Data Structures & Algorithms LAB – Fall 2015

(BS-SE-F14 Morning & Afternoon)

Lab # 8

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: 0 Minutes)*

In Lab # 7, 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:
      LinkedList();      // Default constructor
      ~LinkedList();     // Destructor
      void insert (int); // Insert at start O(1)
      bool remove (int);
      void display ();
};
```

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

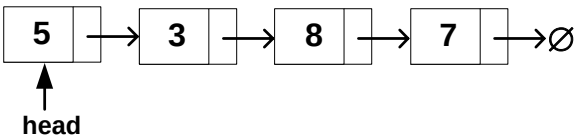
Now, implement the following member function of the above **`LinkedList`** class:

`void reverse ()`

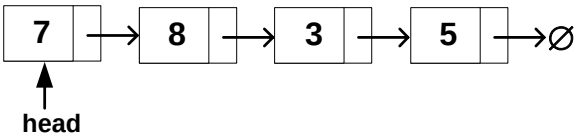
This function should reverse the linked list, iteratively. Keep the following things in mind when implementing this function:

- **You can NOT modify the "data" of any node. So, the reversal is to be done by modifying the links (next pointers) of the nodes in the linked list.**
- You can NOT create any temporary array (or linked list) to perform the reversal.
- The reversal is to be done in one traversal/pass (going from first node to the last node) through the linked list.

For example, if the linked list before calling the **`reverse()`** function is:



Then, after the execution of the **`reverse()`** function the linked list should be:



Note: Use some sample linked lists to see how this function should work. In other words, **do a lot of paperwork before writing the code**. Also, make sure that all the boundary cases are properly handled.

Also write code in main function to test the working of the **`reverse()`** function.

Task # 2.0 *(Pre-Requisite)* *(Max Time: 25 Minutes)*

Implement a class for **Circular Doubly Linked List (with a dummy header node)** which stores integers **sorted in ascending order**. 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 {
  private:
      DNode head;              // Dummy header node
  public:
      CDLinkedList();          // Default constructor
      ~CDLinkedList();         // Destructor
      bool insert (int val);   // Inserts val into the linked list
      bool remove (int v);     // Remove the first occurrence of val from the linked list
      void display();          // Displays the contents of linked list on screen
      …
};
```

Task # 2.1 *(Max Time: 35 Minutes)*

Now, implement the following public member functions of the **`CDLinkedList`** 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 4th 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 # 2.2 *(Max Time: 15 Minutes)*

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

`void combine (CDLinkedList& list1, CDLinkedList& 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`**). The time complexity of **`combine`** 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.

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.*

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

Task # 2.3 *(Max Time: 15 Minutes)*

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

`void shuffleMerge (CDLinkedList& list1, CDLinkedList& 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 CDLinkedList 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: 40 Minutes)*

The Josephus problem is the following game: **N** people, numbered **1** to **N**, are sitting in a circle. Starting at person 1, a token is passed. After **M** passes, the person holding the token is eliminated, the circle closes together, and the game continues with the person who was sitting after the eliminated person picking up the token. The last remaining person wins. Thus, if **M** = 0 and **N** = 5, players are eliminated in order 1, 2, 3, 4, and player 5 wins. If **M** = 1 and **N** = 5, the order of elimination is 2, 4, 1, 5, and player 3 wins. If **M** = 2 and **N** = 5, the order of elimination is 3, 1, 5, 2, and player 4 wins.

You are required to write a program to solve the Josephus problem for general values of **M** and **N**, according to the following specifications:

```
class JosephusList;

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

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

The **`JosephusList`** class will have the following public member functions:

`JosephusList (int N)`
This is the constructor of JosephusList class. This constructor will create a **circular singly linked list** of **N** nodes containing values from **1** to **N**. For example, if at declaration time the user specifies **N** to be 5, this constructor should create a circular linked list containing these 5 values **{1 2 3 4 5}**.

`~JosephusList ()`
Destructor

`int getWinner (int M)`
This function will take M as an argument, and will return the number of the winning person determined through the process described above. *During its execution, this function will display the numbers corresponding to the persons which are being eliminated.* Make sure that the nodes corresponding to the eliminated persons are deleted from the list correctly. When this function completes its execution, there should be only one node (the winner) left in the list.

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.