Replace

This topic teaches the replace algorithm by demonstrating the usage and explaining the algorithm

- Figure 16.4 demonstrates replacing values from a sequence using algorithms `replace`, `replace_if`, `replace_copy` and `replace_copy_if`.

```
replace(a1.begin(), a1.end(), 10, 100);
```

- Uses the `replace` algorithm to replace *all* elements with the value `10` in the range `a1.begin()` up to, but *not* including, `a1.end()` with the new value `100`
- The iterators supplied as the first two arguments must be *forward iterators* so that the algorithm can *modify* the elements in the sequence

```
replace_copy(a2.cbegin(), a2.cend(),
             c1.begin(), 10, 100);
```

- Uses the `replace_copy` algorithm to copy *all* elements in the range `a2.cbegin()` up to, but *not* including, `a2.cend()`, replacing *all* elements with the value `10` with the new value `100`
- The elements are copied into `c1`, starting at position `c1.begin()`
- The iterators supplied as the first two arguments must be input iterators
- The iterator supplied as the third argument must be an output iterator so that the element being copied can be *assigned* to the copy location
- Returns an iterator positioned after the *last* element copied into `c1`

```
replace_if(a3.begin(), a3.end(),
           greater9, 100);
```

- Uses the `replace_if` algorithm to replace *all* those elements from `a3.begin()` up to, but *not* including, `a3.end()` for which the *unary predicate function* `greater9` returns `true`
- The value `100` replaces each value greater than 9
- The iterators supplied as the first two arguments must be *forward iterators*

```
replace_copy_if
    (a4.cbegin(), a4.cend(),
     c2.begin(), greater9, 100);
```

- Uses the `replace_copy_if` algorithm to copy *all* elements from `a4.cbegin()` up to, but *not* including, `a4.cend()`
- Elements for which the *unary predicate function* `greater9` returns `true` are replaced with the value `100`
- The elements are placed in `c2`, starting at position `c2.begin()`

- The iterators supplied as the first two arguments must be input iterators
- The iterator supplied as the third argument must be an *output iterator* so that the element being copied can be assigned to the copy location
- Returns an iterator positioned after the *last* element copied into `c2`.

This topic taught the replace algorithm by demonstrating the usage and explaining the algorithm