**ECE250**

**–**

**Project 1**

**Creating a deque driver using a doubly linked-lists**

Yikai Zhou, UserID: y597zhou

Jan 30th, 2020

1. **Class “Node”:**

* Class defines the properties of nodes in a doubly linked-list that stores an int and points to a previous node and next node
* **Member variables:**
  + Pointer to next node
  + Pointer to previous node
  + Data

**Class “DoublyLinkedList”:**

* Class defines the properties and the full set of valid operations of a doubly linked-list data structures with valid operations such as insertion, deletion, printing, etc.
* **Member variables:**
  + Pointer to head
  + Pointer to tail
  + List size
* **Member functions:**
  + Insert: inserts node at index i
  + Remove: removes node at index i
  + Print\_forward: prints list from head node to tail node
  + Print\_backward: prints list from tail node to head node
  + Get\_size: returns size of list
  + Get\_data\_head: returns data stored in head node
  + Get\_data\_tail: returns data stored in tail node

**Class “Deque”:**

* Class defines the properties and valid operations of a deque data structures such as inserting, removing, clearing, printing, etc. It also encapsulates the low-level details of linked-lists such as the concept of a node
* **Member variables:**
  + Pointer to head
  + Pointer to tail
  + List size
* **Member functions:**
  + enqueue\_front: inserts data entry at the front of deque
  + enqueue\_back: inserts data entry at the back of deque
  + dequeue\_front: removes data entry at the front of deque
  + dequeue\_back: removes data entry at the back of deque
  + print\_forward: prints deque from the front to the back
  + print\_backward: prints deque from the back to the front
  + clear: removes all entries of the deque
  + get\_front: returns data stored in the first data entry
  + get\_back: returns data stored in the last data entry
  + is\_empty: returns true if deque is empty, false otherwise
  + get\_size: returns the size of list

**Class “deque\_empty”:**

* Custom exception class that has the error message “failure”
* **Member function:**
  + What: returns a error message as a string

|  |
| --- |
| **Deque** |
| - my\_deque: DoublyLinkedList\* |
| - enqueue\_front(i: int): void  - enqueue\_back(i: int): void  - dequeue\_front(): void  - dequeue\_back(): void  - print\_forward(): void  - print\_backward(): void  - clear(): void  - get\_front(): int  - get\_back(): int  - is\_empty(): bool  - get\_size(): int |

|  |
| --- |
| **DoublyLinkedList** |
| - head: Node\*  - tail: Node\*  - list\_size: int |
| Insert(index: int, data: int): void  Remove(index: int): void  Print\_forward(): void  Print\_backward(): void  Get\_size(): int  Get\_data\_head(): int  Get\_data\_tail(): int |

|  |
| --- |
| **Node** |
| next: Node\*  previous: Node\*  data: int |

→contains

→contains

Associated

|  |
| --- |
| **Dequeuer-empty** |
| What(): char\* |

1. **Constructors/Destructors/Operator overloading**
   * **Constructors:**
     1. Deque: initializes the pointer by allocating memory for a DoublyLinkedList object
     2. DoublyLinkedList: initializes head and tail pointers to nullptr and size of list with 0
     3. Node: 2 constructors:
        1. The first one takes no parameters and initializes pointers to nullptr and the internal data to 0
        2. the other takes a pointer to a Node object to initialize another Node with the same private members
     4. empty\_deque: using the default constructor from std::exception
   * **Destructors:**
     1. Deque: deallocates memory used by its private member of type DoubleLinkedList
     2. Uses a while loop to deallocate each node that was dynamically allocated
     3. Node and empty\_deque: using default destructors for both classes
   * **No operator overloading is used in this project**
2. **Test cases:**
   * **Test1: enqueue\_front 3,4,5, deque\_front x 4 times**
     1. Expecting success for enqueueing and the last dequeue command to fail since list will be empty
     2. Results: Success, success, success, success, success, success, failure
   * **Test2: print/clear when list is empty**
     1. Expecting nothing from print and success for clear
     2. Results: “ ”, success
   * **Test3: enqueue\_front 4, enqueue\_back 5, enqueue\_front 3, enqueue\_back 6, print, clear, print**
     1. Expecting success for enqueuing and 3 4 5 6 6 5 4 3 and another success for clear and nothing after
     2. Results: success, success, success, success, 3 4 5 6, 6 5 4 3, success, “ ”
   * **Test4: enqueue\_back 3, 4, 5, 6, list\_size, dequeue\_front 4 times**
     1. Expecting successes for enqueueing, getting size = 4, and successes for de-queueing
     2. Results: success, success, success, success, size is 4, success, success, success, success
3. **Performance:**
   * All operations which do not require a for/while loop such as insertion/enqueue, deletion/dequeue, get\_size, is\_empty, get\_front\_data, etc. operate in constant time because these operations are independent of the size of the list. So, T(n) = θ(1).
   * Operations such as print\_forward, print\_backward, clear\_list, and the DoublyLinkedList’s class’s destructor operate in linear time, since as the size of the deque increases by a factor of n, the above operations’ worst cases’ time complexities also increase by the same factor. So, T(n) = θ(n).