Homework 2: Working with Linked List, Stack and Queue

In this assignment you will learn how to work with linked lists and how linked lists can be used to implement other data structures such as stacks and queues.

You have been provided three different linked list implementations: *LinkedList.cpp*, *LinkedListWithTail.cpp* and *LinkedListDoubly.cpp*. The files implement following versions of linked list:

- *LinkedList.cpp* is a singly linked list implementation with a head pointer.
- *LinkedListWithTail.cpp* is a singly linked list with an additional tail pointer.
- *LinkedListDoubly.cpp* is a doubly linked list with head and tail pointer.

You are also provided the following two classes.

- *Stack.cpp,* implements a stack data structure using the *LinkedList* class in *LinkedList.cpp.*

- *Queue.cpp,* implements a queue data structure using the *LingedListWithTail* class in *LinkedListWithTail.cpp.*

For this homework, you are required to add different functionalities to the provided implementation.

Task 1 – 5 are to be added in LinkedList.cpp file.

The file implements a singly linked list using a head pointer that points to the first node of the list. Your job is to add following features.

Task 1: Add insertLast function. Add a new function *insertLast(int item)* to the list. This function will insert a new item at the end of the list.

Task 2: Add insertAfter function. Add a new function *insertAfter(int oldItem, int newItem)* to the list. This function will insert a new item after an existing item in the list. The function will first search for the *oldItem* in the list. Then it will insert the *newItem* after the *oldItem* in the list. If the *oldItem* is not present in the list, then insertion should be discarded.

Task 3: Add getItemAt function. Add a new function *getItemAt(int pos)* that returns a pointer to the item at the position *pos*. Assume the item at the beginning of the list corresponds to position no 1. If *pos* is greater than the length of the list, it will return a NULL_VALUE.

Task 4: Add deleteFirst function. This function will delete the first element of the list. You must ensure that memory of the deleted item is released properly. In case the item is not found in the list, return a NULL_VALUE, otherwise return SUCCESS_VALUE.

Task 5: Add destructor function. The destructor will delete all the items and thereby free all the memory allocated for the list.

Task 6 - 9 is to be added in LinkedListWithTail.cpp file.

The class includes an additional pointer *tail* that should always point to the last node of the list. In the given implementation, *tail* pointer is not set properly in implemented functions: *insertItem* and *deleteItem*. Your job is to add following features.

Task 6: Set tail pointer correctly. Add required codes in *insertItem* and *deleteItem* functions to set up the pointer correctly so that it always points to the last node of the linked list.

Task 7: Make insertLast function efficient. For this task, your job is to use this *tail* pointer to make the *insertLast* function more efficient than your first implementation in Task 1 above, which runs in *O(n)* time. If you have *tail* a pointer, then you will not require searching the whole list to find the end of the list. So, change your previous implementation accordingly so that new version runs in *O(1)* constant time. Ensure that the *tail* pointer is correctly set after insertion.

Task 8: Add getItemAt function. Same as the one implemented for *LinkedList* class. It will return a pointer to the item at the position *pos*. Assume the item at the beginning of the list corresponds to position no 1. If *pos* is greater than the length of the list, it will return a NULL_VALUE.

Task 9: Add deleteLast function. This function will delete the last element of the list. You must ensure that memory of the deleted item is correctly released. In case the item is not found in the list, return a NULL_VALUE, otherwise return SUCCESS_VALUE. Ensure that the *tail* pointer is correctly set after deletion.

Task 10 – 12 is to be added in LinkedListDoubly.cpp file.

The class implements a doubly linked list. In this type of lists, each node has two pointers: *next* and *prev*. The *next* pointer points to the next node in the list while the *prev* pointer points to the previous node in the list. You job is to add following features.

Task 10: Make printListBackward function: Write a function *printBackward* that will print the whole list in reverse order starting from the last node of the list. Use *tail* pointer to start traversing the list backward.

Task 11: Modify insertLast function: Add *insertLast* function to the doubly linked list. You can do this easily by modifying the implementation you did in Task 7. Add codes to set up both *next* and *prev* pointers correctly. Ensure that your function runs in *O(1)* constant time as before.

Task 12: Make deleteLast function efficient: You already implemented *deleteLast* function as part of Task 9 that possibly runs in *O(n)* time. Now, you will implement an improved version of the same function for the doubly linked list. You will observe that using the doubly linked list allows us to implement the *deleteLast* function in *O(1)* time compared to O(n) time in previous implementation.

Task 13 – 14 is to be added in Stack.cpp file.

The file implements a stack using the class *LinkedList*. A stack is a Last-In-First-Out (LIFO) data structure where the last element added will be the first one to be removed. Your job is to add following features. You are not allowed to modify the *LinkedList* class and add any new functionality beyond the ones done as part of tasks 1-5.

Task 13: Add push function. This function will insert an item in the stack.

Task 14: Add pop function. This function will return the most recently inserted item and remove the item from the stack as well. If the stack is empty, it will return NULL_VALUE.

Task 15 – 16 is to be added in Queue.cpp file.

The file implements a queue using the class *LinkedListWithTail*. A queue is a Fast-In-First-Out (FIFO) data structure where the first element added will be the first one to be removed. Your job is to add following features. You are not allowed to modify the *LinkedListWithTail* class and add any new functionality beyond the ones done as part of tasks 6-9.

Task 15: Add enque function. This function will insert an item in the queue.

Task 16: Add deque function. This function will return the least recently inserted item and remove the item from the queue as well. If the queue is empty, it will return a NULL_VALUE.

Assignment: designed by Sukarna Barua, *Assistant Professor, CSE, BUET*
and modified by Khaled Mahmud Shahriar, *Assistant Professor, CSE, BUET*

You must also satisfy the following requirements:

- You must extend the given code.
- You must *free* unused memory where it is required.
- For any clarification and help, contact your teacher over mail or messaging through moodle course page.
- You may be provided an updated and corrected version of this document later if It is required.
- *You must not use other's code. You must not share your code. You must not copy from any other sources such as web, friends, relatives, etc. In all cases, you will earn a 0 and will move closer to getting an "F" grade in the course.*

Assignment: designed by Sukarna Barua, *Assistant Professor, CSE, BUET*
and modified by Khaled Mahmud Shahriar, *Assistant Professor, CSE, BUET*