**CSE 4304-Data Structures Lab. Winter 23-24**

**Batch:** CSE 22

**Date**: October 02, 2024

**Target Group:** All

**Topic**: Linked Lists

**Instructions**:

* Regardless of how you finish the lab tasks, you must submit the solutions in Google Classroom. In case I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab took place.
* Task naming format: fullID\_T01L01\_2A.c/cpp
* If you find any issues in the problem description/test cases, comment in the Google Classroom.
* If you find any tricky test cases that I didn’t include but that others might forget to handle, please comment! I’ll be happy to add them.
* Use appropriate comments in your code. This will help you to recall the solution in the future easily.
* Obtained marks will vary based on the efficiency of the solution.
* Do not use <bits/stdc++.h> library.
* Modified sections will be marked with BLUE color.
* You are allowed to use the STL stack unless it’s specifically mentioned to use manual functions.

| **Group** | **Tasks** |
| --- | --- |
| 2A | 1 2 5 6 |
| 1B | 1 2 5 6 |
| 1A |  |
| 2B |  |
| **Assignments (all groups)** |  |

**Task-01**:

Implement the basic operations using a ‘Singly Linked list’. Your program should include the following functions:

1. **Insert\_front**(int key):
   * Insert the element with the ‘’ at the beginning of the list.
   * Time Complexity: O(1)
2. **Insert\_back**(int key):
   * Insert the element with the ‘’ at the end of the list.
   * **Time Complexity: O(1)**
3. **Insert\_after\_node** (int key, int v):
   * Insert a node with the ‘’ after the node containing the value ‘’ if it exists. (shows error message otherwise).
   * Time complexity:
4. **Update\_node** (int key, int v):
   * Looks for the node with value *v* and updates it with the new value ‘key’ (error message if the node doesn’t exist)
   * Time complexity: O(n)
5. **Remove\_head** ():
   * Remove the first node from the linked list.
   * Time complexity: O(1)
6. **Remove\_element** (int key):
   * Removes the node containing the ‘key’ if it exists (else throw an error message).
   * Time complexity O(n)
7. **Remove\_end** ():
   * Remove the last node from the linked list.
   * Time complexity: O(n)

**Input format**:

* The program will offer the user the following operations (as long as the user doesn’t use option 7):
  + Press 1 to insert at front
  + Press 2 to insert at back
  + Press 3 to insert after a node
  + Press 4 to update a node
  + Press 5 to remove the first node
  + Press 6 to remove a node
  + Press 7 to remove the last node
  + Press 8 to exit.
* After the user chooses an operation, the program takes necessary actions (or asks for further info if required).

**Output format**:

* After each operation, the status of the linked list is printed with the head and tail nodes.

| **Sample input** | **Sample Output** |
| --- | --- |
| 1 10 | Head=10, Tail=10, 10 |
| 7 | Head=Null, Tail=Null, Enmpy |
| 7 | Underflow  Head=Null, Tail=Null, Enmpy |
| 6 10 | Value Not found  Head=Null, Tail=Null, Enmpy |
| 5 | Head=Null, Tail=Null, Enmpy |
| 5 | Underflow  Head=Null, Tail=Null, Enmpy |
| 2 20 | Head=20, Tail=20, 20 |
| 1 30 | Head=30, Tail=20, 30 20 |
| 2 40 | Head=30, Tail=40, 30 20 40 |
| 3 50 20 | Head=30, Tail=40, 30 20 50 40 |
| 3 60 40 | Head=30, Tail=60, 30 20 50 40 60 |
| 5 | Head=20, Tail=60, 20 50 40 60 |
| 7 | Head=20, Tail=40, 20 50 40 |
| 4 70 50 | Head=20, Tail=40, 20 70 40 |
| 4 80 50 | Value Not found  Head=20, Tail=40, 20 70 40 |
| 4 80 40 | Head=20, Tail=80, 20 70 80 |
| 4 90 20 | Head=90, Tail=80, 90 70 80 |
| 6 70 | Head=90, Tail=80, 90 80 |
| 6 70 | Value Not found.  Head=90, Tail=80, 90 80 |
| 3 100 90 | Head=90, Tail=80, 90 100 80 |

**Note**: You must follow the prescribed input-output format. Otherwise, 50% marks will be discarded.

**Task 2**

* Satisfy the requirements of Task 1 using a ‘**Doubly linked list**’.
* One additional requirement is that you must print the linked list twice after each operation:
  + From head to tail.
  + From the tail towards the head (don’t use recursive implementation; rather, utilize the ‘previous’ pointers).
* The **Remove\_end()** function should be done in **O(1)**

**Task 5**

Given two sorted linked lists in increasing order, your task is to create a **new linked list** that stores the item that intersects in both lists.

The first two lines represent the two lists (input stops with -1). Store them in a linked list, and show their intersection as output. Print ‘empty’ if there is no intersection.

| **Input** | **Output** |
| --- | --- |
| 1 2 3 4 5 6 -1  3 4 5 7 8 -1 | 3 4 5 |
| 10 20 30 40 50 60 70 80 90 -1  1 5 10 15 30 25 40 43 77 80 99 -1 | 10 30 40 80 |
| 2 2 3 3 3 5 5 5 5 5 9 9 9 -1  0 1 2 3 4 5 6 7 -1 | 2 3 5 |
| 1 2 3 4 5 -1  6 7 8 9 -1 | Empty |
| 6 7 8 9 -1  1 2 3 4 5 -1 | Empty |

**Task 6** Implementing the basic operations of **Stack with Linked lists**

Stacks is a linear data structure that follows the Last In First Out (LIFO) principle. The last item to be inserted is the first one to be deleted. For example, you have a stack of trays on a table. The tray at the top of the stack is the first item to be moved if you require a tray from that stack.

The Insertion and Deletion of an element from the stack are a little bit different from the traditional operation. We define the two corresponding operations as Push() and Pop() from the stack.

The first line contains *N* representing the size of the stack. The lines contain the ‘function IDs’ and the required parameter (if applicable). Function ID 1, 2, 3, 4, 5, and 6 corresponds to push, pop, isEmpty, isFull, size, and top. The return type of inEmpty and isFull is Boolean. Stop taking input once given -1.

| **Input** | **Output** |
| --- | --- |
| 10  3  2  1 10  1 20  5  1 30  6  2  1 40  1 50  4  1 60  4  5  1 70  5  2  6  -1 | True  Underflow  10  10 20  2  10 20 30  30  10 20  10 20 40  10 20 40 50  False  10 20 40 50 60  False  5  10 20 40 50 60 70  6  10 20 40 50 60  60 |