Resize Details and Analysis:

What is our resize strategy?

Array Resize Strategy #1:

...total copies across all resizes: _________
...total number of insert operations: _________
...average (amortized) cost of copies per insert: _________

Array Resize Strategy #2:

...total copies across all resizes: _________
...total number of insert operations: _________
...average (amortized) cost of copies per insert: _________

Running Time:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Singly Linked List</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert/Remove at front</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert after a given element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove after a given element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert at arbitrary location</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove at arbitrary location</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A List implementation in std

- std::vector implements a list with dynamic growth
- #include <vector> to use it!
- Documentation widely available, including on CBTF exams

Stack ADT

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Purpose</th>
</tr>
</thead>
</table>

Queue ADT

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Purpose</th>
</tr>
</thead>
</table>

Stack and Queue Implementations
#pragma once
#include <vector>

template <typename T>
class Stack {
public:
   void push(const T & d);
   T pop();
   bool isEmpty();
private:
   std::vector<T> list_;}

#include "Stack.hpp"

Example 1

Example 2

Accessing Every Element in Our List / Queue / [Anything]
Suppose we want to look through every element in our data structure. What if we don’t know what our data structure even looks like?

CS 225 – Things To Be Doing:
1. mp_stickers due today
2. Daily POTDs