Creating Deque Using Linked List
Design Document

Maya Bishop

Jan 26th, 2020

1. Overview of Classes

- What class(es) did you design? - Describe each class in your design; explain the purpose of the class and of their member variables and member functions.

Class:

Deque

Description:

Represents a Deque as a doubly linked list and provides operations associated with deque such as adding, accessing and removing from beginning and end of the deque.

Member variables:

- Pointer to front
- Pointer to end
- Total size of deque

Member functions (operations) - description, parameters and return values

- enqueue_front: Adds element to the front of the deque and returns whether it was successful
- enqueue_back: Adds element to the back of the deque and returns whether it was successful
- dequeue_front: Removes element from the front and returns whether it was successful based off whether there was an item in the deque
- dequeue_back: Removes element from the back and returns whether it was successful based off whether there was an item in the deque
- clear: clears the content of the deque and returns successful once it is empty
- front: access the front element and compares it to the given number, returns whether it was successful based off if the numbers are the same
- back: access the back element and compares it to the given number, returns whether it was successful based off if the numbers are the same
- empty: checks if deques is empty and returns the result
- size: gets the size of the deque
- print: if there are items in the deque, print all items in the deque front to back and then back to front otherwise don't print anything

Class:

Node

Description:

Node for a doubly linked list

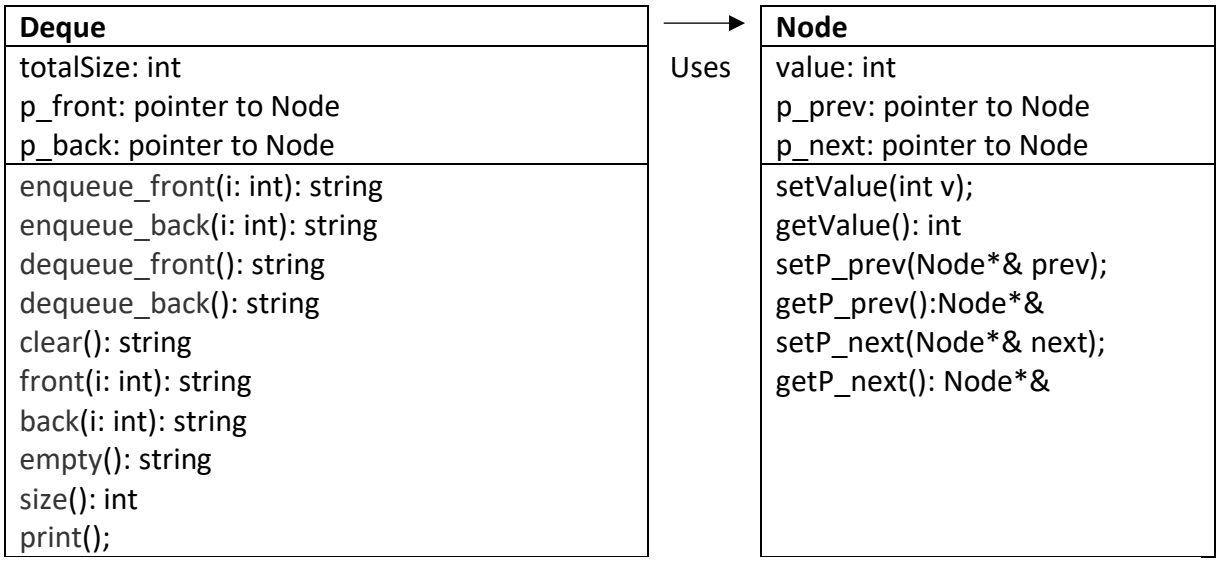
Member variables:

- value for the node
- pointer to next node
- pointer to previous node

Member functions (operations) - description, parameters and return values

- Functions to get and set the above variables from the Node class

UML Class Diagram



2. Constructors/Destructor/Operator overloading

- For each class, what are your design decisions regarding constructors?
- For each class, what are your design decisions regarding destructors?
- Provide your rational for any operators that you needed or decided to override.

Deque:

The constructor will set the original pointers to null and the size to 0. The destructor will call the clear function. I did not find I needed to overwrite any operators for this class

Node:

The constructor will set the original pointers to null and the value to 0. There is also another constructor that sets the pointers and value to passed in parameters. The destructor will set the pointers to null. I did not find I needed to overwrite any operators for this class

3. Test Cases

- Describe your testing strategy for this class. What are some of the test cases you need to consider?

A list of the test cases I tried are adding a value, printing the deque, removing the node, and then printing again. I also checked the size and empty and clear functions when the deque had items in it and when it did not. I added values to the front and back of the deque when it was empty as well as full. I set values and then compared to the front and back of the list with both accurate and in accurate values. I then removed the items and compare again using the front and back commands. I also check the size after adding and removing an item from the deque to make sure it changed.

4. Performance

All of the commands were implemented in O(1) time asides from the clear and print commands as every part of the functions used a constant run time which can be simplified to 1. Clear runs in O(n) time where n is the number of items in the deque as it has to loop through each item in the deque to deallocate that node so that there are no memory leaks. Print also runs in O(n) time even though it loops through the deque in both directions as this results in the time being 2n which has a upper bound of O(n) as there is a const c such that cn is greater than 2n.