THE C++ STANDARD LIBRARY

The C++ Standard Library

- **Library of comprised of components required in the C++ language**
  - containers that implement many of the more commonly used ADTs
  - algorithms for manipulating data in containers

IMPORTANT CATEGORIES OF ITEMS

- **Containers**
  - objects, such as a list, that hold other objects.
- **Iterators**
  - provide a way to cycle through the contents of a container.
- **Algorithms**
  - act on containers
  - Example: a sorting algorithm

Types of C++ Standard Containers

- **Container adapters**
  - provide efficient, restricted, position-based access to the collection.
- **Sequence containers**
  - provide efficient sequential access to the collection.
- **Associative containers**
  - provide efficient key-based access to the collection.

The C++ Standard Library

- **Why develop our own containers?**
  - Gives foundation for learning to develop other ADTs
  - Standard containers are not part of a class hierarchy
  - Not all languages provide predefined ADTs
  - You may need to develop your own or enhance existing ones

Common To All Standard Containers

Operations Common to all C++ Standard Library Containers		
Member function	Description	Efficiency
Constructor	Creates a container; sequence containers allow the user to specify an initial size.	Varies
Destructor	Deallocates all entries in the container, and then deallocates the container.	O(*n*)
bool empty()	Returns true if the container is empty (size is 0).	O(1)
uint[1] size()	Returns the number of locations in the container, that is, its capacity.	O(1)
operator=	Assigns entries in the container on the right-hand side to the container on the left-hand side.	O(*n*)
[1] uint is the data type for an unsigned integer.		

Container Adapters

```
stack
queue
priority_queue
```

C++ Standard Library stack Operations	
Member function	Description
value_type& top()	Returns a reference to the top entry on the stack.
void push(value_type& item)	Adds item to the top of the stack.
void pop()	Removes the top entry from the stack.

C++ Standard Library queue Operations	
Member function	Description
value_type& front()	Returns a reference to the front entry in the queue.
value_type& back()	Returns a reference to the last (back) entry in the queue.
void push(value_type& item)	Adds item to the back of the queue.
void pop()	Removes the front entry from the queue.

C++ Standard Library priority_queue Operations	
Member function	Description
value_type& top()	Returns a reference to the front entry in the priority queue.
void push(value_type& item)	Adds item to the priority queue.
void pop()	Removes the front entry from the priority queue.

SEQUENCE CONTAINERS

```
vector
deck
list
forward_list
```

Operations Common to C++ Standard Library Sequence Containers		
Member function	Description	Efficiency
value_type& front()	Returns a reference to the first (front) entry in the container.	O(1)
value_type& back()	Returns a reference to the last (back) entry in the container.	O(1)
void push_back(value_type& item)	Appends item onto the back of the container, and increases the container's capacity by 1.	O(1)
void pop_back(value_type& item)	Removes the back entry from the container, and decreases the container's capacity by 1.	O(1)
void resize(**uint** newSize)	Changes the container's capacity to newSize.	O(*n*)
void clear()	Deallocates all entries in the container, and changes its capacity to 0.	O(*n*)
void insert(**uint** position, value_type& item)	Adds item to the container at the given position and increases the container's capacity by 1.	Varies
void insert(iterator itPosition, value_type& item)	Adds item to the container at the current position of the iterator itPosition and increases the container's capacity by 1.	Varies
void erase(**uint** position)	Removes from the container the entry at the given position, and decreases the container's capacity by 1.	Varies
void erase(iterator itPosition)	Removes from the container the entry at the current position of the iterator itPosition, and decreases the container's capacity by 1.	Varies
iterator begin()	Returns an iterator that begins at the first element of the container.	O(1)
iterator end()	Returns an iterator that begins at the last element of the container.	O(1)
reverse_iterator rbegin()	Returns a reverse iterator that begins at the last element of the container.	O(1)
reverse_iterator rend()	Returns a reverse iterator that begins at the first element of the container.	O(1)

SEQUENCE CONTAINERS

`array`

STL array Operations		
Member function	Description	Efficiency
value_type& front()	Returns a reference to the first (front) entry in the array.	O(1)
value_type& back()	Returns a reference to the last (back) entry in the array.	O(1)
value_type& at(size_type n)	Behaves the same as [ ], but performs a bounds check.	O(1)
void fill(const value_type& val)	Fills all array elements with the value val.	O(*n*)
iterator begin()	Returns an iterator that begins at the first element of the array.	O(1)
iterator end()	Returns an iterator that begins at the last element of the array.	O(1)
reverse_iterator rbegin()	Returns a reverse iterator that begins at the last element of the array.	O(1)
reverse_iterator rend()	Returns a reverse iterator that begins at the first element of the array.	O(1)

Associative Containers

```
set
map
multiset
multimap
unordered_multiset
unordered_mulimap
```

Container	Search Keys	Duplicates?	Storage
set	Entry	No	Binary Search Tree
multiset	Entry	Yes	Binary Search Tree
unordered_set	Entry	No	Hash Table
unordered_multiset	Entry	Yes	Hash Table
map	Separate Key (Entry-Key Pair)	No	Binary Search Tree
multimap	Separate Key (Entry-Key Pair)	Yes	Binary Search Tree
unordered_map	Separate Key (Entry-Key Pair)	No	Hash Table
unordered_multimap	Separate Key (Entry-Key Pair)	Yes	Hash Table

Associative Containers

```
set
map
multiset
multimap
unordered_multiset
unordered_mulimap
```

Operations Common to the C++ Standard Library set and multiset		
Member function	Description	
void clear()	Deallocates all entries in the container, and changes its capacity to 0.	O(*n*)
void insert(value_type& item)	Adds item to the container and increases the container's capacity by 1.	O(log *n*)
void erase(value_type& item)	Removes all entries matching item from the container and adjusts the container's capacity.	O(log *n*)
void erase(iterator& position)	Removes from the container the entry at the current position of the given iterator and adjusts the container's capacity.	O(1)
iterator find(value_type& item)	Returns the iterator referencing item.	O(log *n*)
uint count(value_type& item)	Counts the occurrences of item in the container. For set this is at most 1; it can vary for multiset.	O(log *n*)
iterator lower_bound(value_type& item)	Returns an iterator referencing the first entry not less than item.	O(log *n*)
iterator upper_bound(value_type& item)	Returns an iterator referencing the first entry greater than item.	O(log *n*)
iterator begin()	Returns an iterator that begins at the first element of the container.	O(1)
iterator end()	Returns an iterator that begins at the last element of the container.	O(1)
reverse_iterator rbegin()	Returns a reverse iterator that begins at the last element of the container.	O(1)
reverse_iterator rend()	Returns a reverse iterator that begins at the first element of the container.	O(1)