Data Structures and Algorithms LAB – Spring 2022
(BS-IT-F20 Morning & Afternoon)

Lab # 9

Instructions:

- Attempt the following tasks exactly **in the given order**.
- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an **"F"** grade in this course and lab.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Do NOT use any **global** or **static** variable(s). However, global named constants may be used.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.

Task # 1.0 *(Pre-Requisite)* *(Max Time: 0 Minutes)*

In the previous lab, you have implemented the **Singly linked list** class which stores integers in **unsorted order**. Your class declarations look like:

```
class LinkedList;
class Node {
      friend class LinkedList;
  private:
      int data;
      Node* next;
};
```

```
class LinkedList {
  private:
      Node* head;
  public:
      …
};
```

Task # 1.1 *(Max Time: 15 Minutes)*

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

`int removeAll (int val)`

This function should remove all occurrences of **`val`** from the linked list and also return the count of the nodes which are removed from the linked list. For example, if the linked list object **`list`** contains these 8 values **{7 4 1 7 7 3 5 6}**, then the function call:

```
int count = list.removeAll (7);
```

should remove all occurrence of **7** from the linked list and should return 3. The resulting **`list`** should contain these 5 values: **{ 4 1 3 5 6 }**. *In your implementation, you are NOT allowed to call any other member function of the* **`LinkedList`** *class. You are also NOT allowed to modify the "data" field of any node.*

Task # 2.0 *(Pre-Requisite)* *(Max Time: 0 Minutes)*

As discussed in the lecture, you should have the implementation of a class named **CDLLD** for a **Circular Doubly Linked List (with a dummy header node)** which stores integers in **unsorted order**. Your class definitions should look like:

```
class CDLLD;

class DNode {
      friend class CDLLD;
  private:
      int data;
      DNode* next;
      DNode* prev;
};

class CDLLD {            // Circular Doubly Linked List with a Dummy header node
  private:
      DNode head;        // Dummy header node
  public:
      CDLLD ();          // Default constructor
      ~CDLLD ();         // Destructor
      …
};
```

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

`bool insertAtStart (int val)` *Time complexity: $O(1)$*

This function should insert **val** at the start of the linked list.

`bool insertAtEnd (int val)` *Time complexity: $O(1)$*

This function should insert **val** at the end of the linked list.

`void display ()`

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

`int countNodes ()`

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

Task # 2.1 *(Max Time: 15 Minutes)*

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

`void combine (CDLLD& list1, CDLLD& list2)` *Time complexity: $O(1)$*

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 (come before) all the nodes of the second list (**list2**). The time complexity of this function must be **constant** i.e., $O(1)$.

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.

Very Important: *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 CDLLD 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 # 2.2 *(Max Time: 15 Minutes)*

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

`void shuffleMerge (CDLLD& list1, CDLLD& 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.

Very Important: *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) contain the **same number of elements/nodes**. You can also assume that the CDLLD 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 # 2.3 *(Max Time: 25 Minutes)*

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

`void splitList (CDLLD& leftHalf, CDLLD& rightHalf)`

This function will split the list (on which it is called) into two halves, and store these halves in the two linked lists passed into this function. Make sure that all boundary cases are properly handled.

For example, if **`list1`** contains **{8 4 2 9 1 5 3}** and **`list2`** and **`list3`** are empty, then after the function call:

```
list1.splitList(list2,list3);
```

`list2` should contain **{8 4 2 9}** and **`list3`** should contain **{1 5 3}** and **`list1`** should be empty now. *Note that if the list contains an ODD number of elements, then the one extra element should go into the **left half**.*

Very Important: *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 CDLLD objects being passed into this function are empty at the start of this function.*

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

Task # 2.4 *(Max Time: 10 Minutes)*

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

`bool isSorted () const`

This function should determine whether the linked list is **sorted in increasing order** or not. It should return **`true`** if the values of the linked list are sorted in increasing order. Otherwise, it should return **`false`**.

Task # 2.5 *(Max Time: 25 Minutes)*

Implement the following three public member functions of the **CDLLD** class:

`bool removeLastNode (int& val)` *Time complexity: $O(1)$*

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)` *Time complexity: $O(1)$*

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 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 **CDLLD** object **list** contains these 8 values **{4 2 8 1 9 5 4 6}**, then the function call:

```
list.removeKthNode(4,val);
```

should remove the 4th element (node) from the linked list and the resulting **list** should contain these 7 values: **{ 4 2 8 9 5 4 6 }**. 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 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 driver main function to test the working of the above three functions.

VERY IMPORTANT

In the next Lab, you will 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.