Lists

- At the end of this topic, you will be able to perform the following with lists
  - create
  - Insert and delete
  - Iterate
  - Create a forward_list
  - splice, push_front, pop_front, pop_back, assign, remove, remove_if, unique, merge, reverse sort, splice_after, swap

- `list` *sequence container* (from header `<list>`)
  - Allows insertion and deletion operations at *any* location in the container
- If most of the insertions and deletions occur at the ends of the container
  - `deque` data structure provides a more efficient implementation.
- Implemented as a *doubly linked list*
  - Every node in the `list` contains a pointer to the previous node in the `list`
  - And to the next node in the `list`
  - Enables support of *bidirectional iterators*
    - Allow the container to be traversed both forward and backward

- Any algorithm that requires *input*, *output*, *forward* or *bidirectional iterators*
  - Can operate on a `list`
- Many `list` member functions manipulate the elements of the container as an ordered set of elements

- C++11 includes `forward_list` sequence container (header `<forward_list>`)
  - Implemented as a *singly linked list*
  - Every node in the list contains a pointer to the next node in the list
- Supports *forward iterators* that allow the container to be traversed in the forward direction
- Any algorithm that requires *input*, *output* or *forward iterators* can operate on a `forward_list`

- Figure 15.13 demonstrates several features of class `list`

```
values.sort(); // sort values
```

- Uses `list` member function `sort` to arrange the elements in the `list` in *ascending order*

- A second version of function `sort` allows you to supply a *binary predicate function*
  - That takes two arguments (values in the list)
  - Performs a comparison and returns a `bool` value indicating whether the first argument should come before the second in the sorted contents
  - This function determines the order in which the elements of the `list` are sorted

```
values.splice(values.end(), otherValues);
```

- Uses `list` function `splice` to remove the elements in `otherValues` and insert them into `values` before the iterator position specified as the first argument

- There are two other versions of this function
  - With three arguments allows one element to be removed from the container specified as the second argument from the location specified by the iterator in the third argument
  - With four arguments uses the last two arguments to specify a range of locations that should be removed from the container in the second argument and placed at the location specified in the first argument.

`values.merge( otherValues );`

- Removes all elements of `othervalues` and inserts them in sorted order into `values`
- Both `list`s must be sorted in the same order before this operation is performed

- A second version of `merge` enables you to supply a *binary predicate function* that takes two arguments (values in the list) and returns a `bool` value
  - The predicate function specifies the sorting order used by `merge`

```
values.pop_front();
```

- Removes the first element in the `list`

```
values.pop_back();
```

- Available for *sequence containers* other than `array` and `forward_list`
- Removes the last element in the `list`

`values.unique();`

- *Removes duplicate elements* in the `list`
- The `list` should be in *sorted* order (so that all duplicates are side by side) before this operation is performed, to guarantee that all duplicates are eliminated

- A second version of `unique` enables you to supply a *predicate function* that takes two arguments (values in the list
  - Returns a `bool` value specifying whether two elements are equal

`values.swap( otherValues );`

- Available to all *first-class containers*)
- Exchanges the contents of `values` with the contents of `otherValues`

```
values.assign(otherValues.cbegin(),
otherValues.cend());
```

- Available to all *sequence containers*
- Replace the contents of `values` with the contents of `othervalues` in the range specified by the two iterator arguments

- A second version of `assign` replaces the original contents with copies of the value specified in the second argument
  - The first argument of the function specifies the number of copies
-

```
values.remove( 4 );
```

- Uses `list` function `remove` to delete all copies of the value `4` from the `list`

- This topic taught you how to and performed the following with lists
  - create
  - Insert and delete
  - Iterate
  - Create a forward_list
  - splice, push_front, pop_front, pop_back, assign, remove, remove_if, unique, merge, reverse sort, splice_after, swap