Homework 2: Playing with LinkedList

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.

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

Task 1 – 4 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 insertBefore function. Add a new function $insertBefore(int\ oldItem, int\ newItem)$ to the list. This function will insert a new item before an existing item in the list. The function will first search for $oldItem$ in the list. Then it will insert the $newItem$ immediately before the $oldItem$ in the list. If the $olditem$ is not present in the list, then insertion should be done at the beginning of the list.

Task 3: Add deleteAfter function. Add a new function $deleteAfter(int\ oldItem)$ to the list. This function will delete the item immediately after the given parameter $item$. The function will first search for input $item$ in the list. Then it will delete the item immediately after the $item$ in the list. If the $item$ is not present in the list, then no deletion should be done.

Task 4: 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.

Task 5 - 6 is to be added in LinkedListWithTail.cpp file.

The file 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 5: Set $tail$ pointer correctly. Add required codes in $InsertItem$ and $deleteAfter$ functions you implemented as part of Task 1-4 to set up the $tail$ pointer correctly so that it always points to the last node of the linked list.

Task 6: 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 a $tail$ 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 7 – 9 is to be added in LinkedListDoubly.cpp file.

The file 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. A function $insertFirst$ is already implemented in the file. This function inserts an item at the beginning of the linked list. A print function $printListForward$ is also implemented. Your job is to add following features.

Task 7: 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 8: Modify deleteAfter function: Add $deleteAfter$ function to the doubly linked list. You can do this easily by modifying the implementation you did in Task 5. Add codes to set up both $next$ and $prev$ pointers correctly. Ensure that your function runs in $O(1)$ constant time as before.

Task 9: Make deleteLast function efficient: You already implemented $deleteLast$ function as part of Task 4 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.

You must also satisfy the following requirements:

- You must extend the given code.
- You cannot use any function of C library except $malloc$ and input output functions.
- You cannot use object oriented programming.
- You must *free* unused memory where it is required.
- For any clarification and help, contact your teachers.
- 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.*