**Data Structures & Algorithms LAB – Fall 2015**

(BS-SE-F14 Morning & Afternoon)

# Lab # 7

**Instructions:**

- **Attempt the following tasks exactly in the given order.**
- Make sure that there are no ***dangling pointers*** or ***memory leaks*** in your programs.
- Indent your code properly.
- Use meaningful variable and function names. Follow the naming conventions.
- Use meaningful prompt lines/labels for all input/output.

## Task # 0 *(Pre-Requisite)* *(Max Time: 25 Minutes)*

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:

### `void insert (int val)`

To 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.

### `bool remove (int val)`

This function removes the node containing the *first* occurrence of **`val`** from the linked list. This function should return true if such a node is found (and removed) from the linked list, and it should return false otherwise.

### `void display ()`

This function displays the contents of the linked list on screen.

## Task # 1 *(Max Time: 45 Minutes)*

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 deallocating 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 deallocating the node, this function should store the data present in that node into the variable **`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 4$^{th}$ element (node) from the linked list and the resulting **`list`** should be: **{4 2 8 9 5 4 6}**. Before deallocating the node, this function should store the data present in that node in the variable **`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 # 2 *(Max Time: 20 Minutes)*

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

### `void combine (LinkedList& list1, LinkedList& list2)`

This function will combine the nodes of the two linked lists (**`list1`** and **`list2`**) into one list. All the nodes of the first list (**`list1`**) will precede 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.

## Task # 3 *(Max Time: 25 Minutes)*

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

### `void shuffleMerge (LinkedList& list1, LinkedList& list2)`

This function will merge the nodes of the two linked lists (**`list1`** and **`list2`**) to make one list, by taking nodes alternately from the two lists.

For example, if **`list1`** contains **{2 6 4}** and **`list2`** contains **{8 1 3}**, then after the function call **`list3.shuffleMerge(list1,list2)`**, **`list3`** should contain **{2 8 6 1 4 3}** 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 both lists (which are being merged) have the* ***same number of elements****. You can also 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.

**VERY IMPORTANT**

In the next few lab(s), you may need some or all of the functions from Today's Lab. So, make sure that you have the working implementation of **all** the functions of Today's Lab, when you come to the next lab.