ECE 250
Project 1 - Dynamic Deques
May Toyingsirikul, student ID: mtoyings

Overview of Classes

Class 1: Node

Description: A node class where each node stores the address of the data, and data is stored of type string. Each node contains reference to the previous and next node.

Member Variables:

Public Variables:

`value`: the value stored in the node

`prev`: stores the previous pointer node of the current node

`next`: stores the next pointer node of the current node

Class 2: DoublyLinkedList

Description: Generic doubly linked list class with some basic operations of doubly linked list, where each element is stored of type Node class. It provides reference to the head and tail node of the linked list, and contains functions as follows: get head and tail node, set head and tail node, insert a node at the beginning and the end of the list, remove a node from the beginning and the end of the list, find the given element in the list.

Member Variables:

Protected Variables:

`p_head`: a pointer for the first element of the list of a Node type

`p_tail`: a pointer for the last element of the list of a Node type

Member Functions:

`get_head`: an accessor that returns the head pointer

`get_tail`: an accessor that returns the tail pointer

`set_head`: a mutator that set the head pointer

`set_tail`: a mutator that set the tail pointer

`insert_front`: insert an element (node) to the front of the linked list

`insert_tail`: insert an element (node) to the back of the linked list

`remove_head`: remove the head node if not empty

`remove_tail`: remove the last node if not empty

`find`: find the given URL name in the list if it is not empty

Class 3: Deque

Description: A queue type data structure implemented using doubly linked list which is inherited from the DoublyLinkedList class. Performs operations as follows: setting maximum size of the deque, check whether the deque is empty, clear the queue, add elements to the beginning or the end of the deque. It also contains two variables size and max_size, which store the current size of the deque and maximum size of the deque respectively.

Member Variables:

Private Variables:

`size`: current size of the deque

`max_size`: maximum capacity of the deque

Member Functions:

`m`: set the maximum capacity of the deque

`isEmpty`: return whether the deque is empty or not by checking if both head and tail pointer is null

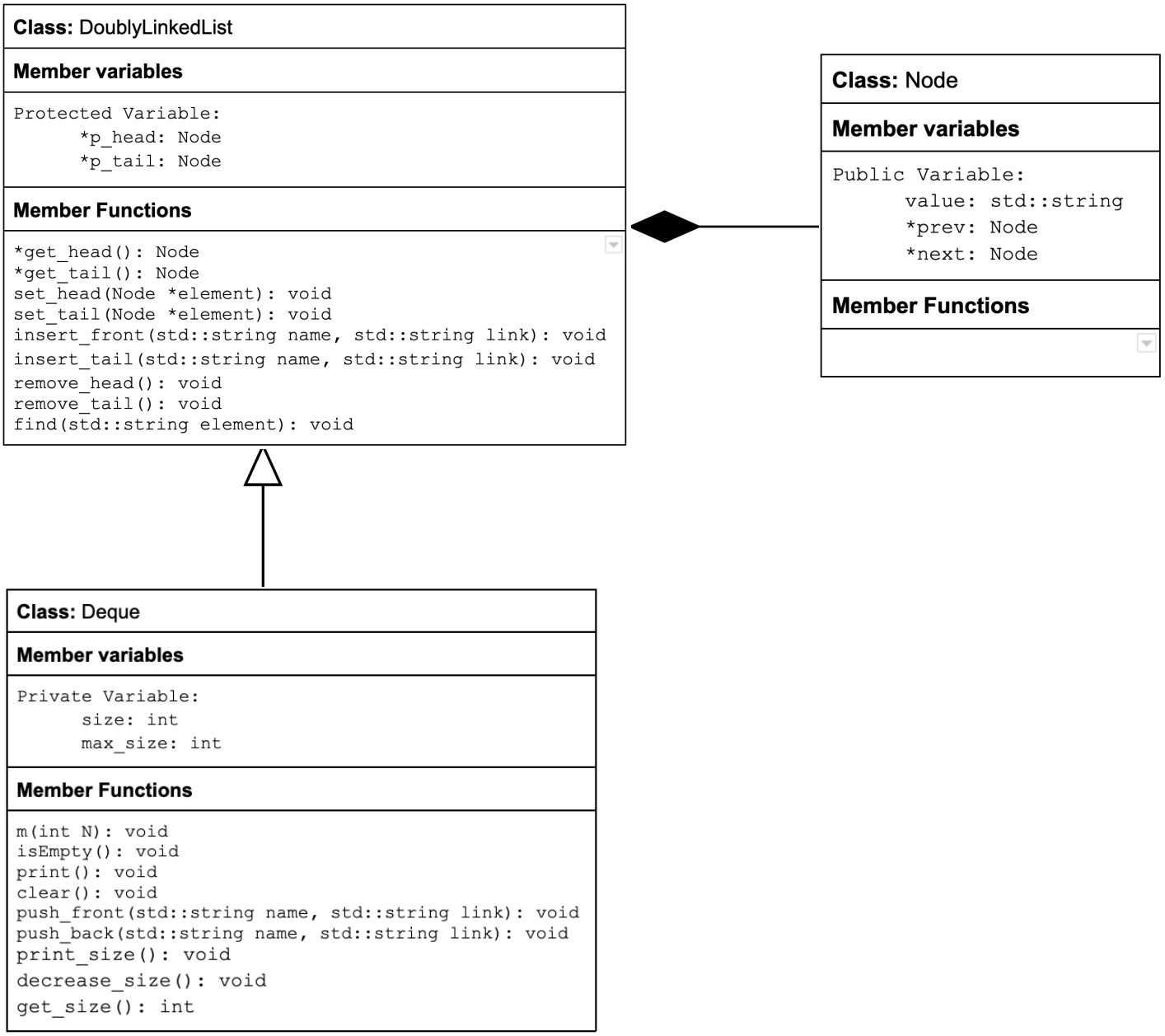
`print:` if the list is not empty, print all entries in deque from from back to front

`clear`: traverse through the deque and delete each node

`push_front`: insert a node to the front of the queue, if the size is full the last node will be popped out

`push_back`: insert a node to the back of the queue, if the list is full the front node will be popped out
`print_size`: print the current size of the deque
`decrease_size`: a mutator that decreases the current size of the list by 1
`get_size`: an accessor that return the current size of the deque

UML Class Diagram



Design Decision Details

Class 1: Node
Constructors: initialise variable `value` at "", and `prev` and `next` Node as `nullptr`
Destructors: set `prev` and `next` Node to `nullptr`

Class 2: DoublyLinkedList
Constructors: initialise both `p_head` and `p_tail` pointer as `nullptr`
Destructors: traverse the list and delete each element in it, then delete `p_head` and `p_tail` Node
Const Parameters: functions with field(s) could be constant since it does not need to be manipulated. Functions with parameter(s) are `set_head`, `set_tail`, `insert_front`, `insert_back`, and `find`.

Class 3: Deque

Constructors: default constructor
Destructors: nothing to be deallocated/destructed
Const Parameters: Each of the function parameters could be constant since it does not need to be manipulated. Functions with parameter(s) are `m`, `push_front`, and `push_back`.

Test Cases

Provided test cases:
Test01.in: create deque of maximum capacity N, exit the code
Test02.in: create, push_front, print, exit
Test06.in: create, push_back, print, push_back, print, push_back, print

Other Test Cases:
Test03.in: create, back, front, find, pop_front, pop_back, size, empty, print, exit. Perform operations on an empty list to test the output of each function when the list is empty.
Test04.in: create, push_front and push_back more than the maximum capacity, print
Test05.in: create, push_front/push_back, find element that is in the deque, find element that is not in the deque
Test07.in: create, push_front/push_back, size, empty, print, clear, empty, print, exit.

Performance Considerations

Since deque is implemented using a doubly linked list, `push_front`, `push_back`, `pop_front`, `pop_back`, `front`, `back`, `empty` could be done in O(1). Note that there is a condition for `push_front` and `push_back` but executing condition is a constant; if the size is already at maximum capacity, popping takes a time constant as well.

The function `m` is a time constant O(1) since it is only assigning value to a variable. Retrieving `size` and `exit` is also a time constant O(1).

The functions `find`, `clear`, and `print`, requires traversing through the entire deque to perform operations therefore the time complexity is O(N)

Reference

https://www.programiz.com/dsa/doubly-linked-list
https://algorithmtutor.com/Data-Structures/Basic/Doubly-Linked-Lists/