LAB-03

Doubly Linked List

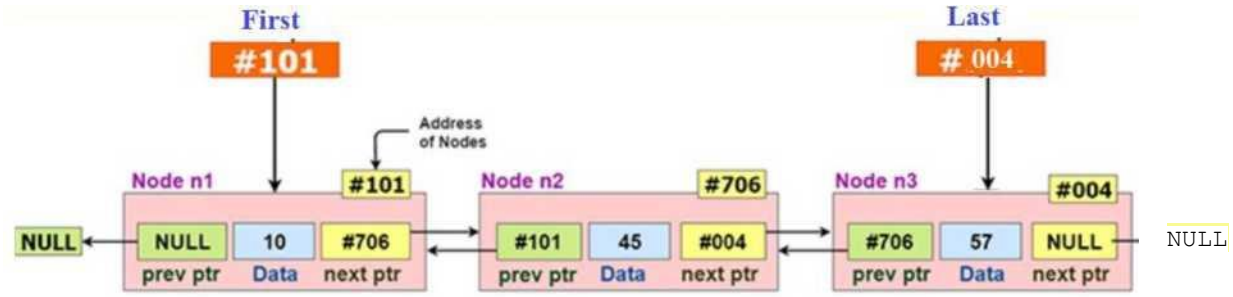
Activity Outcomes:

This lab teaches you the following topics:

- **Creation of doubly linked list**
- **Insertion in doubly linked list**
- **Deletion from doubly linked list**
- **Traversal of all nodes**

1) Useful Concepts

A *doubly-linked* list is a (dynamically allocated) list where the nodes feature two relationships: *successor* and *predecessor.* A model of such a list is given in figure . The type of a node in a doubly-linked list may be defined as follows:




```
public class Node {
      public Node(int val, Node next, Node prev) {
          // TODO Auto-generated constructor stub
           this.data = val;
           this.prev = prev;
           this.next = next;   }

     // stores data
    public int data;
    // pointer to the next node
    public Node next;
    // pointer to the previous node
    public Node prev;

    public int get() {return data;}
    public Node getNext() {return next;}
}//end class
```

Activity 1:

Create node at the end of the list

```
// Function to add a node in the back of doubly linked list
     public void addNodeBack(int val) {
          // Creating a new node
          Node current = new Node(val, null, tail);
          // Cheking if tail is null
          if(tail != null)          {
              // Assigning current to tail.next
              tail.next = current;
          }
          // Assigning current to tail
          tail = current;
          // Cheking if head is null
          if(head == null)
          { // Assigning current to head
              head = current;
          }
          System.out.println("New node added: " + val);
     }//end addNodeBack
```

Activity 2:

Display all list

```
public void display() {
           Node current = head;
           while(current!=null) {
                System.out.print(current.get()+" ");

                current = current.getNext();
           }}//end display
```

Activity 3:

Insertion in doubly linked list

1. insert node before list

```
// Function to add a node in the front of doubly linked list
     public void addNodeFront(int val) {
          // Creating a new node
          Node current = new Node(val, head, null);
          // checking if head is null
          if(head != null )
          {// Assigning current to head.prev
              head.prev = current;
          }
```

```
            // Assigning current to head
            head = current;
            // Cheking if tail is null
            if(tail == null)
            { // Assigning current to tail
                tail = current;
            }
            System.out.println("New node added: " + val);
        }//addNodeFront
```

2. insert node at back of the list

```
// Function to add a node in the back of doubly linked list
        public void addNodeBack(int val) {
            // Creating a new node
            Node current = new Node(val, null, tail);
            // Cheking if tail is null
            if(tail != null)          {
                // Assigning current to tail.next
                tail.next = current;
            }
            // Assigning current to tail
            tail = current;
            // Cheking if head is null
            if(head == null)
            { // Assigning current to head
                head = current;
            }
            System.out.println("New node added: " + val);
        }//end addNodeBack
```

Activity 4:

Deletion from doubly linked list

1. delete specific value from the list

```
public void delete(int key) {
        Node current = head;
        while(current!=null) {
             if(current.data==key)
             {     if(current.prev!=null)
                   { current.prev.next = current.next;}
             else{head = current.next;}
             if(current.next!=null) {
                   current.next.prev = current.prev;}
                   break;
             }
             current = current.next;}//end while }
```

2. clear all list removing each node one by one

```
public void clear() {
    Node current = head;
    while (current != null) {
        Node nextNode = current.next;
        current.prev = null;
        current.next = null;
        current = nextNode;
    }
    head = null;
}
```

Activity 4:

Traversal of all nodes

1. Traverse Forward

```
public void iterateForward(){
        Node current = head;
         while(current != null){
             System.out.print(current.data+" ");
             current = current.next;
         }
         System.out.println();
     }//end iterateForward
```

2. Traverse Backward

```
public void iterateBackward(){
         Node current = tail;
         while(current != null){
             System.out.print(current.data+" ");
                 current = current.prev;
             }
         System.out.println();
         }//end iterateBackward
```

2) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab

Lab Task 1

Write a function which reverses the order of a doubly linked list.

Lab Task 2

Write a function which takes two values as input from the user and searches them in the list. If both the values are found, your task is to swap both the nodes in which these values are found. Note, that you are not supposed to swap values.

Lab Task 3

Write a function that takes a singly linked list as parameter and creates a doubly linked list for the same data present in the singly linkd list i.e. for user the singly linked list will be converted into doubly linked list.