LWG Issue 897 and other small changes to forward_list

This is a set of small bug fixes and minor improvements to `forward_list`. It addresses all the issues in raised in LWG Issue 897, and several other matters.

Thanks to Howard Hinnant for material contributions. Thanks to Matt Austern for reviewing this and making suggestions, and of course for `forward_list` itself!

**General Comments**

**erase_after**

I believe that `erase_after` should return `void`. Returning its argument does nothing (potentially for a price), and confuses the programmer. I think it's better that the programmer notice that `forward_list` is different from other containers (and better not to pass a value through a function unnecessarily). The typical code to erase a selection of elements might be something like this:

```cpp
for (auto i = fl.before_begin();;
    auto next(i); ++next;
    if (next == fl.end()) break;
    if (want_to_erase(*next))
        fl.erase_after(i);
    else ++i;
}
```

**before_begin**

In a number of sections, occurrences of "`before_begin()`" should be "`before_begin()`". This seems like it was a paste error and is probably editorial.

**23.3.3 Class template forward_list [forwardlist]**

**New ¶ after 2**

[ *Note:* Modifying any list requires access to the element *preceding* the first element of interest, but in a `forward_list` there is no (constant time) way to access a preceding element. For this reason ranges that are modified, such as those supplied to erase and splice, must be open at the beginning. - *end note* ]

**Change in class definition:**

```cpp
iterator void erase_after(const_iterator position);
```
iterator void erase_after(const_iterator position, iterator last);

23.3.3.4 forward_list modifiers [forwardlist.modifiers]

¶ 1

None of the overloads of insert_after shall affect the validity of iterators and references, and erase_after shall invalidate only the iterators and references to the erased elements. If an exception is thrown during insert_after there shall be no effect. Inserting n elements into a forward_list is linear in n, and the number of calls to the copy or move constructor of T is exactly equal to n. Erasing n elements from a forward_list is linear time in n and the number of calls to the destructor of type T is exactly equal to n.

¶ 12

12 Effects: insert_after(p, il.begin(), il.end()).

After ¶ 14

iterator void erase_after(const_iterator position);

¶ 17

Remove.

After ¶ 17 - ¶ 20

Note the change to the ranges from half to full open.

... iterator void erase_after(const_iterator position, const_iterator last);

18 Requires: All iterators in the range [position, last) are dereferenceable.
19 Effects: Erases the elements in the range [position, last).
20 Returns: last

23.3.3.5 forward_list operations [forwardlist.ops]

¶ 2

2 Effects: Inserts the contents of x before position, and x becomes empty. ...

¶ 4

4 Complexity: \( O(1) \) if \( \text{distance(x.begin(), x.end())} \)

¶ 6

6 Effects: Inserts the element following i into *this, following position, and removes it from x. Pointers and references to the moved elements of x now refer to those same elements but as members of *this. Iterators referring to the moved elements will continue to refer to their elements, but they now behave as iterators into *this, not into x. The result is unchanged if position == i or position == ++i. Pointers and references to *i continue to refer to this same element but as a member of *this. Iterators to *i (including i itself) continue to refer to the same element, but now behave as iterators into *this, not into x.

New ¶ after 10

Complexity: \( O(\text{distance(first, last)}) \)