STL Containers and Container Adaptors: Deque, Set, Map Stack, Queue, Priority_Queue

Revisit STL containers

- Sequence containers
  - C++ Vectors //allow random access, insert data at the end with push_back() (unless using insert() with iterator()
  - C++ Lists //doubly linked, no random access
  - C++ Double-Ended Queues //doubly linked, allow random access
- Associative Containers
  - C++ Bitsets
  - C++ Maps
  - C++ Multimaps
  - C++ Sets
  - C++ Multisets
- Container Adapters
  - C++ Stacks
  - C++ Queues
  - C++ Priority Queues

deque -- *double-ended queue* "deck"

- Sequence container
- Can expand in either direction
- Similar interface as vector, but allow insertion/deletion at the beginning.
- More complex internally. Not a single array, can be scattered in different chunks of storage. The container keeps the necessary information to provide direct access.
- For frequent insertion or removal of elements at positions other than beginning or the end, suggest list.

deque

- Constructors:
  - deque<data_type> deque_name; //empty
  - deque<data_type> deque_name(other_deque);
  - deque<data_type> deque_name(initial_size);
- Methods and operations:
  - push_front(value);
  - pop_front();
  - push_back(value);
  - pop_back();
  - front() //return a reference to the first element in the deque
  - back() //return a reference to the last element in the deque
  - at(index)
  - [index] //deque_name[i]

Example of push_front()

```
#include <iostream>
#include <deque>
int main() {
std::deque <int> mydeque(2, 100);
// std::deque <int>::iterator it; // but use auto
mydeque.push_front(200);
mydeque.push_front(300);
std::cout << "mydeque contains: ";
for(auto it = mydeque.begin(); it != mydeque.end(); ++it)
std::cout << ' ' << *it;
std::cout << '\n';
return 0; }
// mydeque contains: 300 200 100 100
```

Deque random access

```
#include <iostream>
#include <deque>
int main() {
using namespace std;
deque <int> deq;
for(int nCount = 0; nCount < 3; nCount++) {
   deq.push_back(nCount);
   deq.push_front(10 - nCount);
 }
for(int nIndex = 0; nIndex < deq.size(); nIndex++)
   cout << deq[nIndex] << " ";
return 0; }
// Output: 8 9 10 0 1 2
```

Associative Containers

- A set is a container that stores unique keys.
- A multiset allows multiple elements with the same key.
- A map stores key/value pairs. The key is used for sorting and indexing the data, and must be unique.
- A multimap allows multiple elements with the same key.

- Newer C++ standard has unordered_set and unordered_map, which allow faster look up but do no keep a sorted order.

Set

- Sets are containers that store **unique** elements following a specific order.
  https://cplusplus.com/reference/set/set/?kw=set

- Set and multiset are typically implemented using “binary search tree”. We will cover this later in the class.
- Unordered_set are typically implemented using “hash table”, we will also cover this later in the class.
- They are not required to be implemented in this manner, but it tends to match the requirement the best.

Example:

```
int myints[] = {75,23,65,42,13};
std::set<int> myset (myints,myints+5);

for (std::set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)
        std::cout << ' ' << *it;
```

Output: 13 23 42 65 75 (sorted)

- Cannot change set value – elements are always const. Because otherwise may compromise the correct ordering.
- If you want to “change”, erase it, then insert a new one.

- The iterators provided by associative containers are bidirectional iterators.

Example of insert in a set

```
#include <iostream>
#include <set>
#include <vector>
using namespace std ;
int main() {
vector <int> v;
set <int> s;
v.push_back(2);
v.push_back(10);
for(int i = 7; i <= 9; i++)
   s.insert(i);
s.insert (v.begin (), v.end());

for(auto it = s.begin(); it != s.end(); ++it)
cout << *it << " ";
return 0; }
// Output: 2 7 8 9 10     <- sorted!
```

- mySet.erase(val); -- delete all elements with the value val
- mySet.erase(iteratorPos); -- delete the element at position pointed to by iteratorPos
- mySet.erase(iteratorBegin, iteratorEnd); -- delete the elments in the range of [begin, end).

Example of erase on a set

```
#include <iostream>
#include <set>
using namespace std;
int main() {
  set <int> s;
  for(int i = 1; i <= 9; i++)
     s.insert(i);
  s.erase(5);
  auto it = s.begin();
  ++ it;
  it = s.erase(it);
  s.erase(it, s.find(7));
  for (it = s.begin(); it != s.end(); ++it)
    cout << *it << " ";
  return 0; }
// Output: 1 7 8 9
```