Friday, August 26, 2022.

Data Structures and Algorithms LAB – Spring 2022

(BS-IT-F20 Morning & Afternoon)

Lab # 10

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

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**:

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

```
class CDLLD {
  private:
      DNode head; // Dummy header node
  public:
      …
};
```

Task # 1.2 *(Pre-Requisite)* *(Max Time: 0 Minutes)*

You should have the implementation of 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**.*

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

Task # 1.3 ***(Max Time: 25 Minutes)***

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

`bool merge (CDLLD& list1, CDLLD& 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 CDLLD 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 driver main function to test the working of the above function.

Task # 1.4 ***(Max Time: 20 Minutes)***

Now, you are required to implement another public member function of the **CDLLD** 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 **`splitList`** (from **Task # 1.2**) and **`merge`** (from **Task # 1.3**).

Task # 1.5 ***(Max Time: 20 Minutes)***

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

`void reverse ()`

This function should iteratively reverse the **CDLLD** object on which it has been called. Keep the following things in mind when implementing this function:

- **You are NOT allowed to create any new node in this function. You are also NOT allowed to modify the "data" of any node. So, the reversal is to be done by modifying the links (next and previous pointers) of the nodes in the linked list.**
- You are **NOT** allowed to 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.

Task # 1.6 *(Max Time: 25 Minutes)*

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

```
void insertionSort ()
```

This function should iteratively sort the **`CDLLD`** object on which it has been called using the **Insertion sort** algorithm. Keep the following things in mind when implementing this function:

- **You are NOT allowed to create any new node in this function. You are also NOT allowed to modify the "data" of any node. So, the sorting is to be done by modifying the links (next and previous pointers) of the nodes in the linked list.**
- You are **NOT** allowed to create any temporary array (or linked list) to perform sorting.

Task # 2 *(Max Time: 40 Minutes)*

The Josephus problem is as follows: **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. See the following examples:

- 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 PersonNode {
      friend class JosephusList;
  private:
      int id;
      PersonNode* next;
};
```

```
class JosephusList {
  private:
      PersonNode* first;
  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** person-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 singly 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.