Inplace

This topic teaches the inplace, unique, and reverse algorithms by demonstrating the usage including inplace_merge, unique_copy and reverse_copy and explaining the algorithms

- Figure 16.9 demonstrates algorithms `inplace_merge`, `unique_copy` and `reverse_copy`.

```
inplace_merge(a1.begin(),
              a1.begin() + 5,
              a1.end() );
```

- Uses the `inplace_merge` algorithm to merge two *sorted sequences* of elements in the *same* container
- In this example, the elements from `a1.begin()` up to, but *not* including, `a1.begin() + 5` are merged with the elements from `a1.begin() + 5` up to, but not including, `a1.end()`
- Requires its three iterator arguments to be at least *bidirectional iterators*
- A second version of this algorithm takes as a fourth argument a *binary predicate function* for comparing elements in the two sequences

```
unique_copy(a1.cbegin(), a1.cend(),
            back_inserter(results1));
```

- Uses the `unique_copy` algorithm to make a copy of all the unique elements in the sorted sequence of values from `a1.cbegin()` up to, but not including, `a1.cend()`
- The copied elements are placed into `vector results1`
- The first two arguments must be at least *input iterators* and the last must be at least an output iterator.
- We did *not* preallocate enough elements in `results1` to store all the elements copied from `a1`
  - Instead, we use function `back_inserter` (defined in header `<iterator>`) to add elements to the end of `results1`

- The `back_inserter` uses `vector`'s `push_back` member function to insert elements at the end of the `vector`
- Because the `back_inserter` *inserts* an element *rather than replacing* an existing element's value, the `vector` is able to grow to accommodate additional elements
- A second version of the `unique_copy` algorithm takes as a fourth argument a *binary predicate function* for comparing elements for *equality*

```
reverse_copy(a1.cbegin(), a1.cend(),
             back_inserter(results2));
```

- Uses the `reverse_copy` algorithm to make a reversed copy of the elements in the range from `a1.cbegin()` up to, but *not* including, `a1.cend()`
- The copied elements are inserted into `results2` using a `back_inserter` object to ensure that the `vector` can grow to accommodate the appropriate number of elements copied
- Requires its first two iterator arguments to be at least bidirectional iterators and its third to be at least an output iterator

This topic taught the inplace, unique, and reverse algorithms by demonstrating the usage including `inplace_merge`, `unique_copy` and `reverse_copy` and explaining the algorithms