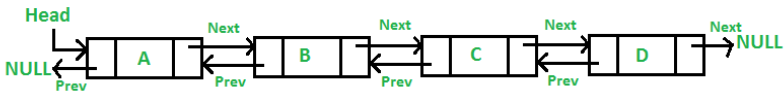


Name	Hatim Yusuf Sawai
UID no.	2021300108
Experiment No.	4

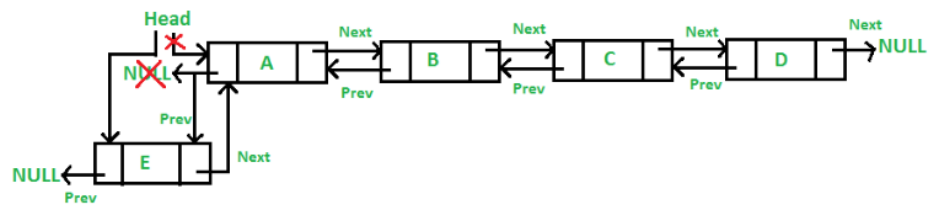
AIM:	Doubly Linked List - insert & delete operations
Program	
PROBLEM STATEMENT:	Perform insert & delete operations on doubly linked list. Take position as input from the user and make functions for left/right specific insertion/deletion.
THEORY:	<p>Doubly Linked List (DLL)</p> <p>A Doubly Linked List (DLL) contains an extra pointer, typically called the previous pointer, together with the next pointer and data which are there in the singly linked list.</p> <p>Structure:</p>  <p>Advantages:</p> <p>The advantage of a doubly linked list is that given a node in the list, we can navigate in both directions. A node in a singly linked list cannot be removed unless we have the pointer to its predecessor. But in a doubly linked list, we can delete a node even if we don't have the previous node's address (since each node has a left pointer pointing to the previous node and can move backwards).</p> <p>Disadvantages:</p> <p>The primary disadvantages of doubly linked lists are:</p> <ul style="list-style-type: none"> • Each node requires an extra pointer, requiring more space. • The insertion or deletion of a node takes a bit longer (more pointer operations).

Insertion in DLL:

A node can be added in four ways:

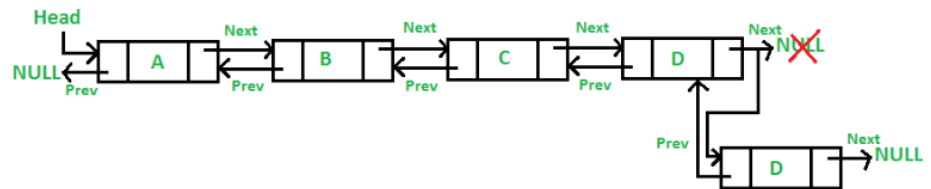
1. At the front of the DLL

Update the right pointer of the new node to point to the current head node and make the left pointer of the new node NULL. Update the head node's left pointer to point to the new node and make the new node as head.

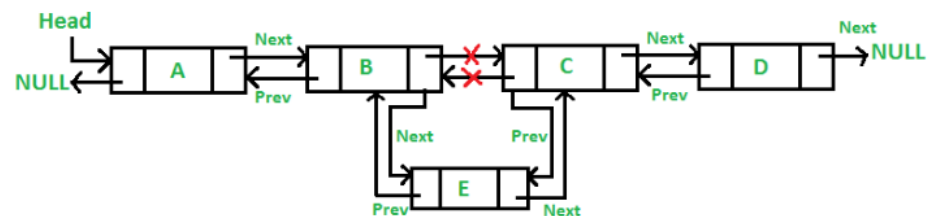


2. At the end of the DLL

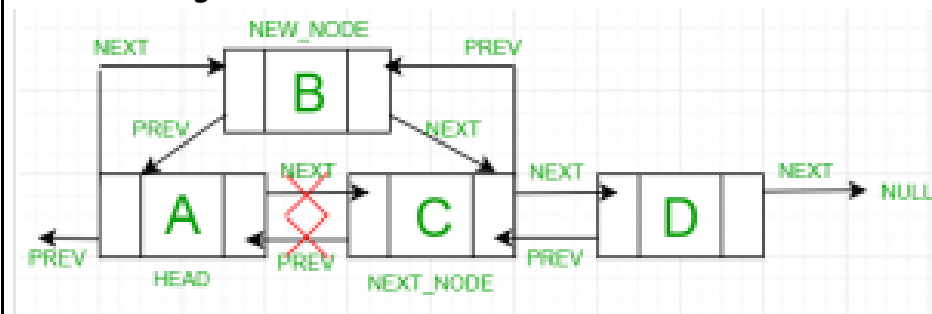
New node's right pointer points to Null and left pointer points to the end of the list. Update right pointer of last node to point to new node.



3. After a given node:



4. Before a given node:

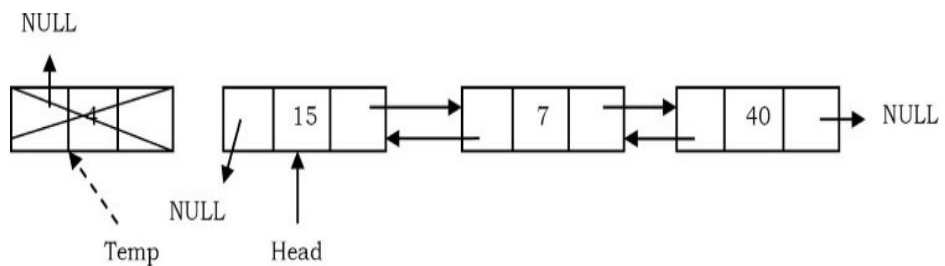


Deletion in DLL:

Similarly, deletion in DLL can be done in 3 ways:

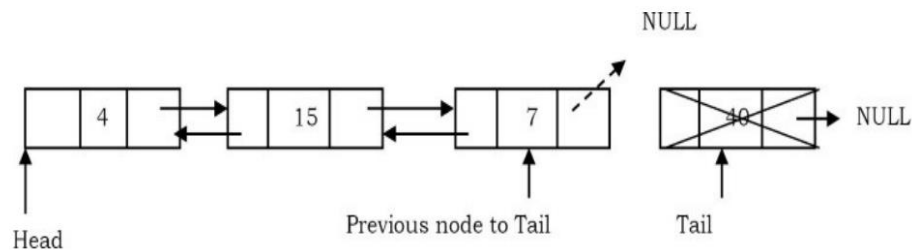
1. Deleting from front

Create a temporary node which will point to the same node as that of head. Now, move the head nodes pointer to the next node and change the heads left pointer to NULL & dispose of the temporary node.



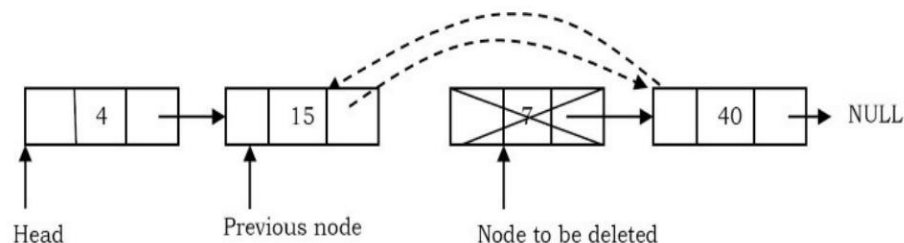
2. Deleting from end

Traverse the list and while traversing maintain the previous node address also. By the time we reach the end of the list, we will have two pointers, one pointing to the tail and the other pointing to the node before the tail. Update the next pointer of previous node to the tail node with NULL.



3. Deleting intermediate node

Like the previous case, maintain the previous node while also traversing the list. Upon locating the node to be deleted, change the previous node's next pointer to the next node of the node to be deleted.



	<p>Applications of DLL:</p> <ul style="list-style-type: none"> • Doubly linked list can be used in navigation systems where both forward and backward traversal is required. • It can be used to implement different tree data structures. • It can be used to implement undo/redo operations.
ALGORITHM:	<ol style="list-style-type: none"> 1. Create DLL Class with inner Node class 2. Create 2 ref vars: next, prev & int var data 3. Initialize head, tail to null 4. Create methods for diff operations of DLL <p>insertAtFront Method:</p> <ol style="list-style-type: none"> 1. Create newnode(data) 2. Check if head==null: 3. Set head & tail as newnode 4. Else: 5. prev of head = newnode 6. next of newnode = head 7. head = newnode <p>insertAtEnd Method:</p> <ol style="list-style-type: none"> 1. Create newnode(data) 2. Check if head==null: 3. Set head & tail as newnode 4. Else: 5. Set next of tail = newnode 6. Prev of newnode = tail 7. Set tail = newnode <p>insertAtPos Method:</p> <ol style="list-style-type: none"> 1. Check if pos is <1 or pos>length+1: 2. Print Invalid location 3. Check if pos==1: 4. Call insertAtFront 5. Check if pos==len+1: 6. Call insertAtEnd

7. Else:
8. Create newnode(data) & temp pointer = head
9. Traverse linked list till temp is behind required pos
10. Set next of newnode = next of temp
11. Set prev of next of temp = newnode
12. Set next of temp as newnode
13. Set prev of newnode = temp

deleteAtFront Method:

1. Check if head == null:
2. Print list is empty
3. Else:
4. Initialize data to head.data
5. Set head = next of head
6. Set prev of head as null
7. return data

deleteAtEnd Method:

1. Check if head == null:
2. Print list is empty
3. Else:
4. Initialize data to tail.data
5. Set tail = prev of tail
6. Set next of tail as null
7. return data

deleteAtPos Method:

1. Check if pos is <1 or pos>length+1:
2. Print Invalid location
3. Check if pos==1:
4. Call deleteAtFront
5. Check if pos==len+1:
6. Call deleteAtEnd
7. Else:
8. Create temp pointer = head
9. Traverse linked list till temp is behind required pos

10. Store data of next of temp
11. Set next of temp = next of next of temp
12. Set prev of next of temp = temp
13. return data

PROBLEM SOLVING:

HATIM SAWAI

★ DLL Operations:

Input:

- i) insert at front: 50
- ii) insert at end: 51
- iii) insert at right: pos: $\frac{1}{2}$, ele: 52 } pos
- iv) insert at left: pos: 2, ele: 72 } pos
- v) delete at front:
- vi) delete at right: pos: 2 } pos
- vii) delete at left: pos: 2 } pos
- viii) delete at end:

i) before: head \rightarrow null
after: head \rightarrow [null | 50] \rightarrow null

ii) before: head \rightarrow [null | 50] \rightarrow null
after: head \rightarrow [null | 50] $\xrightarrow{1}$ [51] \rightarrow null

iii) after: head \rightarrow [null | 50] $\xrightarrow{1}$ [51] \rightarrow null
 [50] $\xrightarrow{2}$ [52] $\xrightarrow{3}$ [51] \rightarrow null

iv) after: head \rightarrow [null | 50] $\xrightarrow{1}$ [51] \rightarrow null
 [50] $\xrightarrow{2}$ [72] $\xrightarrow{3}$ [52] $\xrightarrow{4}$ [51] \rightarrow null

v) after: head \rightarrow [null | 72] $\xrightarrow{1}$ [52] $\xrightarrow{2}$ [51] \rightarrow null
 [72] $\xrightarrow{\text{left}}$ [52] $\xrightarrow{\text{right}}$ [51]

vii/v) after: head \rightarrow [null | 52] \rightarrow null

viii) after: head \rightarrow null

PROGRAM:**DLLCheck.java:**

```
import java.util.Scanner;
import dlinkedlistds.DLL;
public class DLLCheck {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        DLL d = new DLL();
        int choice,flag,pos,data;
        while(true) {
            System.out.println(
                "Select an option:\n1.Insert at front\n2.Insert at
Right\n3.Insert at left\n4.Insert at end\n5.Delete at front\n6.Delete
at right\n7.Delete at left\n8.Delete at end");
            choice = sc.nextInt();
            switch(choice) {
                case 1:
                    System.out.print("Enter the element to be inserted: ");
                    d.insertAtFront(sc.nextInt());
                    System.out.println("The list is: " + d.printList());
                    break;
                case 2:
                    System.out.print("Enter the element to be inserted: ");
                    data = sc.nextInt();
                    System.out.print("Enter the pos after which to insert: ");
                    pos = sc.nextInt();
                    if (pos == -1) {
                        System.out.println("Element not found!");
                    } else {
                        d.insertAtPos(data, pos+1);
                        System.out.println("The list is: " + d.printList());
                    }
                    break;
                case 3:
                    System.out.print("Enter the element to be inserted: ");
                    data = sc.nextInt();
                    System.out.print("Enter the pos before which to insert:
```

```

");

    pos = sc.nextInt();
    if (pos == -1) {
        System.out.println("Element not found!");
    } else {
        d.insertAtPos(data,pos);
        System.out.println("The list is: " + d.printList());
    }
    break;
case 4:
    System.out.print("Enter the element to be inserted: ");
    d.insertAtEnd(sc.nextInt());
    System.out.println("The list is: " + d.printList());
    break;
case 5:
    System.out.println("Deleted element:
"+d.deleteAtFront());
    System.out.println("The list is: " + d.printList());
    break;
case 6:
    System.out.print("Enter pos after which to delete: ");
    pos = sc.nextInt();
    if(pos == -1) {
        System.out.println("Element not found!");
    } else {
        System.out.println("Deleted element:
"+d.deleteAtPos(pos+1));
        System.out.println("The list is: " + d.printList());
    }
    break;
case 7:
    System.out.print("Enter pos before which to delete: ");
    pos = sc.nextInt();
    if (pos == -1) {
        System.out.println("Element not found!");
    } else {

```



```

        System.out.println("Deleted element: " +
d.deleteAtPos(pos-1));
        System.out.println("The list is: " + d.printList());
    }
    break;
case 8:
    System.out.println("Deleted element: "+d.deleteAtEnd());
    System.out.println("The list is: " + d.printList());
    break;
case 9:
    System.out.println("The list is: "+d.printList());
    break;
default:
    System.out.println("Invalid choice!");
}
System.out.println("Do you want to continue?\n1. Yes\t2. No");
flag = sc.nextInt();
if (flag == 2) {
    break;
}
}
sc.close();
}
}

```

DLL.java:

```

package dlinkedlistds;
public class DLL {
    class Node {
        int data;
        Node next,prev;
        public Node(int data) {
            this.data = data;
            next = null;
            prev = null;
        }
    }
}

```

```

Node head=null;
Node tail=null;
int len;
public void insertAtFront(int data) {
    Node newNode = new Node(data);
    if(head==null) {
        head = newNode;
        tail = newNode;
    } else {
        head.prev = newNode;
        newNode.next = head;
        head = newNode;
    }
    len++;
}
public void insertAtEnd(int data) {
    Node newNode = new Node(data);
    if(head==null) {
        head = newNode;
        tail = newNode;
    } else {
        tail.next = newNode;
        newNode.prev = tail;
        tail = newNode;
    }
    len++;
}
public void insertAtPos(int data,int pos) {
    if(pos<1||pos>len+1) {
        System.out.println("Invalid position");
    } else if(pos==1) {
        insertAtFront(data);
    } else if(pos==len+1) {
        insertAtEnd(data);
    } else {
        Node newNode = new Node(data);

```

```

        Node temp = head;
        for(int i=1;i<pos-1;i++) {
            temp = temp.next;
        }
        newNode.next = temp.next;
        temp.next.prev = newNode;
        temp.next = newNode;
        newNode.prev = temp;
        len++;
    }
}

public int deleteAtFront() {
    if(head==null) {
        System.out.println("List is empty");
        return -1;
    } else {
        int data = head.data;
        head = head.next;
        head.prev = null;
        len--;
        return data;
    }
}

public int deleteAtEnd() {
    if(head==null) {
        System.out.println("List is empty");
        return -1;
    } else if(len==1) {
        int data = head.data;
        head = null;
        tail = null;
        len--;
        return data;
    } else {
        int data = tail.data;
        tail = tail.prev;
    }
}

```

```

        tail.next = null;
        len--;
        return data;
    }
}

public int deleteAtPos(int pos) {
    if(pos<1||pos>len) {
        System.out.println("Invalid position");
        return -1;
    } else if(pos==1) {
        return deleteAtFront();
    } else if(pos==len) {
        return deleteAtEnd();
    } else {
        Node temp = head;
        for(int i=1;i<pos-1;i++) {
            temp = temp.next;
        }
        int data = temp.next.data;
        temp.next = temp.next.next;
        temp.next.prev = temp;
        len--;
        return data;
    }
}

public String printList() {
    Node temp = head;
    String s = "head->";
    while(temp!=null) {
        s += temp.data+(temp.next!=null?"->":"" );
        temp = temp.next;
    }
    s += "<-tail";
    return s;
}
}

```

OUTPUT:

1. Insert at Right

```
The list is: head->48->49->50->51->52->53<-tail
Do you want to continue?
1. Yes  2. No
1
Select an option:
1.Insert at front
2.Insert at Right
3.Insert at left
4.Insert at end
5.Delete at front
6.Delete at right
7.Delete at left
8.Delete at end
2
Enter the element to be inserted: 71
Enter the pos after which to insert: 2
The list is: head->48->49->71->50->51->52->53<-tail
Do you want to continue?
1. Yes  2. No
█
```

2. Insert at Left

```
The list is: head->48->49->71->50->51->52->53<-tail
Do you want to continue?
1. Yes  2. No
1
Select an option:
1.Insert at front
2.Insert at Right
3.Insert at left
4.Insert at end
5.Delete at front
6.Delete at right
7.Delete at left
8.Delete at end
3
Enter the element to be inserted: 72
Enter the pos before which to insert: 4
The list is: head->48->49->71->72->50->51->52->53<-tail
Do you want to continue?
1. Yes  2. No
█
```

3. Delete at Right

```
The list is: head->49->71->72->50->51->52<-tail
Do you want to continue?
1. Yes  2. No
1
Select an option:
1.Insert at front
2.Insert at Right
3.Insert at left
4.Insert at end
5.Delete at front
6.Delete at right
7.Delete at left
8.Delete at end
6
Enter pos after which to delete: 2
Deleted element: 72
The list is: head->49->71->50->51->52<-tail
Do you want to continue?
1. Yes  2. No
```

4. Delete at Left

```
The list is: head->49->71->50->51->52<-tail
Do you want to continue?
1. Yes  2. No
1
Select an option:
1.Insert at front
2.Insert at Right
3.Insert at left
4.Insert at end
5.Delete at front
6.Delete at right
7.Delete at left
8.Delete at end
7
Enter pos before which to delete: 3
Deleted element: 71
The list is: head->49->50->51->52<-tail
Do you want to continue?
1. Yes  2. No
2
PS D:\Data Structures\Exp4>
```

CONCLUSION:

In this experiment, we learned how to implement doubly linked list in java and perform insertion & deletion operation on it using position rather an data matching.