National University of Computer & Emerging Sciences,
Karachi
Computer Science Department
Fall 2022, Lab Manual – 05

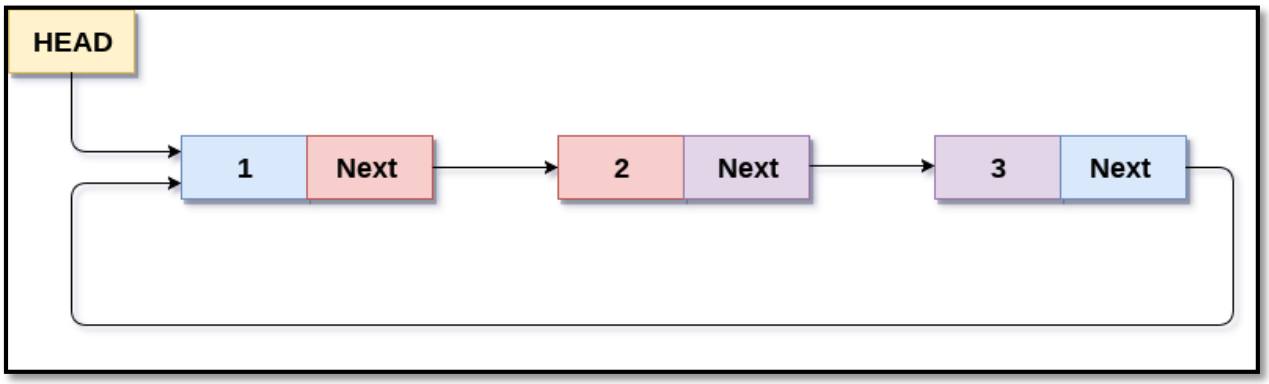
Course Code: CL-2001	Course : Data Structures - Lab
Instructor(s) :	Abeer Gauher, Sobia Iftikhar

LAB - 5

CIRCULAR AND DOUBLY LINKED LIST

Circular Linked List

The circular linked list is the collection of nodes in which tail node also point back to head node.



Simple Algorithm:

- **Define a Node class which represents a node in the list. It has two properties data and next which will point to the next node.**
- **Define another class for creating the circular linked list and it has two nodes: head and tail.**
- **To add: It first checks whether the head is null, then it will insert the node as the head.**
- **Both head and tail will point to the newly added node.**

Sample code for adding a new node in a circular linked list:
Example:

```
public class CreateList {
    //Represents the node of list.
    public class Node{
        int data;
        Node next;
        Node tail;
        public Node(int data) {
            this.data = data;
            this.next=null;
        }
    }
    //Declaring head and tail pointer as null.
    Node head = null;
    Node tail=null;
```

```java
public void add(int data){
    //Create new node
    Node newNode = new Node(data);
    //Checks if the list is empty.
    if(head == null) {
        //If list is empty, both head and tail would point to new node.
        head = newNode;
        tail = newNode;
        newNode.next = head;
    }
    else {
        //tail will point to new node.
        tail.next = newNode;
        //New node will become new tail.
        tail = newNode;
        //Since, it is circular linked list tail will point to head.
        tail.next = head;
    }
}
```

```java
//Displays all the nodes in the list
public void display() {
    Node current = head;
    if(head == null) {
        System.out.println("List is empty");
    }
    else {
        System.out.println("Nodes of the circular linked list: ");
        do{
            //Prints each node by incrementing pointer.
            System.out.print(" "+ current.data);
            current = current.next;
        }while(current != head);
        System.out.println();
    }
}
```

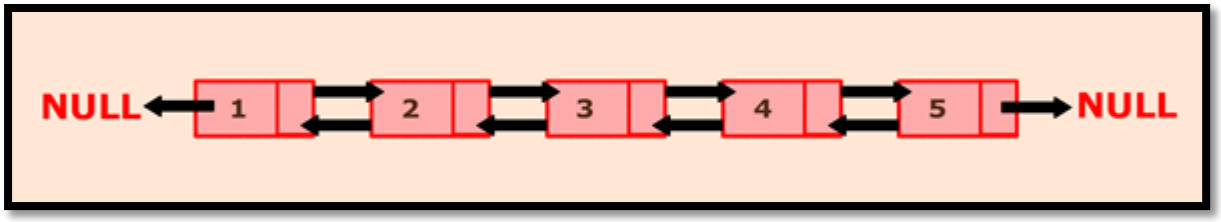
```
public static void main(String[] args) {
    CreateList cl = new CreateList();
    //Adds data to the list
    cl.add(1);
    cl.add(2);
    cl.add(3);
    cl.add(4);
    //Displays all the nodes present in the list
    cl.display();
```

```
Nodes of the circular linked list:
 1 2 3 4
```

Doubly Linked List

One of the limitations of the singly linked list is that it can be traversed in only one direction that is forward.
The doubly linked list has overcome this limitation by providing an additional pointer that points to the previous node.
With the help of the previous pointer, the doubly linked list can be traversed in a backward direction thus making insertion and deletion operation easier to perform.
So, a typical node in the doubly linked list consists of three fields:

- Data represents the data value stored in the node.
- Previous represents a pointer that points to the previous node.
- Next represents a pointer that points to next node in the list.



Above picture represents a doubly linked list in which each node has two pointers to point to previous and next node respectively. Here, node 1 represents the head of the list. The previous pointer of the head node will always point to NULL. Next pointer of node one will point to node 2. Node 5 represents the tail of the list whose previous pointer will point to node 4, and next will point to NULL.
Simple Algorithm:

- **Define a Node class which represents a node in the list. It will have three properties: data, previous which will point to the previous node and next which will point to the next node.**
- **Define another class for creating a doubly linked list, and it has two nodes: head and tail. Initially, head and tail will point to null.**
- **To add: It first checks whether the head is null, then it will insert the node as the head.**
- **Both head and tail will point to a newly added node.**
- **Head's previous pointer will point to null and tail's next pointer will point to null.**

Sample code for adding a new node in a doubly linked list:

```
public class DoublyLinkedList {
    class Node{
        int data;
        Node previous;
        Node next;

        public Node(int data) {
            this.data = data;
            this.previous = null;
            this.next = null;
        }
    }
```

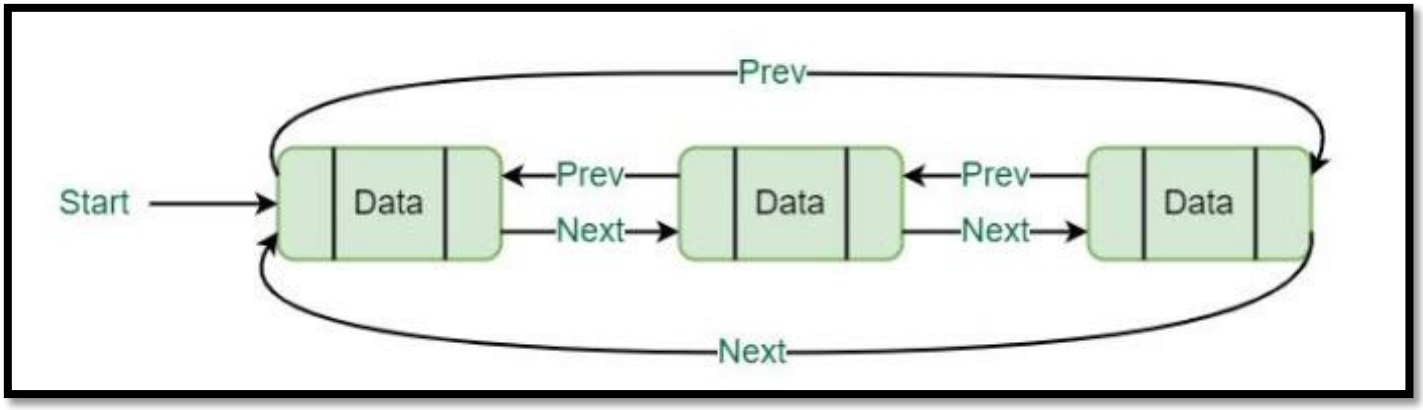
```
Node head, tail = null;
//addNode() will add a node to the list
public void addNode(int data) {
    Node newNode = new Node(data);
    //If list is empty
    if(head == null) {
        //Both head and tail will point to newNode
        head = tail = newNode;
        //head's previous will point to null
        head.previous = null;
        //tail's next will point to null, as it is the last node of the list
        tail.next = null;
    }
    else {
        //newNode will be added after tail such that tail's next will point to newNode
        tail.next = newNode;
        //newNode's previous will point to tail
        newNode.previous = tail;
        //newNode will become new tail
        tail = newNode;
        //As it is last node, tail's next will point to null
        tail.next = null;
    }
}
```

```java
public static void main(String[] args) {

    DoublyLinkedList dList = new DoublyLinkedList();
    dList.addNode( data: 1);
    dList.addNode( data: 2);
    dList.addNode( data: 3);
    dList.addNode( data: 4);
    dList.addNode( data: 5);

    //Displays the nodes present in the list
    dList.display();
```

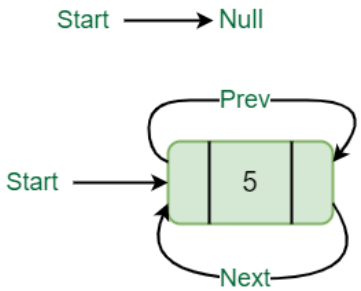
```
Nodes of doubly linked list:
1 2 3 4 5
```

Circular Doubly Linked List

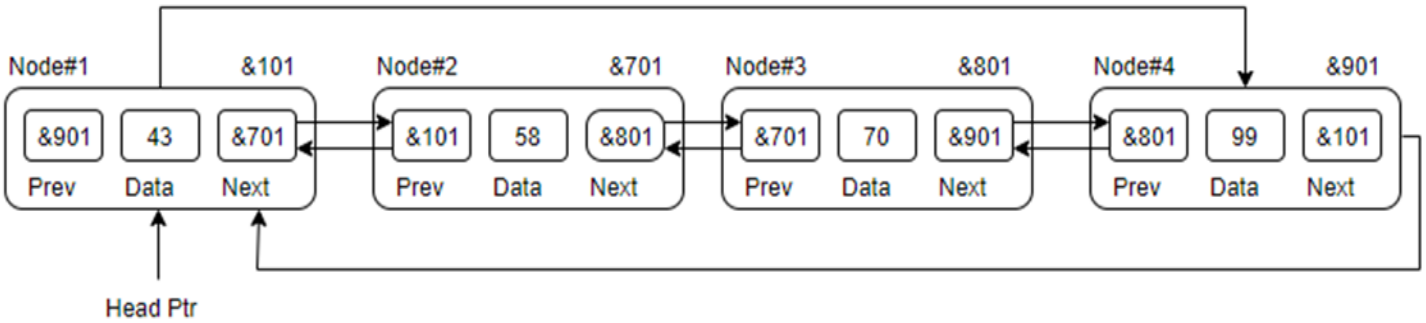
Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous and next pointer and the last node points to the first node by the next pointer and also the first node points to the last node by the previous pointer.



Insertion at the end of the list or in an empty list:
A node(Say N) is inserted with data = 5. So, the previous pointer of N points to N and the next pointer of N also points to N. But now start pointer points to the first node of the list.



```
static void insertEnd(int value)
{

    // If the list is empty, create a single node circular and doubly list
    if (start == null) {
        Node new_node = new Node();
        new_node.data = value;
        new_node.next = new_node.prev = new_node;
        start = new_node;
        return;
    }

    // If list is not empty

    // Find last node
    Node last = (start).prev;
    Node new_node = new Node();
    new_node.data = value;

    // Start is going to be next of new_node
    new_node.next = start;

    // Make new node previous of start
    (start).prev = new_node;

    // Make last previous of new node
    new_node.prev = last;

    // Make new node next of old last
    last.next = new_node;
}
```



Lab Tasks:

Task#1:
Create a menu based driven system that implements a doubly linked list:

- Insert a new node at the end of the list.
- Insert a new node at the beginning of list.
- Insert a new node at given position by the user.
- Delete a node at the end of the list.
- Delete a node at the start of the list.
- Delete a node at any point as given by the user.
- Print the complete doubly link list.

Task#2:
Given two circular linked lists L1 and L2, the task is to find if the two circular linked lists are identical or not.
Input: L1: 1 -> 2 -> 3 -> 4 -> 5 -> 1 -> 2 -> 6
L2: 5 -> 1 -> 2 -> 6 -> 1 -> 2 -> 3 -> 4
Output: Yes
Explanation: If checked the 5th element of L1 and 1st element of L2 then they are identical.
As they are circular, does not matter from where we start checking.

Input: L1: 1 -> 2 -> 3
L2: 1 ->3 -> 2
Output: No

Task#3:
Implement a program that does the following:
You need to swap two nodes x and y (not the contents) in a doubly linked list. You only need to change the links in nodes x, y and their neighbours; data elements do not change. Your program should work in the following conditions:

i. Nodes x and y are not next to each other in the list.
ii. Node x is the previous node of node y in the list.
iii. Node x is the next node after node y in the list.
iv. x and y are the same node (no need to swap).

Task#4:
You are given a circular doubly linked list that contains integers as the data in each node. The data on each node is distinct. Your algorithm prints the three continuous elements of the list, starting from the first element or head of the list and runs for infinite time. For example, if the list is {1,9,12,7} , then the output of the algorithm will be {1,9,12,9,12,7,12,7,1…} . The output contains the infinite number of elements because it is a circular list.
Your task is to determine the number of elements available in the original list and print the respective elements.