Data Structures & Algorithms LAB
(BSCS-F18 Morning & Afternoon)
Lab # 8

Task # 1

Implement a class for **Circular Doubly Linked List (with a dummy header node)** which stores integers in **unsorted 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 at the start of linked list - Time complexity: O(1)
        void display(); // Display the contents of linked list on screen
…
};
```

Task # 2

Now, implement the following public member functions of the **CDLinkedList** class:

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

This function will remove the *last node* from the linked list. Before de-allocating 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 de-allocating 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 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 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 drive main function to test the working of each of the above-mentioned functions.

Task # 3

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 # 4
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. 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 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.