# Remove

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

- Figure 16.3 demonstrates removing values from a sequence with algorithms `remove`, `remove_if`, `remove_copy` and `remove_copy_if`.

```
auto newLastElement =
    remove(a1.begin(), a1.end(), 10);
```

- Uses the `remove` algorithm to eliminate from `a1` *all* elements with the value `10` in the range from `a1.begin()` up to, but *not* including, `a1.end()`
- The first two iterator arguments must be *forward* iterators
- This algorithm does *not* modify the number of elements in the container or destroy the eliminated elements, but it does move *all* elements that are *not* eliminated toward the *beginning* of the container
- Returns an iterator positioned after the last element that was not removed.
- Elements from the iterator position to the end of the container have *unspecified* values

```
remove_copy
    (a2.cbegin(), a2.cend(),
     c.begin(), 10);
```

- Uses the `remove_copy` algorithm to copy *all* elements from `a2` that do *not* have the value `10` in the range from `a2.cbegin()` up to, but *not* including, `a2.cend()`
- The elements are placed in `c`, starting at position `c.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 *inserted* into the copy location
- Returns an iterator positioned after the last element copied into `vector c`

```
newLastElement =
    remove_if(a3.begin(), a3.end(),
                 greater9);
```

- Uses the `remove_if` algorithm to delete from `a3` *all* those elements in the range from `a3.begin()` up to, but *not* including, `a3.end()` for which our user-defined unary predicate function `greater9` returns `true`
- Function `greater9`
  - returns `true` if the value passed to it's greater than 9
  - otherwise, it returns `false`

- The iterators supplied as the first two arguments must be *forward* iterators
- Does *not* modify the number of elements in the container
  - But it does move to the *beginning* of the container *all* elements that are *not* removed
- Returns an iterator positioned after the last element that was *not* removed
- All elements from the iterator position to the end of the container have *undefined* values

```
remove_copy_if(a4.begin(), a4.end(),
               c2.begin(), greater9);
```

- Uses the `remove_copy_if` algorithm to copy all those elements from `a4` in the range from `a4.cbegin()` up to, but *not* including, `a4.cend()` for which the *unary predicate function* `greater9` returns `true`
- The elements are placed in `c2`, starting at `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 remove algorithm by explaining  and demonstrating the usage