation-defined types (_lib.type.descriptions_), and member functions (_lib.functions.within.classes_).

--- 615,621 ----

where \&\_\& is type _wchar_t_ optionally qualified by any combination of _const_.

--- 634,643 ----

where \&\_\& is type _wchar_t_ optionally qualified by any combination of _const_.

--- 717,729 ----

This subclause provides a summary of the entities defined in the \*C Standard library.

Subclause _lib.contents_ provides an alphabetical listing of entities by type, while subclause _lib.headers_ provides an alphabetical listing of library headers.

--- 739,757 ----

All library entities except macros,
This subclause provides a summary of the entities defined in the \*C Standard Library. Subclause _lib.contents_ provides an alphabetical listing of entities by type, while subclause _lib.headers_ provides an alphabetical listing of library headers.

The \*C Standard Library provides definitions for the following types of entities:
- Macros
- Values
- Types
- Templates
- Classes
- Functions
- Objects.

All library entities except macros,
as shown in Table \n+Tn:

**************

*** 789,801 ****
as specified in ISO/IEC 9899:1990 Programming Languages C (Clause 7),
or ISO/IEC:1990 Programming Languages\(\text{em}C\ \text{AMENDMENT 1: C Integrity,}
(Clause 7), as appropriate, as if by inclusion. In the \"C Standard
! library, however, the declarations and definitions (except for
names which are defined as macros in C) are within namespace scope
(_basic.scope.namespace_) of the namespace
.CW std.
.P
Names which are defined as macros in C shall be defined as macros in
! the \"C Standard Library, even if license is granted in C for
implementation as functions.
.N[
the names defined as macros
--- 823,841 ----
as specified in ISO/IEC 9899:1990 Programming Languages C (Clause 7),
or ISO/IEC:1990 Programming Languages\(\text{em}C\ \text{AMENDMENT 1: C Integrity,}
(Clause 7), as appropriate, as if by inclusion. In the \"C Standard
! ." 17.3.1.2 Headers \[lib.headers\]
! ." CD2 comment UK 682
! ." Steve Rumsby
! Library, however, the declarations and definitions (except for
names which are defined as macros in C) are within namespace scope
(_basic.scope.namespace_) of the namespace
.CW std.
.P
Names which are defined as macros in C shall be defined as macros in
! ." 17.3.1.2 Headers \[lib.headers\]
! ." CD2 comment UK 682
! ." Steve Rumsby
! the \"C Standard Library, even if license is granted in C for
implementation as functions.
.N[
the names defined as macros

**************

*** 811,817 ****

.N[
.P
Names that are defined as functions in C shall be defined as
! functions in the \"C Standard library.\*f
.Fs
This disallows the practice, allowed in C, of providing a
"masking macro" in addition to the function prototype. The only
--- 851,860 ----

.N[
.P
Names that are defined as functions in C shall be defined as
! ." 17.3.1.2 Headers \[lib.headers\]
! ." CD2 comment UK 682
! ." Steve Rumsby
! functions in the \"C Standard Library.\*f
.Fs
This disallows the practice, allowed in C, of providing a
"masking macro" in addition to the function prototype. The only

**************

*** 841,847 ****
.ix "hosted implementation"
describes the set of available headers.
.P
! A freestanding implementation has
.ix "freestanding implementation
has an implementation-defined set of headers.
A freestanding implementation has an implementation-defined set of headers.

The entities in the \C Standard Library are defined in headers, whose contents are made available to a translation unit when it contains the appropriate

```
CW #include
```

This subclause describes how a \C program gains access to the facilities of the \C Standard library.

Subclause _lib.using.headers_ describes effects during translation phase 4, while subclause _lib.using.linkage_ describes effects during phase 8 (_lex.phases_).

Entities in the \C Standard Library have external linkage (_basic.link_).

Unless otherwise specified, objects and functions have the default

```
\f5extern "C++"
```

Entities in the \*C Standard Library have external linkage (_basic.link_).

Unless otherwise specified, objects and functions have the default `\f5extern "C++"\fP` linkage (_dcl.link_).

This subclause describes restrictions on \*C programs that use the facilities of the \*C Standard library.

The following subclauses specify constraints on the program's namespace (_lib.reserved.names_), its use of headers (_lib.alt.headers_),

---

The \*C Standard library reserves the following kinds of names:

- Macros
- "implementation-defined"

If a file with a name equivalent to the derived file name for one of the \*C Standard library headers is not provided as part of the implementation, and a file with that name is placed in any of the standard places for a source file to be included
If a file with a name is not provided as part of the implementation, and a file with that name is placed in any of the standard places for a source file to be included, the behavior is undefined.

Clauses _lib.language.support_ through _lib.input.output_ describe the behavior of numerous functions defined by the \*C Standard Library. Under some circumstances, however, certain of these function descriptions also apply to replacement functions defined in the program (_lib.definitions_).

---

**Handler functions** (_lib.language.support_) provide default versions of the three handler functions: new_handler

---

**(95-0023/N0623, as amended in Austin:**
In certain cases (replacement functions, handler functions, operations on types used to instantiate standard library template components), the C Standard library depends on components supplied by a C program. If these components do not meet their requirements, the Standard places no requirements on the implementation.

--- 1286,1295 ---

The C Standard Library depends on components supplied by a C program. If these components do not meet their requirements, the Standard places no requirements on the implementation.

***************

*** 1250,1257 ****

Each of the following statements applies to all arguments to functions defined in the C Standard library, unless explicitly stated otherwise.

LI
If an argument to a function has an invalid value (such

--- 1323,1333 ----

Each of the following statements applies to all arguments to functions defined in the C Standard Library, unless explicitly stated otherwise.

LI
If an argument to a function has an invalid value (such

***************

*** 1327,1333 ****

It is unspecified whether any global functions in the C Standard library are defined as

CW inline
(_dcl.fct.spec_).

--- 1403,1412 ----

It is unspecified whether any global functions in the C Standard Library are defined as
It is unspecified whether any member functions in the \*C Standard library are defined as

classes (with names reserved to the implementation).

Certain classes defined in the \*C Standard Library are derived from other classes!
in the \*C Standard library:

It is unspecified whether a class described in the \*C Standard Library as
derived from another class is derived from that class directly, or through other

classes (with names reserved to the implementation).

Certain classes defined in the \*C Standard Library are derived from other classes!
in the \*C Standard library:

It is unspecified whether a class described in the \*C Standard Library as
derived from another class is derived from that class directly, or through other

Any of the functions defined in the \*C Standard library

can report a failure by throwing an exception of the type(s) described in their
paragraph and/or their
Any of the functions defined in the \*C Standard Library can report a failure by throwing an exception of the type(s) described in their paragraph and/or their

Any of the functions defined in the \*C Standard library that do not have an exception-specification may throw implementation-defined exceptions.

The result of the application of the algorithms in the library to invalid ranges is undefined.

All the categories of iterators require only those functions that are realizable for a given category in
A class or a built-in type satisfies the requirements of an output iterator if the following expressions are valid, as shown in Table 

**Output iterator requirements**

--- 317,330 ----

A class or a built-in type.

--- 982,992 ****

reverse_iterator operator--(int);

--- 992,1005 ----

reverse_iterator operator--(int);

--- 1347,1353 ****

typedef Container container_type;
explicit back_insert_iterator(Container& x);
back_insert_iterator<Container>&
operator=(const typename Container::value_type& value);

--- 1360,1368 ----

typedef Container container_type;
explicit back_insert_iterator(Container& x);
back_insert_iterator<Container>&
CD2 comment Germany_244
Steve Rumsby

operator=(const typename Container::reference value);

back_insert_iterator<Container>& operator*();

Effects:
container.push_back(value);

---

typedef Container container_type;
explicit front_insert_iterator(Container& x);
front_insert_iterator<Container>&
operator=(const typename Container::value_type& value);

Effects:
container.push_back(value);

---

typedef Container container_type;
explicit front_insert_iterator(Container& x);
front_insert_iterator<Container>&
operator=(const typename Container::value_type& value);

Effects:
container.push_front(value);

---

typedef Container container_type;
explicit front_insert_iterator(Container& x);
front_insert_iterator<Container>&
operator=(const typename Container::value_type& value);

Effects:
container.push_front(value);

---

typedef Container container_type;
explicit front_insert_iterator(Container& x);
front_insert_iterator<Container>&
operator=(const typename Container::value_type& value);

Effects:
container.push_front(value);
The class

***************

*** 621,627 ****
    s.flags(o.flags());
    s.imbue(o.getloc());
    s.precision(o.precision());
    ! s << '(' << x.real() << ',' << x.imag() << ')' << ends;
    return o << s.str();
}

.Ce
--- 625,634 ----
    s.flags(o.flags());
    s.imbue(o.getloc());
    s.precision(o.precision());
    ! ." 26.2.6 complex non-member operations [lib.complex.ops]
    ! ." CD2 comment CD2-26-003
    ! ." Steve Rumsby
    ! s << '(' << x.real() << ',' << x.imag() << ')';
    return o << s.str();
}

.Ce
***************

*** 899,909 ****
    template<class T> valarray<bool> operator>= (const valarray<T>&, const T&);
    template<class T> valarray<bool> operator>= (const T&, const valarray<T>&);
    .Ce
    .Cb
-   template<class T> T min (const valarray<T>&);
-   template<class T> T max (const valarray<T>&);
-   .Ce
-   .Cb
    template<class T> valarray<T> abs (const valarray<T>&);
    template<class T> valarray<T> acos (const valarray<T>&);
    template<class T> valarray<T> asin (const valarray<T>&);
--- 906,919 ****
    template<class T> valarray<bool> operator>= (const valarray<T>&, const T&);
    template<class T> valarray<bool> operator>= (const T&, const valarray<T>&);
    .Ce
+ ." 26.3.1 Header <valarray> synopsis [lib.valarray.synopsis]
+ ." CD2 comment Germany _263/
+ ." Steve Rumsby
+ ."
Implementations introducing such replacement types shall provide additional functions and operators as follows:

- for every function taking a `const valarray<T>&`, identical functions taking the replacement types shall be added;

---

Implementations introducing such replacement types shall provide additional functions and operators as follows:

- for every function taking a `const valarray<T>&`, identical functions taking the replacement types shall be added;

---

void operator>>=(const valarray<T>&) const;

---

This function has reference semantics, assigning the value of its argument.
This function has reference semantics, assigning the value of its argument

*** 2154,2160 ****
  void operator>>(const valarray<T>&) const;
 .Ce
 .Cb
 !     void fill(const T&);
 ~gslice_array();
 private:
  gslice_array();
--- 2171,2179 ----
  void operator>>(const valarray<T>&) const;
 .Ce
 .Cb
 !." CD2 comment Germany _263/
 !." Steve Rumsby
 !     void operator= (const T&);
 ~gslice_array();
 private:
  gslice_array();
***************
*** 2256,2262 ****
 .H4 "\&f7gslice_array\fP\& fill function" lib.gslice.array.fill
 .ix "[gslice__array] [fill]"
 .Pb
 ! void fill(const T&);
 .Pe
 .P
 This function has reference semantics, assigning the value of its argument
--- 2275,2283 ----
 .H4 "\&f7gslice_array\fP\& fill function" lib.gslice.array.fill
 .ix "[gslice__array] [fill]"
 .Pb
 !." CD2 comment Germany _263/
 !." Steve Rumsby
 ! void operator= (const T&);
 .Pe
 .P
 This function has reference semantics, assigning the value of its argument
***************
*** 2287,2293 ****
  void operator>>(const valarray<T>&) const;
 .Ce
 .Cb
 !     void fill(const T&);
 ~mask_array();
 private:
  mask_array();
--- 2308,2316 ----
  void operator>>(const valarray<T>&) const;
 .Ce
 .Cb
 !." CD2 comment Germany _263/
 !." Steve Rumsby
 ! void operator=(const T&);
 ~mask_array();
 private:
  mask_array();
***************
*** 2383,2389 ****
This function has reference semantics, assigning the value of its argument to the elements of the array.

--- 2406,2414 ----
This function has reference semantics, assigning the value of its argument to the elements of the array.

***************

*** 2413,2419 ****
void operator>>=(const valarray&lt;T&gt;&amp;) const;
private:
  indirect_array();
*** 2438,2446 ****
void operator>>=(const valarray&lt;T&gt;&amp;) const;
private:
  indirect_array();

***************

*** 2484,2490 ****
int addr[] = {2, 3, 1, 4, 4};
valarray&lt;size_t&gt; indirect(addr, 5);
valarray&lt;double&gt; a(0., 10), b(1., 5);
array[indirect] = b;
results in undefined behavior since element 4 is specified twice in the indirection.
--- 2511,2520 ----
int addr[] = {2, 3, 1, 4, 4};
valarray&lt;size_t&gt; indirect(addr, 5);
valarray&lt;double&gt; a(0., 10), b(1., 5);
results in undefined behavior since element 4 is specified twice in the indirection.
***************

*** 2534,2540 ****
void operator>>=(const valarray&lt;T&gt;&amp;) const;
private:
  indirect_array();

***************
This function has reference semantics, assigning the value of its argument

--- 2564,2572 -----
.H4 "\&f7indirect_array\fP\& fill function" lib.indirect.array.fill
[ix "[indirect__array] [fill]"
.Pb
! "$ CD2 comment Germany _263/
! "$ Steve Rumsby
! " void operator=(const T&);
.Pe
.P
This function has reference semantics, assigning the value of its argument
***************
*** 2600,2606 ****
BinaryOperation \f6binary_op\fP);
.Pe
.La Effects:
! Initializes the accumulator
.CW acc
with the initial value
.CW init
--- 2632,2641 -----
BinaryOperation \f6binary_op\fP);
.Pe
.La Effects:
! ." 26.4.1 Accumulate [lib.accumulate]
! ." Library issue 26-002
! ." Steve Rumsby
! Computes its result by initializing the accumulator
.CW acc
with the initial value
.CW init
***************
*** 2619,2624 ****
--- 2654,2665 -----
difficulty of defining the result of reduction on an empty sequence by always requiring an initial value.
.Pe
.La Requires:
+ ." 26.4.1 Accumulate [lib.accumulate]
+ ." Library issue 26-002
+ ." Steve Rumsby
+ T must meet the requirements of CopyConstructible (_lib.copyconstructible_)
+ and Assignable (_lib.container.requirements_) types.
+ .P
.CW binary_op
shall not cause side effects.
."===
***************
*** 2654,2659 ****
--- 2695,2703 -----
.CW "{first2, first2 + (last - first)) "
in order.
.La Requires:
+ ." 26.4.2 Inner product [lib.inner.product]
+ ." Library issue 26-002
+ ." Steve Rumsby
.CW binary_op1
and
.CW binary_op2
diff -rc copyof~1/lib-strings librar~1/lib-strings
*** copyof~1/lib-strings Tue Jul 15 17:24:48 1997
I "character type", that precede the termination null character type value.

In Table \n+Tn,
X::eq(f,X::eof())\fP\ if \f5X::eq_int_type(e,X::eof())\fP\ is false, otherwise a value \f5f\fP\ such that \\f5X::eq(f,X::eof())\fP\ is false.

--- 239,248 ----

T) constant

--- 239,248 ----

_ X::not_eof(e) _ \f5int_type\fP T{

--- 239,248 ----

\f5X::eq_int_type(X::to_int_type(c),
X::to_int_type(d))\fP; otherwise, yields \\f5true\fP\ if \f5e\fP\ and \f5f\fP\ are both \\f5copies\fP\ of \f5X::eof()\fP\.

T) constant

--- 275,284 ----

\f5X::eq_int_type(X::to_int_type(c),
X::to_int_type(d))\fP; otherwise, yields \\f5true\fP\ if \f5e\fP\ and \f5f\fP\ are both \\f5copies\fP\ of \f5X::eof()\fP\;

--- 275,284 ----

\f5true\fP\ if one of \f5e\fP\ and \f5f\fP\ are both copies of \f5X::eof()\fP\ and the other is not;

--- 275,284 ----

\f5false\fP\ otherwise the value is unspecified.

T) constant

--- 303,312 ----

The struct template

--- 303,312 ----

\f5template<class charT> struct char_traits \{ \};

The struct template

--- 303,312 ----

\f5template<class charT> struct char_traits { };  

shall be provided in the header

--- 303,312 ----

The struct template

--- 303,312 ----

\f5template<class charT> struct char_traits;

shall be provided in the header

--- 303,312 ----

The type

--- 303,312 ----

\f5type\fP\ of the iostream class member functions.

--- 303,312 ----

typedef OFF_T off_type;
The type `std::char_traits<char>` represents a character container type which can hold end-of-file to be used as a return type of the iostream class member functions. If `std::eof()` can be held in `std::char_traits<char>`, then some iostreams operations may give surprising results.

```cpp
typedef OFF_T off_type;
```

--- 340,358 ----

--- 460,466 ****

```cpp
static int compare(const char_type* s1, const char_type* s2, size_t n);
static size_t length(const char_type* s);
static const char_type* find(const char_type* s, int n,
const char_type& a);
static char_type* move(char_type* s1, const char_type* s2, size_t n);
static char_type* copy(char_type* s1, const char_type* s2, size_t n);
--- 486,495 ----

static int compare(const char_type* s1, const char_type* s2, size_t n);
static size_t length(const char_type* s);
static const char_type* find(const char_type* s, size_t n,
const char_type& a);
static char_type* move(char_type* s1, const char_type* s2, size_t n);
static char_type* copy(char_type* s1, const char_type* s2, size_t n);
```

--- 529,534 ****

```cpp
static int compare(const char_type* s1, const char_type* s2, size_t n);
static size_t length(const char_type* s);
```

--- 558,571 ----

```
and
```

```cpp
and
```
static int compare(const char_type* s1, const char_type* s2, size_t n);
static size_t length(const char_type* s);
static const char_type* find(const char_type* s, int n, const char_type& a);
static char_type* move(char_type* s1, const char_type* s2, size_t n);
static char_type* copy(char_type* s1, const char_type* s2, size_t n);

---

static int compare(const char_type* s1, const char_type* s2, size_t n);
static size_t length(const char_type* s);
static const char_type* find(const char_type* s, size_t n, const char_type& a);
static char_type* move(char_type* s1, const char_type* s2, size_t n);
static char_type* copy(char_type* s1, const char_type* s2, size_t n);

***************
*** 546,552 ****

static int compare(const char_type* s1, const char_type* s2, size_t n);
static size_t length(const char_type* s);
static const char_type* find(const char_type* s, int n, const char_type& a);
static char_type* move(char_type* s1, const char_type* s2, size_t n);
static char_type* copy(char_type* s1, const char_type* s2, size_t n);
---

static int compare(const char_type* s1, const char_type* s2, size_t n);
static size_t length(const char_type* s);
static const char_type* find(const char_type* s, size_t n, const char_type& a);
static char_type* move(char_type* s1, const char_type* s2, size_t n);
static char_type* copy(char_type* s1, const char_type* s2, size_t n);

***************
*** 626,631 ****
---

and

respectively.

+ ." [lib.char.traits.specializations.wchar.t] 21.1.4.2 struct
char_traits<wchar_t>
+ ." CD2 comment Germany 21-015
+ ." Steve Rumsby
+ .P
+ The member
+ .CW "eof()"
+ returns
+ .CW WEOF .

--------

.H2 "String classes" lib.string.classes

+.

*************************
*** 821,826 ****
---

References, pointers, and iterators referring to the elements of a
basic_string sequence may be
invalidated by the following uses of that basic_string object:
+ .LI
+ As an argument to non-member functions
+ .CW "swap()"
+ (_lib.string.special_),
+ .CW "operator>>()"
+ (_lib.string.io_), and
+ .CW "getline()"
+ (_lib.string.io_).
+ .LI
+ As an argument to

---
+ .CW "basic_string::swap()" .
+ .LI
+ Calling
+ .CW "data()"
+ and
+ .CW "c_str()"
+ member functions.
+ .LI
+ Calling non-const member functions, except
+ .CW "operator[]()" , "at()" , "begin()" ,
+ .CW "rbegin()" , "end()" ,
+ and
+ .CW "rend()" .
+ .LI
+ Subsequent to any of the above uses except the forms of
+ .CW "insert()"
+ and
+ .CW "erase()"
+ which return iterators, the first call to non-const member functions
+ .CW "operator[]()" , "at()" , "begin()" , "rbegin()" , "end()" ,
+ or
+ .CW "rend()" .
+ .P
+ .N
+ These rules are formulated to allow, but not require,
+ a reference counted implementation.
+ A reference counted implementation must have the same semantics
+ as a non-reference counted implementation.
+ .E[
+ + .Cb
+ + string s1("abc");
+ + string::iterator i = s1.begin();
+ + string s2 = s1;
+ + *i = 'a'; // Must modify only s1
+ + .Ce
+ + .E]
+ + .N]
+ .Cb
namespace std {
    template<class charT, class traits = char_traits<charT>,
    ***************
*** 910,915 ****
--- 1014,1023 ----
    basic_string& append(size_type n, charT c);
    template<class InputIterator>
    basic_string& append(InputIterator first, InputIterator last);
+ ." 21.3  Template class basic_string
+ [lib.basic.string]
+ ." CD2 comment France 22
+ ." Steve Rumsby
+ ." void push_back(const charT);
  .Ce
+ .Cb
    basic_string& assign(const basic_string&);
  ***************
*** 1078,1084 ****
    size()  \\
  capacity()  a value at least as large as \\
  ! get_allocator() \\
  .TE
size() \&\f6rlen\fP\&
capacity() a value at least as large as \f5size()\fP

" 21.3.1 basic_string constructors
[lib.string.cons]
"  CD2 comment France 23
"  Steve Rumsby
"  get_allocator() \&\f6str\fP\&.get_allocator()
.TE
.Te
.Pb
***************
*** 1310,1315 ****
--- 1421,1428 ----
.Pe
.La Returns:
The maximum size of the string.
+ .La Note:
+ See Container requirements table (_lib.container.requirements_).
." 
.ix "[basic__string] [resize]"
.Pb
***************
*** 1372,1379 ****
.CW capacity()
is greater or equal to the argument of
.CW reserve .
! Reallocation invalidates all the references, pointers, and iterators referring
! to the elements in the sequence.
." 
.ix "[basic__string] [clear]"
.Pb
--- 1485,1502 ----
.CW capacity()
is greater or equal to the argument of
.CW reserve .
! " 21.3.3 basic_string capacity [lib.string.capacity]
! " CD2 comment 21-001
! "  Steve Rumsby
! "  Reallocation invalidates all the references, pointers, and iterators referring
! " to the elements in the sequence.
! " 21.3.3 [lib.string.capacity]
! " CD2 comment 21-004
! "  Steve Rumsby
! .La Throws:
! .CW length_error
! if ! .CW \"\f6res_arg\fP > max_size()\" .
." 
.ix "[basic__string] [clear]"
.Pb
***************
*** 1399,1411 ****
const_reference operator[](size_type \f6pos\fP) const;
reference operator[](size_type \f6pos\fP);
.Pe
! .La Effects:
! The reference returned is invalid after any subsequent call to
! .CW c_str() ,
Returns: If 
"&pos& < size()"
--- 1522,1537 ----
const_reference operator[](size_type pos) const;
reference operator[](size_type pos);
Pe
" 21.3.4 basic_string elements access [lib.string.access]
" CD2 comment 21-001
" Steve Rumsby
" La Effects:
" The reference returned is invalid after any subsequent call to
" c_str(),
" or any
" \f1non-\fPconst
" member function for the object.
La Returns:
If 
"&pos& < size()",
***************
*** 1416,1422 ****
the
.CW const
version returns
! \f1const
! .CW traits::eos() .
Otherwise, the behavior is undefined.
.ix "undefined"
"
--- 1542,1550 ----
the
.CW const
version returns
! \f1const
! .CW charT() .
Otherwise, the behavior is undefined.
.ix "undefined"
"
***************
*** 1655,1661 ****
La Returns:
.CW
insert(pos,basic_string<charT,traits,Allocator>(n,c))
.Pb
! iterator insert(iterator \f6p\fP, charT \f6c\fP);
.Pe
! La Requires:
! \f6p\fP is a valid iterator on
--- 1783,1792 ----
La Returns:
.CW
insert(pos,basic_string<charT,traits,Allocator>(\f6n\fP,\f6c\fP))
.Pb
! \" 21.3.5.4 basic_string::insert [lib.string::insert]
! \" CD2 comment 21-005
! \" Steve Rumsby
! iterator insert(iterator \f6p\fP, charT \f6c\fP = charT());
.Pe
La Requires:
**1681,1690****

* CW *this.
  * CW \[f6first\, f6last\] is a valid range.

  Effect:
  inserts copies of the characters in the range
  ! CW \[f6first\, f6last\] before the character referred to by \f6p\.

```
H4 "\&f7basic_string::erase\fP\" lib.string::erase
 ix "[basic__string] [erase]"
```

--- 1812,1826 ----

* CW *this.
  * CW \[f6first\, f6last\] is a valid range.

```
\" 21.3.5.4  basic_string::insert [lib.string::insert]
! \" Library issue 21-018
! \" Steve Rumsby
! \" .La Effects:
! \" inserts copies of the characters in the range
! \" .CW \[f6first\, f6last\] before the character referred to by \f6p\.
! \La Returns:
! \" .CW "insert(\f6p\,basic_string<
 charT,traits,Allocator>(\f6first\,f6last\))"
```

```
H4 "\&f7basic_string::erase\fP\" lib.string::erase
 ix "[basic__string] [erase]"
```

--- 1945,1951 ****

```
1953,1962 ****
```

--- 1951,1962 ----

* CW basic_string
  that designates the same object as
  CW this.

--- 2081,2089 ----

```
\" CD2 comment 21-002
! \" Steve Rumsby
! \" .CW charT().
  .La Requires:
  The program shall not alter any of the values stored in the array.
  Nor shall the program treat the returned value as a valid pointer value
```

--- 2091,2102 ----

```
\" Pb
```

```
\" 2091,2102 ----
  .CW basic_string
  that designates the same object as
```

Begins by constructing a sentry object &k as if by
"basic_istream<charT,traits>::sentry \&k(is)".
If bool(k) is true, it calls

--- 2799,2807 ----

Begins by constructing a sentry object &k as if by
"21.3.7.9 [lib.string.io]
Library issue 21-011"
"basic_istream<charT,traits>::sentry \&k(is, true)".
If bool(k) is true, it calls
diff -rc copyof~1/lib-support librar~1/lib-support
*** copyof~1/lib-support Tue Jul 15 17:24:52 1997
--- librar~1/lib-support Tue Jul 15 21:39:38 1997
***************
*** 92,97 ****
--- 92,102 ----
accepts a restricted set of &type arguments in this International Standard.
&type shall be a POD structure or a POD union (_class_).
+ "18.1 Types [lib.support.types]
+ "CD2 comment CD2-18-001
+ "Steve Rumsby
+ The result of applying the offsetof macro to a field which is
+ a static data member, a function member is undefined.
.Xr
subclause _expr.sizeof_, Sizeof,
subclause _expr.add_, Additive operators,
***************
*** 132,138 ****
that they are usable as integral constant expressions.
.
P
Non-fundamental types, such as _lib.complex_, shall not have specializations.
.
--- 137,146 ----
that they are usable as integral constant expressions.
.
P
18.2.1 [lib.limits]
Library issue 18-008

Non-fundamental standard types, such as

The contents are the same as the Standard C library header
\texttt{<stdlib.h>},
with the following changes:

\begin{itemize}
\item \texttt{abort}\,(void)
\item \texttt{atexit}\,(void \(*\&\f6f\fP\&\))
\end{itemize}

that displaces the default version defined by the
\texttt{\C Standard library}.

\begin{itemize}
\item \texttt{\texttt{bad_alloc}}
\item \texttt{\texttt{bad__alloc}}
\end{itemize}

that displaces the default version defined by the
\texttt{\C Standard library}.

\begin{itemize}
\item \texttt{\texttt{new.delete.single}}
\end{itemize}

that displaces the default version defined by the
\texttt{\C Standard library}.

\begin{itemize}
\item \texttt{\texttt{Effects}}
\end{itemize}

Constructs an object of class
The result of calling .CW what() on the newly constructed object is implementation-defined.

virtual const char* what() const throw();

Returns: An implementation-defined value.

An implementation-defined NTBS.

const char* name() const;

Returns: an implementation-defined value.

An implementation-defined value.

an implementation-defined value.
La Effects:
Constructs an object of class
.CW bad_cast .
+ ." 18.5.2 [lib.bad.cast]
+ ." CD2 comment UK 699
+ ." Steve Rumsby
+ .La Notes:
+ The result of calling
+ .CW what()
+ on the newly constructed object is implementation-defined.
+ .ix "implementation-defined"
 ." 
 .ix "[bad__cast] [bad__cast]"
 .ix "[bad__cast] [operator=]"

La Notes:
The result of calling
.CW what()
on the newly constructed object is implementation-defined.
 ix "implementation-defined"
 .ix "[bad__cast] [bad__cast]"
 .ix "[bad__cast] [operator=]"

La Effects:
Copies an object of class
.CW bad_cast .
- .La Notes:
- The result of calling
- .CW what()
- on the newly constructed object is implementation-defined.
- .ix "implementation-defined"
- .ix "[bad__cast] [what]"
.Pb
virtual const char* what() const throw();
.Pe
La Returns:
An implementation-defined value.
 ix "implementation-defined [bad_cast::what]"
 .La Notes:
The message may be a null-terminated multibyte string
._lib.multibyte.strings_,
--- 1545,1561 ----
La Effects:
Copies an object of class
.CW bad_cast .
."
 .ix "[bad__cast] [what]"
.Pb
virtual const char* what() const throw();
.Pe
La Returns:
An implementation-defined value.
 ix "implementation-defined [bad_cast::what]"
 .La Notes:
The message may be a null-terminated multibyte string
._lib.multibyte.strings_,

La Effects:
Constructs an object of class
.CW bad_typeid .
+ ." 18.5.3 [lib.bad.typeid]
The result of calling `.CW what()` on the newly constructed object is implementation-defined.

**Effects:**
Copies an object of class `.CW bad_typeid`

**Notes:**
The result of calling `.CW what()` on the newly constructed object is implementation-defined.

**Returns:**
An implementation-defined value.

---

**Effects:**
Constructs an object of class `.CW bad_exception`

**Notes:**
The result of calling `.CW what()` on the newly constructed object is implementation-defined.

**Returns:**
An implementation-defined value.
La Effects:
Copies an object of class 
CW bad_exception .
- .La Notes:
- The result of calling
- .CW what()
- on the newly constructed object is implementation-defined.
- .ix "implementation-defined"
."
- .ix "[bad_exception] [what]"
.Pb
virtual const char* what() const throw();
.Pe
La Returns:
! An implementation-defined value.
.i"implementation-defined [bad_exception::what]"
.La Notes:
The message may be a null-terminated multibyte string
(_lib.multibyte.strings_),
--- 1777,1793 ----
La Effects:
Copies an object of class 
CW bad_exception .
."
- .ix "[bad_exception] [what]"
.Pb
virtual const char* what() const throw();
.Pe
La Returns:
!." 18.6.2.1 /5 [lib.bad.exception]
!." CD2 comment UK 664
!." Library issue 18-007
!." Steve Rumsby
! An implementation-defined NTBS.
.i"implementation-defined [bad_exception::what]"
.La Notes:
The message may be a null-terminated multibyte string
(_lib.multibyte.strings_),
--- 299,312 ----
_ X a(b); post: \f5Y(a) == b\fp

_ ! x.construct(p,t) (not used) Effect: \f5new((void*)p) T(t)\f5

_ ! x.destroy(p) (not used) Effect: \f5((T*)p)->~T()\f5

TE
.TE
! It is intended that \&\f5a::allocate\fP\& be an efficient means
of allocating a single object of type \&\f5T\fP\&, even when
\&\f5sizeof(T)\fP\& is small. That is, there is no need for a
container to maintain its own ```free list''.
--- 299,318 ----
_ X a(b); post: \f5Y(a) == b\fp

_
It is intended that \&a.allocate\& be an efficient means of allocating a single object of type \&T\&, even when \&sizeof(T)\& is small. That is, there is no need for a container to maintain its own `free list'.

The library provides basic function object classes for all of the comparison operators in the language (\_\_expr.rel\_, \_\_expr.eq\_).

In all cases, type
\_\_T\_ is convertible to type
\_\_bool\_.

--- 764,774 ----

The library provides basic function object classes for all of the comparison operators in the language (\_\_expr.rel\_, \_\_expr.eq\_).

--- 892,901 ----

```cpp
template <class Predicate>
    class unary_negate
    : public unary_function<Predicate::argument_type, bool> {
public:
    explicit unary_negate(const Predicate& \f6pred\fP);
    bool operator()(const argument_type& \f6x\fP) const;
}
```

--- 892,901 ----

```cpp
template <class Predicate>
    class unary_negate
    : public unary_function<
    typename Predicate::argument_type, bool> {
public:
    explicit unary_negate(const Predicate& \f6pred\fP);
    bool operator()(const argument_type& \f6x\fP) const;
}
```

--- 905,912 ----

```cpp
template <class Predicate>
    class unary_negate
    : public unary_function<
    typename Predicate::argument_type, bool> {
public:
    explicit unary_negate(const Predicate& \f6pred\fP);
    bool operator()(const argument_type& \f6x\fP) const;
```
template <class Predicate>
    class binary_negate
    
    explicit binary_negate(const Predicate& \f6pred\fP);
    bool operator()(const first_argument_type&  \f6x\fP,
    bool operator() (const first_argument_type& \f6x\fP,

--- 916,926 ----

...
Class \textit{binder2nd}\textregistered

```cpp
template <class Operation>
    class binder2nd
        : public unary_function<typename Operation::first_argument_type,
                              typename Operation::result_type> {
    protected:
        Operation op;
        typename Operation::second_argument_type value;
    public:
        binder2nd(const Operation& \textit{x}, const typename
                  Operation::second_argument_type& \textit{y});
        result_type operator()(const argument_type& \textit{x}) const;
};
```

--- 992,1019 ----

\textit{binder1st<Operation>} bind1st(const Operation& \textit{op}, const T& \textit{x});

--- 1012,1018 ----

\textit{binder2nd<Operation>} bind2nd(const Operation& \textit{op}, const T& \textit{x});
binder2nd<Operation> bind2nd(const Operation& \f6op\fP, const T & \f6x\fP);

Returns:

Library issue 20-005

Steve Rumsby

CW "binder2nd<Operation>(\f6op\fP, typename Operation::second_argument_type(\f6x\fP))"

member

public unary_function<T*, S> {
    public:
        explicit mem_fun_t(S (T::*p)());

    S operator()(T* p);
};

S operator()(T* p) const;

mem_fun1_t(S (T::*p)(A));

S operator()(T* p, A x);

S operator()(T* p, A x) const;

mem_fun_ref_t(S (T::*p)());

S operator()(T& p);

member

public binary_function<T*, A, S> {
    public:
        explicit mem_fun1_t(S (T::*p)(A));

    S operator()(T* p, A x);
};

S operator()(T* p, A x) const;

member

public unary_function<T, S> {
    public:
        explicit mem_fun_ref_t(S (T::*p)());

    S operator()(T& p);
};

S operator()(T& p);
public unary_function<T, S> {
    public:
    explicit mem_fun_ref_t(S (T::*p)());
    S operator()(T& p) const;
};

***************
*** 1161,1167 ****
: public binary_function<T, A, S> {
    public:
    explicit mem_fun1_ref_t(S (T::*p)(A));
    S operator()(T& p, A x);
};

***************
*** 1392,1398 ****
namespace std {
    template <class OutputIterator, class T>
    class raw_storage_iterator
    public:
        explicit raw_storage_iterator(OutputIterator p);
};

***************
*** 1601,1608 ****
    template<class Y> auto_ptr(const auto_ptr<Y>& p) throw();
--- 1642,1653 ****
    }
if and only if $\text{fp}$ owned

**template<class Y> auto_ptr(const auto_ptr&lt;Y&gt;& $\text{fp}$) throw();**

if and only if $\text{fp}$ owned

```
template&lt;class Y&gt; auto_ptr(const auto_ptr&lt;Y&gt;& $\text{fp}$) throw();
```

```--- 1666,1677 ----		CW *this
owns
.CW *get()
if and only if $\text{fp}$ owned
.CW *$\text{fp}$ .
.ix "[auto__ptr] [operator=]"
.Pb
```

--- 1666,1677 ----
```