CSE 4304-Data Structures Lab. Winter 2022-23
Batch: CSE 21
Lab: 07
Date: September 18, 2023.
Target Group: All
Topic: Linked List, Binary Trees

Instructions:

- Regardless of finishing the lab tasks, you must submit the solutions in the Google Classroom. If I forget to upload the tasks there, CR should contact me. The deadline will always be 11:59 PM on the day the lab has occurred.
- Task naming format: fullID_T01L07_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 case that I didn't include but others might forget to handle, please comment! I'll be happy to add.
- Use appropriate comments in your code. This will help you to easily recall the solution in the future.
- 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.

Group	Tasks
2A	1 2 3
2B	1 2 3
1B	1 2 3
1A	
Assignment	**The tasks not covered in your lab**

<u>Task-01</u>:

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 'key' at the beginning of the list.
   - Time Complexity: O(1)
2. **Insert_back**(int key):
   - Insert the element with the 'key' at the end of the list.
   - **Time Complexity: O(1)**
3. **Insert_after_node** (int key, int v):
   - Insert a node with the 'key' after the node containing the value 'v' if it exists. (shows error message otherwise).
   - Time complexity: $O(n)$
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 3

Implement the basic operations of a 'Binary tree'. Your program should include the following functions:

1. **Insert**:
    - Assuming each node contains a unique value.
    - Input starts with a number N (representing the number of nodes), followed by N lines containing the info of the node.
    - Every line contains three values 'data', 'parent', and '1(left child) or 2(right child)'.
    - The second and third parameters for the root are Null (0).
    - After successful insertion, print the entire tree using preorder traversal. Beside each node, print the info of its parent as well.
2. **Removal**: if an internal node is removed, its subtrees are also removed. Print 'Not Found' if the node is not in the tree.
3. **Search**: Returns the node if it is present and prints its description. Otherwise, print 'Not Found'.
4. **Height**: Returns the height of the tree, where the height of a leaf node is 0.

Input	Output	Explanation

7
1 0 0
2 1 1
3 1 2
5 2 2
4 2 1
6 3 1
7 3 2
	1(N) 2(1) 4(2) 5(2) 3(1) 6(3) 7(3)
	(preorder)
		Note: The tree looks as follows:
		      1
		    / \
		   2 3
		  / \ / \
		 4 5 6 7

3 3	Present, Left(6), Right(7)	(search 3)
2 6	1(N) 2(1) 4(2) 5(2) 3(1) 7(3) (preorder)	(remove 6)
3 1	Present, Left(2), Right(3)	(search 1)
3 5	Present, Left(null), Right(null)	
3 6	Not present	
3 3	Present, Left(null), Right(7)	
4	2	
2 3	1(N) 2(1) 4(2) 5(2) (preorder)	
4	2	
3 3	Not present	
2 2	1(N) (preorder)	
4	0	

Note: To connect the child with parent, first find the node that contains the parent, for which, you may use any traversal algorithm that returns a Node pointer. If you manually assign the nodes but satisfy all other requirements, 50% marks.