Friday, June 03, 2022.

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

Lab # 8

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

Implement a **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 insertAtStart (int val)`
This function should insert a new value (`val`) at the start of the linked list. The time complexity of this function should be **constant** i.e. ***O*(1)**.

`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 # 1

(Max Time: 20 Minutes)

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

`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 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 function.

Task # 2

(Max Time: 25 Minutes)

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}**, **list2** contains **{5 9}**, and **list3** is Empty, 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 LinkedList object on which this function is called is empty at the start of this function.*

Hint 1: Do a lot of **paperwork** before writing the code. Also, make sure that all the **boundary cases / special cases** are properly handled.

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

Hint 2: Pseudo-code outline of **`combine`** function:

```
if (Both list1 and list2 are EMPTY)
{

}
else if (list1 is EMPTY)
{

}
else if (list2 is EMPTY)
{

}
else          // Both list1 and list2 contain at least one node
{

}
```

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

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

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

This function should shuffle-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}**, **`list2`** contains **{8 1 3}**, and **`list3`** is Empty, 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 shuffle-merged) contain the **same number of elements/nodes**. You can also assume that the LinkedList object on which this function is called is empty at the start of this function.*

Hint: Do a lot of **paperwork** before writing the code. Also, make sure that all the **boundary cases / special cases** are properly handled.

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

Task # 4 *(Max Time: 20 Minutes)*

Implement the following two 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 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).

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

Task # 5 *(Max Time: 20 Minutes)*

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

`int findMin ()`

This function should *iteratively* determine and return the **Smallest** integer present in the linked list. If the linked list is empty, this function should return the special value **`999`**.

`int findMax ()`

This function should *iteratively* determine and return the **Largest** integer present in the linked list. If the linked list is empty, this function should return the special value **`-999`**.

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

Task # 6 *(Max Time: 10 Minutes)*

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

`int findMin ()`

This function should **recursively** determine and return the minimum integer present in the linked list. If the linked list is empty, this function should return the special value **`999`**. This function will call a private member function of the **`LinkedList`** class to accomplish its task.

Task # 7 *(Max Time: 10 Minutes)*

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

```
int countEvens ()
```

This function should **recursively** determine (and return) the count of Even integers present in the linked list. This function will call a private member function of the **`LinkedList`** class to accomplish its task.

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.