# COMSC-200 Lecture Topic 16
# Deques, Lists, And Sets

### Reference
Deitel Ch.21.2-21.5
anaturb.net
Nicolai Josuttis
informit.com

### The `deque` Template Class
a double-ended queue ("deck" or "deek")
uses linked "storage blocks"
  instead of reallocation and copying
fast insertion/deletion at front/back
optimized insertion/deletion in middle
```
#include <deque>
```
declaration:
```
  deque<int> a;
```
member functions (same as `vector`):
- `a.size();` #of accessible array elements
- `a.empty();` true if size is zero
- `a[i]` getter and setter

member functions to add nodes:
- `a.push_front(...)`
- `a.push_back(...)`

member functions to deallocate nodes:
- `a.pop_front()` *void*
- `a.pop_back()` *void*

### The `list` Template Class
a doubly-linked list
efficient for inserting and deleting *in the middle*
less efficient for traversing
use `iterator` to traverse
  way more efficient than `operator[]`
  in fact, `operator[]` *is not supported*
```
#include <list>
```
declarations:
```
  list<int> a, b;
  list<int>::iterator p, q;
```
additional member functions:
- `a.begin();` returns an iterator
- `a.end();` returns a null iterator
- `a.sort();` instead of sort algorithm
- `a.insert(p, ...);` insert at `p`
- `a.erase(p);` erase `p`
- `a.erase(p, q);` erase from `p` up to `q`
  *but not `q` itself*
- `a.remove(...);` remove matching values
- `a.splice(p, b);` move `b`'s values to `a` at `p`
- `a.merge(b);` move's `b`'s values to `a`
  *and resorts: requires `operator<`*

### More Algorithms
`#include <algorithm>` // for the sort function
`std::find(a.begin(), a.end(), ...);` returns iterator to matching value
`std::fill(a.begin(), a.end(), ...);` fills with ...

### `set`: a sorted list (increasing order)
```
set<int> s; ...or, decreasing as follows:
set<int, std::greater<int> > s; // #include <functional>
```
Note: space separates `>` and `>`
the `insert` function needs only one parameter
there can be *no duplicates*
requires `operator<`
  but `greater` requires `operator>` // #include <functional>

### `map`: key/value pairs (increasing key order)
```
map<int, double> m; ...or, decreasing as follows:
map<int, double, std::greater<int> > m; // #include <functional>
```
Note: space separates `>` and `>`

### To Generate A Random Sequence
```
vector<int> a(10); or list or deque
std::generate(a.begin(), a.end(), nextInt);
std::random_shuffle(a.begin(), a.end());
```
using this user-defined function:
```
  int nextInt()
  {
    static int i = 0;
    return ++i;
  }
```

### Sorted Collections
`set`: a sorted list (ascending order)
```
  set<int> s;
```
the `insert` function needs only one parameter
there can be no duplicates
`*(s.begin())` is the first element
`map`: key/value pairs (ascending key order)
```
  map<int, double> m;
```
also, `a.sort(std::greater<Time>());` // #include <functional>

### Custom `set` Sort Order
```
struct compare : public std::binary_function<Time, Time, bool>
{
  bool operator()(const Time& a, const Time& b) const
  {
    return a.h > b.h; // based on increasing hour
  }
};
...
set<Time, compare> s;
a.sort(compare()); // for a vector or deque
```

**An Algorithm For Converting The Case Of A `string`**
1. Include these libraries: `algorithm` and `cctype`.

2. Include this class in your program:

```
struct toLower{char operator()(char c) const {return tolower(c);}};
```

3. Use this statement to conver a string named "s" to lower case:

```
std::transform(s.begin(), s.end(), s.begin(), toLower());
```