Data Structures & Algorithms LAB – Fall 2015

(BS-SE-F14 Morning & Afternoon)

Lab # 9

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 # 1.0 *(Pre-Requisite)* *(Max Time: 10 Minutes)*

In Task # 2 of Lab # 8, you implemented a class named **`CDLinkedList`** for **Circular Doubly Linked List (with a dummy header node)** which stores integers **sorted in ascending order**. In this task, you are required to modify the implementation of **`insert`** and **`remove`** functions of this class, so that the elements of the list are maintained in **unsorted order** now. Your class definitions should look like as shown below:

```
class CDLinkedList;

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

class CDLinkedList {     // An Unsorted Circular Doubly Linked List (with Dummy header)
  private:
      DNode head;              // Dummy header node
  public:
      CDLinkedList();          // Default constructor
      ~CDLinkedList();         // Destructor
      bool insert (int val);   // Insert val at the end of linked list. Time complexity: O(1)
      bool remove (int val);   // Remove the first occurrence of val from the linked list
      void display();          // Display the contents of linked list on screen
      …
};
```

Task # 1.1 *(Max Time: 25 Minutes)*

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

`bool merge (CDLinkedList& list1, CDLinkedList& list2)`

This function will merge the nodes of the two **sorted** circular doubly linked lists (**`list1`** and **`list2`**) to form one sorted list. If the two lists are sorted (in increasing order), this function should merge the nodes of these two lists (as described in the example below) and should return true. If either **`list1`** or **`list2`** is not sorted, this function should not modify any linked list and should return false.

For example, if **`list1`** contains **{4 7 10 12}** and **`list2`** contains **{1 3 6 8 9 15}** and **`list3`** is empty, then after the function call **`list3.merge(list1,list2)`**, **`list3`** should contain **{1 3 4 6 7 8 9 10 12 15}** 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 CDLinkedList object on which this function is called is empty at the start of this function.*

Make sure that all boundary cases are properly handled. Also write a drive main function to test the working of your function.

Task # 1.2 *(Max Time: 25 Minutes)*

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

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

This function will split the list (on which it is called) into two halves, and stores these halves in the two lists passed into this function.

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 number of elements are odd, then the one extra element should go into the left half.*

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 CDLinkedList objects being passed into this function are empty at the start of this function.*

Make sure that all boundary cases are properly handled. Also write a drive main function to test the working of your function.

Task # 1.3 *(Max Time: 15 Minutes)*

Now, you are required to implement another public member function of the **`CDLinkedList`** class. The prototype of your function should be:

```
void mergeSort ()
```

This function should use the **Merge sort** algorithm to **sort the linked list** into **increasing order**. This function should be **recursive**.

Hint: Use the functions **`merge`** and **`splitList`**, which you have implemented in Task # 1.1 and Task # 1.2.

Task # 1.4 *(Max Time: 15 Minutes)*

Implement another public member function of the **`CDLinkedList`** class. The prototype of your function should be:

```
int findMax ()
```

This function should determine and return the maximum integer present in the linked list. If the linked list is empty, this function should return -999. You are required to implement this function **recursively**. This function will call a private workhorse function of the **`CDLinkedList`** class to accomplish its task.

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.