EXPERIMENT NO: 12

Doubly Linked List

Aim

write a java program for the following
1) Create a doubly linked list of elements
2) Delete a given element from the above list
3) Display contents after deletion.

Input

Integers for insertion to the list.

Output Expected

Display the status of doubly linked list after each operation.

Algorithm.

Algorithm Insert-front

```
1. ptr = head. next
2 new = GetNode (Node)
3 If (new ≠ NULL) then.
    3.1 new. prev = head
4 head. next = new
5 new. next = ptr
6 ptr. prev = new
7 new. data = x
```

```
8. Else
        8.1 Print "Memory allocation not possible"
9. EndIf
10. Stop
```

Algorithm Insert_End

```
1. ptr = head
2 while (ptr. next ≠ NULL) do
        2.1 ptr = ptr. next
3 EndWhile
4 new = GetNode (Node)
5 If (new ≠ NULL) then.
    5.1 new. prev = ptr
    5.2 ptr. next = new
    5.3 new. next = NULL
    5.4 new. data = x
6 Else
    6.1 Print "Memory Allocation not possible
7 EndIf
8 Stop
```

Algorithm Insert_Position.

```
1. ptr = head
2. while (ptr. data ≠ KEY) and (ptr. next ≠ NULL)
        2.1 ptr = ptr. next
```

```
3. Endwhile
4. new = GetNode(Node)
5. If (new = NULL) then
    5.1 Print "Memory not allocated"
    5.2 Exit
6. EndIf
7. If(ptr.~~next~~ = NULL) then.
    7.1 new.prev = ptr
    7.2 ptr.next = new
    7.3 new.next = NULL.
    7.4 new.data = X
8 Else
    8.1 ptr1 = ptr.next
    8.2 new.prev = ptr
    8.3 new.next = ptr1
    8.4 ptr.next = new
    8.5 ptr1.prev = new
    8.6 ptr = new
    8.7 new.data = X
9 EndIf
10 Stop
```

Algorithm DeleteFront

```
1. ptr = head.next
2. If(ptr = NULL) then.
    2.1 Print "List is empty : No deletion possible"
    2.2 Exit
```

```
3 Else
    3.1 ptr1 = ptr.next
    3.2 head.next = ptr1
    3.3 If (ptr1 ≠ NULL)
            3.3.1 ptr1.prev = head
    3.4 EndIf
    3.5 ReturNode(ptr)
4 EndIf
5 Stop
```

Algorithm DeleteEnd

```
1. ptr = head
2  while (ptr.next ≠ NULL) do
        2.1 ptr = ptr.next
3  EndWhile
4  If (ptr = head) then.
    4.1 Print "List is empty"
    4.2 Exit
5  Else
     5.1 ptr1 = ptr.prev
     5.2 ptr1.next = NULL
     5.3 ReturnNode(ptr)
6  EndIf
7  Stop
```

7/3/22

Algorithm Delete Position.

```
1 ptr . head.next
2 If (ptr = NULL) then.
    2.1 Print "List is empty"
    2.2 Exit
3 EndIf
4 while (ptr.data ≠ KEY) and (ptr.next ≠ NULL) do
    4.1 ptr = ptr.next
5 Endwhile
6 If (ptr.data = Key) then
    6.1 ptr1 = ptr.prev
    6.2 ptr2 = ptr.next
    6.3 If (ptr2 ≠ NULL) then.
        6.3.1 ptr2.prev = ptr1
    6.4 EndIf
    6.5 Return Node(ptr)
7 Else
    7.1 Print "Node not available"
8 EndIf
9 Stop
```

Result

Output is obtained.

Output

```
Doubly Linked List Operations.
 1. Insertion at front
 2. Insertion at end
 3. Insertion at any position.
 4. Deletion  at front
 5. Deletion. at end
 6. Deletion  at any position.
           Enter your choice : 1
Enter element : 7

Doubly linked list : 7

Do you want to continue
 y
Doubly Linked List operations.

 1. Insertion at front
2. Insertion. at end
 3. Insertion at any position.
 4. Deletion. at front
  5. Deletion at end
 6. Deletion at any position.
           Enter your choice : 2
Enter element : 8
Doubly Linked List : 7⟷8
```

```
Do you want to continue?
y
Doubly Linked List Operations:
1. Insertion at front
2. Insertion at end
3. Insertion at any position
4. Deletion at front
5 Deletion at end
6 Deletion at any position.

                    Enter your Choice : 4
 Doubly Linked List = 8

Do you want to continue?
n
```

```
import java.util.Scanner;

class Node
{
    protected int data;
    protected Node next, prev;

    public Node()
    {
        next = null;
        prev = null;


        data = 0;
    }

    public Node(int d, Node n, Node p)
    {
        data = d;
        next = n;
        prev = p;
    }

    public void setLinkNext(Node n)
    {
        next = n;
    }

    public void setLinkPrev(Node p)
    {
        prev = p;
    }

    public Node getLinkNext()
    {
        return next;
    }

    public Node getLinkPrev()
    {
        return prev;
    }

    public void setData(int d)
    {
        data = d;
    }

    public int getData()
    {
        return data;
    }
}


class linkedList
{
    protected Node start;
    protected Node end ;
    public int size;


    public linkedList()
    {
        start = null;
        end = null;
        size = 0;
    }

    public boolean isEmpty()
    {
        return start == null;
    }
    public int getSize()
    {
        return size;
    }

    public void insertAtStart(int val)
    {
        Node nptr = new Node(val, null, null);
        if(start == null)
        {
            start = nptr;


            end = start;
        }
        else
        {
            start.setLinkPrev(nptr);
            nptr.setLinkNext(start);
            start = nptr;
        }
        size++;
    }

    public void insertAtEnd(int val)
    {
        Node nptr = new Node(val, null, null);
        if(start == null)
        {
            start = nptr;
            end = start;
        }
        else
        {
            nptr.setLinkPrev(end);
            end.setLinkNext(nptr);
            end = nptr;
        }
        size++;
    }

    public void insertAtPos(int val , int pos)
    {
        Node nptr = new Node(val, null, null);
        if (pos == 1)
        {
            insertAtStart(val);
            return;
        }
        Node ptr = start;
        for (int i = 2; i <= size; i++)
        {
            if (i == pos)
            {
                Node tmp = ptr.getLinkNext();
                ptr.setLinkNext(nptr);
                nptr.setLinkPrev(ptr);
                nptr.setLinkNext(tmp);
                tmp.setLinkPrev(nptr);
            }
            ptr = ptr.getLinkNext();
        }
        size++ ;
    }
public void deleteAtFront()
{
    if (size == 1)
        {
            start = null;
            end = null;
            size = 0;
            return;
```

Scanned by TapScanner

```
            start = start.getLinkNext();
            start.setLinkPrev(null);
            size--;
            return ;
  }

  public void deleteAtEnd()
  {
        end = end.getLinkPrev();
            end.setLinkNext(null);
            size-- ;


  }


 public void deleteAtPos(int pos)
 {
        Node ptr = start.getLinkNext();
        for (int i = 2; i <= size; i++)
        {
            if (i == pos)
            {
                Node p = ptr.getLinkPrev();
                Node n = ptr.getLinkNext();

                p.setLinkNext(n);
                n.setLinkPrev(p);
                size-- ;
                return;
            }
            ptr = ptr.getLinkNext();
        }
    }

    public void display()
    {
        System.out.print("\nDoubly Linked List = ");
        if (size == 0)
        {
            System.out.print("empty\n");
            return;
        }
        if (start.getLinkNext() == null)
        {
            System.out.println(start.getData() );
            return;
        }
        Node ptr = start;
        System.out.print(start.getData()+ " <-> ");
        ptr = start.getLinkNext();
        while (ptr.getLinkNext() != null)
        {
            System.out.print(ptr.getData()+ " <-> ");
            ptr = ptr.getLinkNext();
        }
        System.out.print(ptr.getData()+ "\n");
    }
}

public class DoublyLinkedList
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        linkedList list = new linkedList();
        System.out.println("Doubly Linked List Test\n");
        char ch;
        do
        {
            System.out.println("\n---X Doubly Linked List Operations
X---\n");
            System.out.println("1.
System.out.println("3. Insert at position");
System.out.println("4. Delete at front");
System.out.println("5. Delete at end");
System.out.println("6. Delete at position");
  System.out.print("\n\tEnter your choice : ");
int choice = scan.nextInt();
switch (choice)
{
case 1 :
    System.out.print("\nEnter integer element to insert :



    list.insertAtStart( scan.nextInt() );
    break;
case 2 :
    System.out.print("\nEnter integer element to insert :

    list.insertAtEnd( scan.nextInt() );
    break;
case 3 :
    System.out.print("\nEnter integer element to insert :

    int num = scan.nextInt() ;
    System.out.print("\nEnter position : ");
    int pos = scan.nextInt() ;
    if (pos < 1 || pos > list.getSize() )
        System.out.println("Invalid position\n");
    else
        list.insertAtPos(num, pos);
    break;
  case 4 :
              list.deleteAtFront();
             break;
 case 5 :
              list.deleteAtEnd();
             break;
case 6 :
    System.out.print("\nEnter position : ");
    int p = scan.nextInt() ;
    if (p < 1 || p > list.getSize() )
        System.out.println("Invalid position\n");
    else
        list.deleteAtPos(p);
    break;

default :
    System.out.println("Wrong Entry \n ");
    break;
}
/*  Display List  */
list.display();
System.out.println("\nDo you want to continue (Type y or n)
");

ch = scan.next().charAt(0);

} while (ch == 'Y'|| ch == 'y');
}
```