**Data Structures & Algorithms LAB**

(BSCS-F18 Morning & Afternoon)

**Lab # 6**

**Task # 1**

**Implement a singly linked list class which stores integers in unsorted order. Your class**

**declarations should look like:**

|  |  |
| --- | --- |
| **class LinkedList;**  **class Node {**  **friend class LinkedList;**  **private:**  **int data;**  **Node\* next;**  **};** | **class LinkedList {**  **private:**  **Node\* head;**  **public:**  **LinkedList(); *// Default constructor***  **~LinkedList(); *// Destructor***  **…**  **};** |

Apart from the **default constructor** and **destructor**, the **LinkedList** class should also have the following public member functions:

**bool insert (int val)**

This function should insert a new value (**val**) into the linked list. The time complexity of this function should be **constant** i.e. ***O*(1)**. In other words, there should be no loop in this function.

**void display ()**

This function should display the contents of the linked list on screen.

**int countNodes ()**

This function should determine (and return) the **count** of the nodes present in the linked list.

**Task # 2**

Now, implement the following three public member functions of the **LinkedList** class:

**bool removeLastNode (int& val)**

This function will remove the *last node* from the linked list. Before de-allocating the node, this function should store the data present in that node into the reference parameter **val**. This function should return **false** if the list is empty; otherwise it should remove the last node of the linked list and return **true**.

**bool removeSecondLastNode (int& val)**

This function will remove the *second last node* from the linked list. Before de-allocating the node, this function should store the data present in that node into the reference parameter **val**. This function should return **false** if the list contains fewer than two nodes; otherwise it should remove the second last node of the linked list and return **true**. (*Note:* You are NOT allowed to modify the data of any node in the linked list).

**bool removeKthNode (int k, int& val)**

This function will remove the **k**th element (node) from the linked list. For example, if the linked

list object **list** contains **{4 2 8 1 9 5 4 6}**, then the function call **list.removeKthNode(4)**

should remove the 4th element (node) from the linked list and the resulting **list** should be:

**{4 2 8 9 5 4 6}**. Before de-allocating the node, this function should store the data present in that node into the reference parameter **val**. This function should return **false** if the linked list contains fewer than **k** elements; otherwise it should remove the **k**th node from the linked list and return **true**. (*Note:* You are NOT allowed to modify the data of any node in the linked list).

Also write a drive main function to test the working of each of the above-mentioned functions.

**Task # 3**

Implement the following public member function of the **LinkedList** class:

**void combine (LinkedList& list1, LinkedList& list2)**

This function should combine the nodes of the two linked lists (**list1** and **list2**) into one list.

All the nodes of the first list (**list1**) will precede (come before) all the nodes of the second list

(**list2**).

For example, if **list1** contain **{7 3 4 2}** and **list2** contains **{5 9}**, then after the function call

**list3.combine(list1,list2)**, **list3** should contain **{7 3 4 2 5 9}** and **list1** and **list2**

should be empty now.

***Note:*** *You are NOT allowed to create any new node in this function. You are also NOT allowed*

*to modify the “data” field of any node. You can assume that the* ***LinkedList*** *object on which this*

*function is called is empty at the start of this function.*

Also write a driver main function to test the working of the above function.