

# Reordering Algorithms

## reverse()

- reverse() takes the elements in an iterator range and reverses their order

```
reverse(v.begin(), v.end());
```

- There is also a `_copy()` version

```
reverse_copy(v.begin(), v.end(), back_inserter(v3));
```

## erase()

- remove() returns an iterator pointing to the first removed element
- To get rid of the removed elements, we call erase() and pass it this iterator

```
vector<int> v {3, 1, 4, 1, 5, 9};  
// Send all elements with value 1 to the back, invalidate them,  
// and return an iterator to the first one  
auto defunct = remove(v.begin(), v.end(), 1);  
// v will now contain {3, 4, 5, 9, ?, ?};  
v.erase(defunct, v.end());  
// v will now contain {3, 4, 5, 9};
```

## remove\_if()

- By default, remove() uses the == operator to determine whether to remove an element
- remove\_if() allows us to provide our own predicate

```
vector<int> v {3, 1, 4, 1, 5, 9};  
auto defunct = remove_if(v.begin(), v.end(),  
                          [](int n) { return (n % 3 == 0); }  
);
```

```
// Destroy these elements
```

```
v.erase(defunct, v.end());
```

## remove\_copy()

- `remove_copy()` will write the non-matching elements to the target container

```
vector<int> v {3, 1, 4, 1, 5, 9};  
remove_copy(v.begin(), v.end(), back_inserter(v3), 1);  
// v3 will now contain {3, 4, 5, 9};
```

- This is equivalent to copying all the elements which are not equal to the value

```
copy_if(v.begin(), v.end(), back_inserter(v3), [] (int i) { return i != 1; } );  
// v3 will now contain {3, 4, 5, 9};
```

## remove\_copy\_if()

- `remove_copy_if()` is similar, but takes a lambda

```
remove_copy_if(v.begin(), v.end(), back_inserter(v3),  
               [](int n) { return (n % 3 == 0); }  
);  
// v3 will now contain {1, 4, 1, 5};
```

- This can also be rewritten using `copy_if`

## unique()

- unique() removes duplicate adjacent elements
- unique() behaves similarly to remove() in that it does not destroy any elements

```
vector<int> v {3, 1, 4, 1, 5, 9};
```

```
// Sort the vector so that duplicate elements are adjacent to each other
```

```
sort(v.begin(), v.end());
```

```
// v now has the values 1, 1, 3, 4, 5, 9
```

```
auto defunct = unique(v.begin(), v.end());
```

```
// v now has the values 1, 3, 4, 5, 9, ?
```

```
v.erase(defunct, v.end());
```

```
// v now has the values 1, 3, 4, 5, 9
```

- By default, unique() uses the == operator to determine whether two elements are the same
- unique\_if() allows us to provide our own predicate

```
unique_if(v.begin(), v.end(),  
         [] (int m, int n) { return (n == m + 1); }  
);
```

- There are also unique\_copy() and unique\_copy\_if() which avoid modifying the container