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":

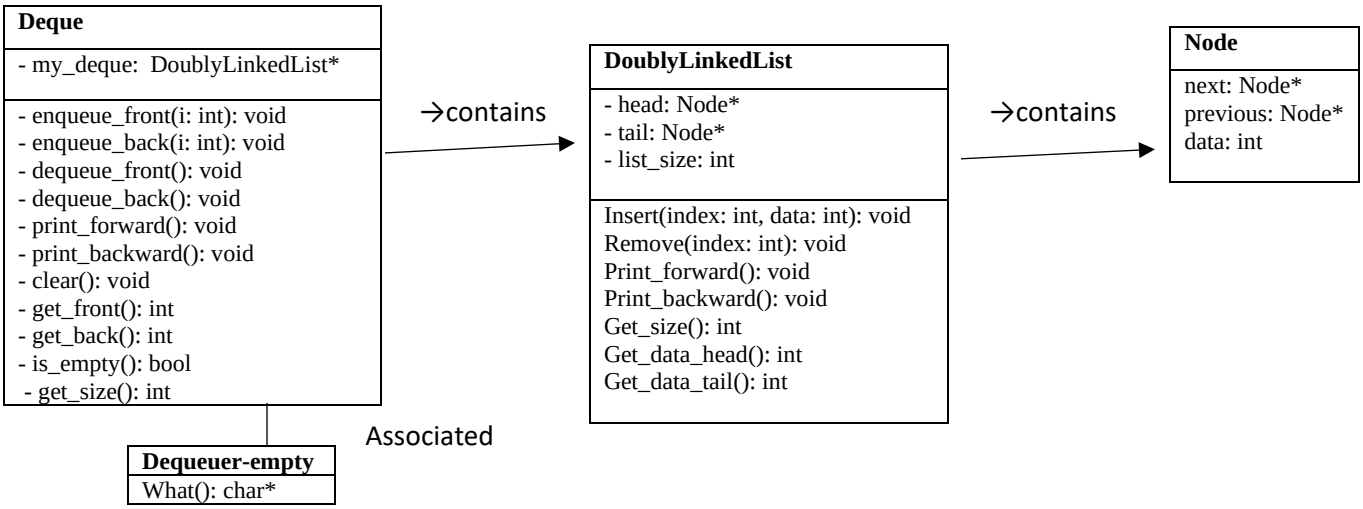
- Class defines the properties and the full set of valid operations of a doubly linked-list data structures with valid operations such as insertion, deletion, printing, etc.
- **Member variables:**
  - Pointer to head
  - Pointer to tail
  - List size
- **Member functions:**
  - Insert: inserts node 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
  - 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

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



2. Constructors/Destructors/Operator overloading

- **Constructors:**
  i. Deque: initializes the pointer by allocating memory for a DoublyLinkedList object
  ii. DoublyLinkedList: initializes head and tail pointers to nullptr and size of list with 0
  iii. 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
  iv. empty_deque: using the default constructor from std::exception

- **Destructors:**
  i. Deque: deallocates memory used by its private member of type DoubleLinkedList
  ii. Uses a while loop to deallocate each node that was dynamically allocated
  iii. Node and empty_deque: using default destructors for both classes
- **No operator overloading is used in this project**

3. Test cases:

- **Test1: enqueue_front 3,4,5, deque_front x 4 times**
  i. Expecting success for enqueueing and the last dequeue command to fail since list will be empty
  ii. Results: Success, success, success, success, success, success, failure

- **Test2: print/clear when list is empty**
  i. Expecting nothing from print and success for clear
  ii. Results: “ ”, success

- **Test3: enqueue_front 4, enqueue_back 5, enqueue_front 3, enqueue_back 6, print, clear, print**
  i. Expecting success for enqueuing and 3 4 5 6 6 5 4 3 and another success for clear and nothing after
  ii. 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**
  i. Expecting successes for enqueueing, getting size = 4, and successes for de-queueing
  ii. Results: success, success, success, success, size is 4, success, success, success, success

4. 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, $T(n) = \theta(1)$.

- 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, $T(n) = \theta(n)$.