TRIBHUWAN UNIVERSITY
JANAPRIYA MULTIPLE CAMPUS
Simalchaur,Pokhara-8,Nepal

LAB REPORT

Data Structure and Algorithm with JAVA

Submitted To:

Mr.Prithvi Raj Paneru

Data Structure and Algorithm with JAVA

Submitted by:
Ms. Krishna Luharuka
4th Semester
Bachelor in Information Management
Roll No: 11

Mangsir,2079

INDEX:

SN	Topic	Date of Experiment	Date of submission	Signature	Remark
1.	Operation in a singly linked list.	11-11-2022	24 -11-2022		
2.	Operation in a doubly linked list.	24-11-2022			
3	Operation in a circular singly linked list.	01-12-2022			
4	Operation in a circular doubly linked list.				

EXPERIMENT: 1

TITLE:

A java program to run various operations of singly linked list.

1.OBJECTIVE:

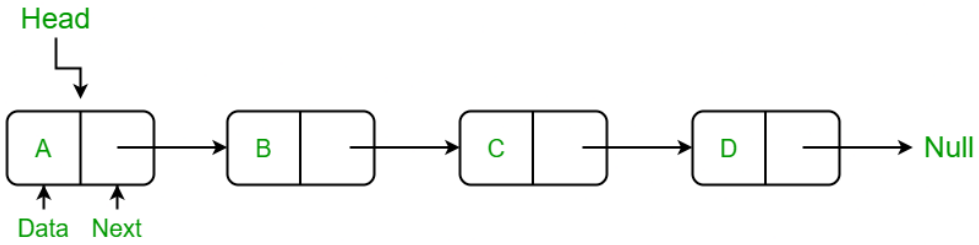
✓To insert a node at beginning, at last and at certain position and display it.

✓To delete a node from the beginning, from last and from certain position.

✓To search a node from singly linked list.

2.THEORY:

Singly linked list is the data structure used for storing collection of nodes and the node has data and reference to next node in a list. The first node is the head node and the last node has data and points to null.



The various operations of singly linked list are discussed below:

2.1. Traverse:

The traverse operation helps to display the content of a singly linked list. Here, to run this operation, the temp node is kept moving to the next one and the content is displayed.

2.2.Insertion:

The insertion operation inserts a node in linked list. The insertion operation in singly linked list can be run in three different methods. They are:

2.2.1.insertatbeg(int x):

In this method, a node is inserted at the beginning of the singly linked list.

2.2.2.insertatlast(int x):

In this method, a node is inserted at the last of the linked list.

2.2.3.insertatpos(int x, int pos):

This method inserts a node at the specified position of the linked list.

2.3. Deletion:

The deletion operation deletes a node from the linked list. The deletion operation in singly linked list can also be run in three different methods. They are:

2.3.1.deleteatbeg():

In this method, a node is deleted from the very beginning of the singly linked list.

2.3.2.deleteatlast():

In this method, a node is deleted from the last of the singly linked list.

2.3.3.deleteatpos(int pos):

In this method, a node is deleted from the specified position of the linked list.

2.4. Search:

The search(int key) operation helps to search a key from the nodes of the linked list and return true if the key exists in the node and return false if the key doesn’t exist in the singly list.

3.IMPLEMENTATION:

The following programs shows the insertion of nodes at the beginning, at last and at certain position in singly linked list data structure in java. Similarly, the deletion of node from the beginning, from the last and from the specified position is also shown. Moreover, the source code and output for search operation is also shown.

3.1.PROGRAM 1: Traverse a singly linked list.

PSEUDOCODE:

```
public void display()
   {
      node temp=head;
      while(temp!=null)
      {
         System.out.print(temp.data+"-->");
         temp=temp.next;
      }
      System.out.print(temp+"\n");
    }
```

SOURCE CODE:

```
class SllForTraverse{
   private static class node {
      int data;
      node next;
      node(int data) {
         this.data=data;
         this.next=null;
      }
   }
   public node head=null;

   public void display(){
      node temp=head;
      while(temp!=null){
         System.out.print(temp.data+"-->");
         temp=temp.next;
      }
      System.out.print(temp+"\n");

System.out.println("**********
**********");
      System.out.print("\n");
    }

   public static void main(String args[])
   {
      SllForTraverse s=new
SllForTraverse();
      s.head= new node(11);
      node second=new node(12);
      node third=new node(13);
      node fourth=new node(14);
      s.head.next=second;
      second.next=third;
      third.next=fourth;
      System.out.println("A singly linked
list traversed is given as:");
      s.display();
   }}
```

OUTPUT:

```
A singly linked list traversed is given as:
11-->12-->13-->14-->null
**********
```

3.2.PROGRAM 2: Insert Element to a Linked List either at the beginning, middle or end of the linked list.

3.2.1. Insert at the beginning:

PSEUDOCODE:

```
public void insertatbeg(int x)
{
   node newnode=new node(x);
   if(head==null)
   {
      head=newnode;
   }
   else
   {
      newnode.next=head;
      head=newnode;
   }
}
```

3.2.2. Insert at the last:

PSEUDOCODE:

```
public void insertatlast(int x)
{
   node newnode=new node(x);
   if(head==null)
   {
      head=newnode;
   }
   else
   {
      node temp=head;
      while(temp.next!=null)
      {
         temp=temp.next;
      }
      temp.next=newnode;
   }
}
```

3.2.3. Insert at specified position:

PSEUDOCODE:

```
public void insertatpos(int x,int pos)
   {
      node newnode=new node(x);
      if(pos<1)
      {
         System.out.println("position less
than 1 is invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            head=newnode;
         }
         else
         {
            newnode.next=head;
            head=newnode;
         }
      }
      else
      {
         node temp=head;
         for(int i=1;i<pos-
1&&temp!=null;i++)
         {
            temp=temp.next;
         }
         if(temp!=null)
         {
         newnode.next=temp.next;
         temp.next=newnode;
         }
         else
         {
            System.out.println("invalid
position for insert");
         }
      }
   }
```

SOURCE CODE OF INSERTION OPERATION:

```
import java.util.Scanner;
class SLL
{
   class node
   {
      int data;
      node next;
      node(int data)
      {
         this.data=data;
         this.next=null;
      }
   }
   public node head=null;

   public void insertatbeg(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         newnode.next=head;
         head=newnode;
      }
   }

   public void insertatlast(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         node temp=head;
         while(temp.next!=null)
         {
            temp=temp.next;
         }
         temp.next=newnode;
      }
   }

   public void insertatpos(int x,int pos)
   {
      node newnode=new node(x);
      if(pos<1)
      {
         System.out.println("position less
than 1 is invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            head=newnode;
         }
         else
         {
            newnode.next=head;
            head=newnode;
         }
      }
      else
      {
         node temp=head;
         for(int i=1;i<pos-
1&&temp!=null;i++)
         {
            temp=temp.next;
         }
         if(temp!=null)
         {
         newnode.next=temp.next;
         temp.next=newnode;
         }
         else
         {
            System.out.println("invalid
position for insert");
         }
      }
   }

   public void display()
   {
      node temp=head;
      while(temp!=null)
      {
         System.out.print(temp.data+"-->");
         temp=temp.next;
      }
```

```
        System.out.print(temp+"\n");

System.out.println("**********
**********");
        System.out.print("\n");

    }

    public static void main(String args[])
    {
        SLL s=new SLL();
        System.out.println("press 1 for
insertion at beginning");
        System.out.println("press 2 for
insertion at last");
        System.out.println("press 3 for
insertion at position");
        System.out.println("enter your
choice");
        Scanner input=new
Scanner(System.in);
        int ch=input.nextInt();
        switch(ch)
        {
            case 1:
            {
                s.insertatbeg(9);
                s.insertatbeg(10);
                s.display();
                break;

            }
            case 2:
            {
                s.insertatlast(11);
                s.insertatlast(15);
                s.display();
                break;
            }
            case 3:
            {
                s.insertatpos(1, -1);
                s.insertatpos(1,1);
                s.insertatpos(3,2);
                s.insertatpos(6,2);
                s.insertatpos(10,10);
                s.display();
                break;
            }
            default:
            {
                System.out.println("please enter
your choice properly");
            }
        }
    }
}
```

OUTPUT:

```
press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
1
10-->9-->null
*****************************


press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
2
11-->15-->null
*****************************


press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
3
position less than 1 is invalid
invalid position for insert
1-->6-->3-->null
*****************************


press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
4
please enter your choice properly
```

3.3.PROGRAM 3: Delete Element from a Linked List either from the beginning, middle or end of the linked list.

3.3.1. Deletion from the beginning:

PSEUDOCODE:

```
public void deleteatbeg()
  {
    if(head==null)
    {
      System.out.println("Empty list");
    }
    else if(head.next==null)
    {
      head=null;
    }
    else
    {
      head=head.next;
    }
  }
```

3.3.2. Deletion from the last:

PSEUDOCODE:

```
public void deleteatlast()

  {
    if(head==null)
    {
      System.out.print("empty list");
    }
    else if(head.next==null)
    {
      head=null;
    }
    else
    {
      node temp=head;
      while(temp.next.next!=null)
      {
        temp=temp.next;
      }
      temp.next=null;
    }
  }
```

3.3.3. Deletion from the specified position:

PSEUDOCODE:

```
public void deleteatpos(int pos){
 if(pos<1){
  System.out.println("pos invalid");
 }
 else if(pos==1){
  if(head==null)
  {
    System.out.print("empty list");
  }
  else if(head.next==null) {
    head=null;
  }
  else {
    head=head.next;
  }
 }
 else
 {
   node temp=head;
   for(int i=1;i<pos-
1&&temp.next!=null;i++)
   {
     temp=temp.next;
   }
   if(temp.next!=null)
   {
     temp.next=temp.next.next;
   }else{
     System.out.println("invalid pos");
   }
 }
}
```

SOURCE CODE OF DELETE OPERATION:

```
import java.util.Scanner;
class SLLfordelete
{
   private static class node
   {
      int data;
      node next;
      node(int data)
      {
         this.data=data;
         this.next=null;
      }
   }
   public node head=null;


   public void deleteatbeg()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         head=head.next;
      }
   }

   public void deleteatlast()
   {
      if(head==null)
      {
         System.out.print("empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         node temp=head;
         while(temp.next.next!=null)
         {
            temp=temp.next;
         }
         temp.next=null;
      }
   }

   public void deleteatpos(int pos)
   {
    if(pos<1)
    {
     System.out.println("pos invalid");
    }
    else if(pos==1)
    {
     if(head==null)
     {
        System.out.print("empty list");
     }
     else if(head.next==null)
     {
        head=null;
     }
     else
     {
        head=head.next;
     }
    }
    else
    {
      node temp=head;
      for(int i=1;i<pos-
1&&temp.next!=null;i++)
      {
        temp=temp.next;
      }
      if(temp.next!=null)
      {
        temp.next=temp.next.next;
      }
      else{
        System.out.println("invalid pos");
      }
    }
```

```
    }

    public void display()
    {
        node temp=head;
        while(temp!=null)
        {
            System.out.print(temp.data+"-->");
            temp=temp.next;
        }
        System.out.print(temp+"\n");

System.out.println("**********
**********");
        System.out.print("\n");
    }

    public static void main(String args[])
    {
        SLLfordelete s=new SLLfordelete();
        s.head= new node(11);
        node second=new node(12);
        node third=new node(13);
        node fourth=new node(14);
        s.head.next=second;
        second.next=third;
        third.next=fourth;
        System.out.println("Given a singly
linked list:");
        s.display();
        System.out.println("press 1 for
deletion from the beginning");
        System.out.println("press 2 for
deletion from the last");
        System.out.println("press 3 for delete
from the certain position");
        System.out.print("enter your choice
\t");
        Scanner input=new
Scanner(System.in);
        int ch=input.nextInt();
        switch(ch)
        {
            case 1:
            {
                s.deleteatbeg();
                s.display();
                break;
            }
            case 2:
            {
                s.deleteatlast();
                s.display();
                break;
            }
            case 3:
            {
                s.deleteatpos(-1);
                s.deleteatpos(1);
                s.deleteatpos(2);
                s.deleteatpos(10);
                s.display();
                break;
            }
            default:
            {
                System.out.println("please enter
your choice properly");
            }
        }
    }
}
```

OUTPUT:

```
Given a singly linked list:
11-->12-->13-->14-->null
press 1 for deletion from the beginning
press 2 for deletion from the last
press 3 for delete from the certain position
enter your choice     1
12-->13-->14-->null
*****************************

Given a singly linked list:
11-->12-->13-->14-->null
press 1 for deletion from the beginning
press 2 for deletion from the last
press 3 for delete from the certain position
enter your choice     2
11-->12-->13-->null
*****************************


Given a singly linked list:
11-->12-->13-->14-->null
press 1 for deletion from the beginning
press 2 for deletion from the last
press 3 for delete from the certain position
enter your choice     3
pos invalid
invalid pos
12-->14-->null
*****************************

Given a singly linked list:
11-->12-->13-->14-->null
press 1 for deletion from the beginning
press 2 for deletion from the last
press 3 for delete from the certain position
enter your choice     4
please enter your choice properly
```

3.4.PROGRAM 4: Search an element from a Linked List.

PSEUDOCODE:

```
public boolean search(int key)
   {
      node temp=head;
      while(temp!=null)
      {
         if(temp.data==key)
         {
            return true;
         }
         temp=temp.next;
      }
      return false;
    }
```

SOURCE CODE:

```
class SllForSearch
{
   private static class node
   {
      int data;
      node next;
      node(int data)
      {
         this.data=data;
         this.next=null;
      }
   }
   public node head=null;

   public boolean search(int key)
   {
      node temp=head;
      while(temp!=null)
      {
         if(temp.data==key)
         {
            return true;
         }
         temp=temp.next;
      }
      return false;
    }

   public void display()
   {
      node temp=head;
      while(temp!=null)
      {
         System.out.print(temp.data+"-->");
         temp=temp.next;
      }
      System.out.print(temp+"\n");

System.out.println("**********
**********");
      System.out.print("\n");
    }

   public static void main(String args[])
   {
      SllForSearch s=new SllForSearch();
      s.head= new node(11);
      node second=new node(12);
      node third=new node(13);
      node fourth=new node(14);
      s.head.next=second;
      second.next=third;
      third.next=fourth;
      System.out.println("Given singly
linked list:");
      s.display();
      System.out.println("Search 11");
```

```
        if(s.search(11))
        {
            System.out.println("element
found");
        }
        else{
            System.out.println("Element not
found");
        }
        System.out.println("Search 1");
        if(s.search(1))
        {
            System.out.println("element
found");
        }
        else{
            System.out.println("Element not
found");
        }

    }
}
```

OUTPUT:

```
Given singly linked list:
11-->12-->13-->14-->null
*******************************

Search 11
element found
Search 1
Element not found
```

4.OUTPUT AND DISCUSSION:

The output of program 1 shows that when the traverse() method is run the elements of the singly list are displayed in the output screen.

The output of the Program 2 shows that when the insertatbeg(x) method is run the entered values are inserted at the beginning in the singly linked list. Similarly, when the insertatlast(x) method is run, the head is null so the entered node automatically becomes the last and the following nodes entered inserts at the end in singly linked list manner. Finally, when the insertatpos(x,pos) is run, pos<1or more than the size of nodes in singly linked list it displays invalid result. Furthermore, it also shows that if any valid or specific position is entered, it inserts the node at the specific position in the singly linked list.

In the output of Program 3, it can be seen that when the user choose choice 1, it takes to deleteatbeg() method and deletes the first node of the linked list. Similarly, when choose 2, deleteatlast() is run which deletes the last node of the list. Besides, when choose 3, deleteatpos(pos) method is run and checking various conditions it delete a node from the specified position. Finally, if any number other than 1,2,3 is entered, a message to enter the proper choice is displayed.

Finally, in the output of Program 4, it is seen that any node can be searched from a given linked list by traversing to each of the node and returning true if the desired key node is found.

Therefore, The output of program 1, 2 and 3 supports the traverse of a nodes in a linked list, insertion of nodes, deletion of nodes and searching of a node from the singly linked list.

5.CONCLUSION:

A Java program to traverse, insert a node at first, last, certain position, delete a node at beginning, at last and at certain position and search a node from the singly linked list is successfully run.

EXPERIMENT: 2

TITLE:

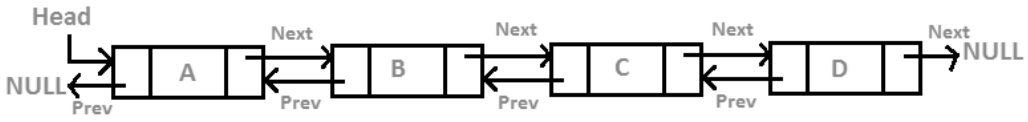
A java program to run various operations of doubly linked list.

1.OBJECTIVE:

- ✓ To insert a node at beginning, at last and at certain position and display it.
- ✓ To delete a node from the beginning, from last and from certain position.
- ✓ To sort the nodes from doubly linked list.

2.THEORY:

Doubly linked list is the data structure used for storing collection of nodes and the node has one data field and two fields for reference to previous node and next node in a list. The first node is the head node and its prev field is null and the last node has data and points to null. The various operations of doubly linked list are discussed below:



2.1. Traverse:

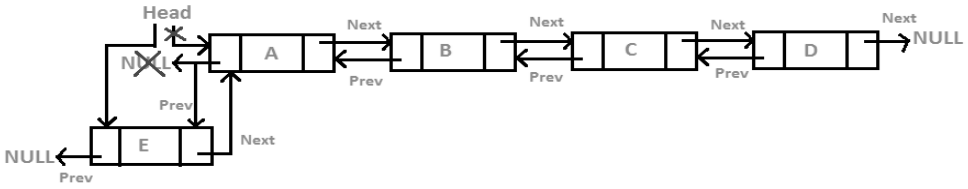
The traverse operation helps to display the content of a doubly linked list. Here, to run this operation, the temp node is kept moving to the next one and the content is displayed.

2.2.Insertion:

The insertion operation inserts a node in linked list. The insertion operation in doubly linked list can be run in three different methods. They are:

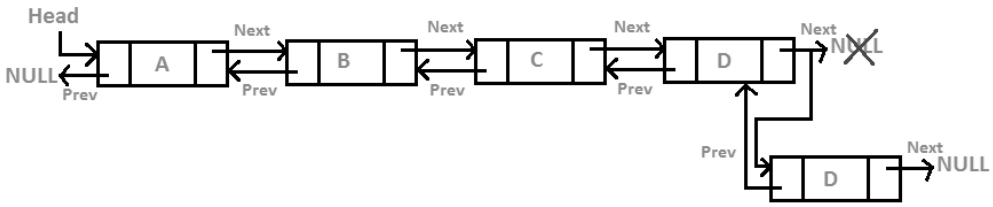
2.2.1.insertatbeg(int x):

In this method, a node is inserted at the beginning of the doubly linked list.



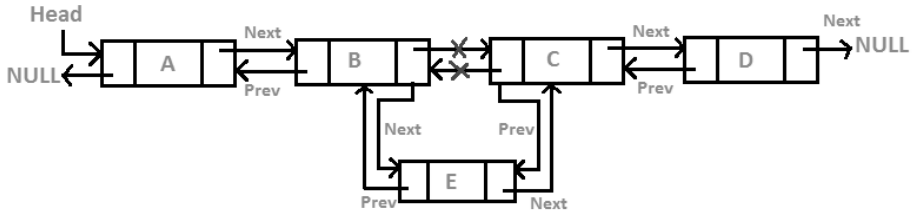
2.2.2.insertatlast(int x):

In this method, a node is inserted at the last of the linked list.



2.2.3.insertatpos(int x, int pos):

This method inserts a node at the specified position of the linked list.

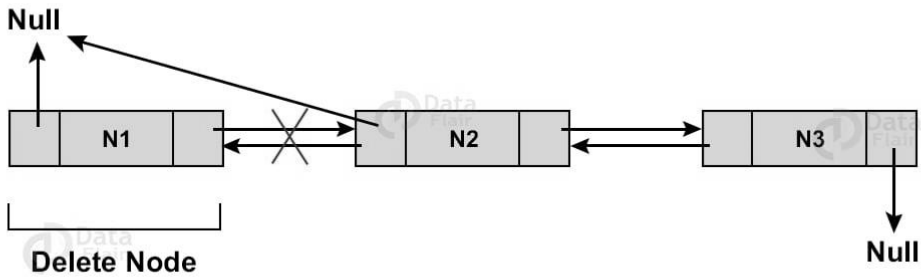


2.3. Deletion:

The deletion operation deletes a node from the linked list. The deletion operation in doubly linked list can also be run in three different methods. They are:

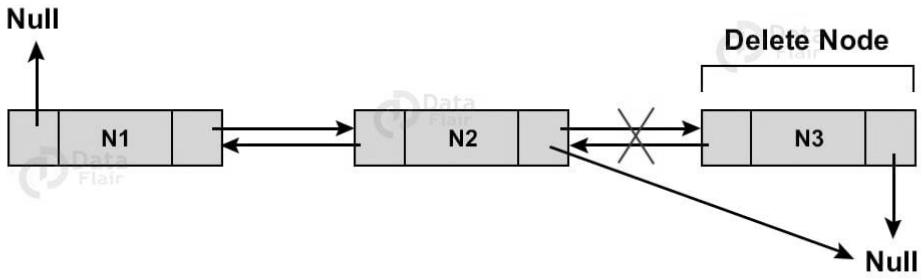
2.3.1.deleteatbeg():

In this method, a node is deleted from the very beginning of the doubly linked list.



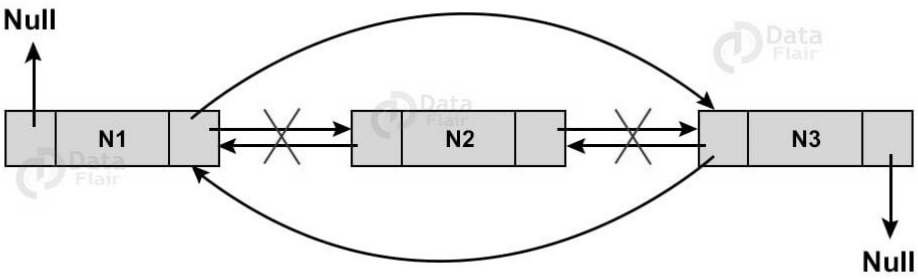
2.3.2.deleteatlast():

In this method, a node is deleted from the last of the doubly linked list.



2.3.3.deleteatpos(int pos):

In this method, a node is deleted from the specified position of the linked list.



2.4. Sort:

This method arranges the data of a node in either ascending or descending order.

3.IMPLEMENTATION:

The following programs shows the insertion of nodes at the beginning, at last and at certain position in doubly linked list data structure in java. Similarly, the deletion of node from the beginning, from the last and from the specified position is also shown. Moreover, the source code and output for sort operation is also shown.

3.1.PROGRAM 1: Traverse a doubly linked list.

PSEUDOCODE:

```
public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else
      {
         node temp=head;
         System.out.print("null<-->");
         while(temp!=null)
         {
            System.out.print(temp.data+"<-->");
            temp=temp.next;
         }
         System.out.print(temp);
      }
   }
```

SOURCECODE:

```
class dlldisplay
{
   class node
   {
      int data;
      node prev;
      node next;
      node(int data)
      {
         this.data=data;
         this.prev=null;
         this.next=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         head.prev=newnode;
         newnode.next=head;
         head=newnode;
      }
   }
   public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else
      {
         node temp=head;
         System.out.print("null<-->");
         while(temp!=null)
         {
            System.out.print(temp.data+"<-->");
            temp=temp.next;
         }
         System.out.print(temp);

      }
   }
  public static void main(String args[])
   {
      dlldisplay d=new dlldisplay();
      d.add(5);
      d.add(10);
      d.add(15);
      d.display();
   }

}
```

OUTPUT:

```
A doubly linked list traversed as:
null<-->15<-->10<-->5<-->null
```

3.2 PROGRAM 2: Insert element to a linked list either at the beginning, middle or end of the linked list.

3.2.1. Insert at the beginning:

PSEUDOCODE:

```
public void insertatbeg(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         head.prev=newnode;
         newnode.next=head;
         head=newnode;
      }
   }
```

3.2.2. Insert at the last:

PSEUDOCODE:

```
public void insertatlast(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         node temp=head;
         while(temp.next!=null)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.prev=temp;
      }
   }
```

3.2.3. Insert at a specified position:

PSEUDOCODE:

```
public void insertatpos(int x,int pos)
   {
      node newnode=new node(x);
      if(pos<1)
      {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            head=newnode;
         }
         else
         {
            newnode.next=head;
            head.prev=newnode;
            head=newnode;
         }
      }
      else{
         node temp=head;
         for(int i=1;i<pos-
1&&temp!=null;i++) {
            temp=temp.next;
         }
         if(temp==null)
         {
            System.out.println("Invalid
position");
         }
         else if(temp.next==null)
         {
            temp.next=newnode;
            newnode.prev=temp;
         }
         else
         {
            newnode.next=temp.next;
            temp.next.prev=newnode;
            temp.next=newnode;
            newnode.prev=temp;
         }
      }
   }
```

SOURCE CODE OF INSERTION OPERATION:

```
import java.util.Scanner;
class dllinsert
{
   class node
   {
      int data;
      node prev;
      node next;
      node(int data)
      {
         this.data=data;
         this.prev=null;
         this.next=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         head.prev=newnode;
         newnode.next=head;
         head=newnode;
      }
   }

   public void insertatbeg(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         head.prev=newnode;
         newnode.next=head;
         head=newnode;
      }
   }

   public void insertatlast(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         node temp=head;
         while(temp.next!=null)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.prev=temp;
      }
   }

   public void insertatpos(int x,int pos)
   {
      node newnode=new node(x);
      if(pos<1)
      {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            head=newnode;
         }
         else
         {
           newnode.next=head;
           head.prev=newnode;
           head=newnode;
         }
      }
      else
      {
         node temp=head;
         for(int i=1;i<pos-
1&&temp!=null;i++)
         {
            temp=temp.next;
         }
         if(temp==null)
```

```
        {
            System.out.println("Invalid
position");
        }
        else if(temp.next==null)
        {
            temp.next=newnode;
            newnode.prev=temp;
        }
        else
        {
            newnode.next=temp.next;
            temp.next.prev=newnode;
            temp.next=newnode;
            newnode.prev=temp;
        }

    }
  }

  public void display()
  {
    if(head==null)
    {
        System.out.println("empty list");
    }
    else
    {
        node temp=head;
        System.out.print("null<-->");
        while(temp!=null)
        {
            System.out.print(temp.data+"<--
>");
            temp=temp.next;
        }
        System.out.println(temp);

    }
  }

  public static void main(String args[])
  {
    dllinsert d=new dllinsert();
    Scanner input=new
Scanner(System.in);
    d.add(1);
    d.add(2);
    d.add(3);
    System.out.println("This is doubly
linked list:");
    d.display();
    int ch;
    do
    {
        System.out.println();
        System.out.println("Where do you
want to insert your newnode?");
        System.out.println("At the
beginning? Press 1");
        System.out.println("At the last?
Press 2");
        System.out.println("At your
specified position? Press 3");
        ch=input.nextInt();
        switch(ch)
        {
            case 1:
            {
                System.out.println("Inserting
a node at beginning.");
                System.out.println("What is
the data of your newnode?");
                int a=input.nextInt();
                d.insertatbeg(a);
                d.display();
                break;
            }
            case 2:
            {
                System.out.println("Inserting
a node at last.");
                System.out.println("What is
the data of your newnode?");
                int a=input.nextInt();
                d.insertatlast(a);
                d.display();
                break;
            }
            case 3:
            {
                System.out.println("Inserting
a node at specified position");
                System.out.println("What is
the data of your newnode?");
                int a=input.nextInt();
                System.out.println("What is
the position of your newnode?");
                int b=input.nextInt();
```

```
    d.insertatpos(a,b);
    d.display();
    break;
}
default:
{
                System.out.println("Enter your choice from 1 to 3.");
            }
        }
    }while(ch<=3);
  }
}
```

OUTPUT:

```
This is doubly linked list:
null<-->3<-->2<-->1<-->null

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
1
Inserting a node at beginning.
What is the data of your newnode?
4
null<-->4<-->3<-->2<-->1<-->null
*****************************************

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
2
Inserting a node at last.
What is the data of your newnode?
6
null<-->4<-->3<-->2<-->1<-->6<-->null
*********************************************

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
3
Inserting a node at specified position
What is the data of your newnode?
7
What is the position of your newnode?
```

```
-1
pos<1 is invalid
null<-->4<-->3<-->2<-->1<-->6<-->null

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
3
Inserting a node at specified position
What is the data of your newnode?
7
What is the position of your newnode?
1
null<-->7<-->4<-->3<-->2<-->1<-->6<-->null

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
3
Inserting a node at specified position
What is the data of your newnode?
5
What is the position of your newnode?
5
null<-->7<-->4<-->3<-->2<-->5<-->1<-->6<-->null

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
3
Inserting a node at specified position
What is the data of your newnode?
8
What is the position of your newnode?
8
null<-->7<-->4<-->3<-->2<-->5<-->1<-->6<-->8<-->null

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
```

```
3
Inserting a node at specified position
What is the data of your newnode?
10
What is the position of your newnode?
10
invalid position
null<-->7<-->4<-->3<-->2<-->5<-->1<-->6<-->8<-->null
**********************************************

Where do you want to insert your newnode?
At the beginning? Press 1
At the last? Press 2
At your specified position? Press 3
4
Enter your choice from 1 to 3.
```

3.3. PROGRAM 3: Delete element from a linked list either from the beginning, middle or end of the linked list.

3.3.1. Deletion from the beginning:

PSEUDOCODE:

```
public void deleteatbeg()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         head=head.next;
         head.prev=null;
      }
   }
```

3.3.2. Deletion from the last:

PSEUDOCODE:

```
public void deleteatlast()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         node temp=head;
         while(temp.next.next!=null)
         {
            temp=temp.next;
         }
         temp.next=null;
      }
   }
```

3.3.3. Deletion from the specified position*:*

PSEUDOCODE:

```
public void deleteatpos(int pos)
   {
      if(pos<1)
      {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            System.out.println("Empty list");
         }
         else if(head.next==null)
         {
            head=null;
         }
         else
         {
            head=head.next;
            head.prev=null;
         }
      }
      else
      {
         node temp=head;
         for(int i=1;i<pos-
1&&temp.next!=null;i++)
         {
            temp=temp.next;
         }
         if(temp.next!=null)
         {
            temp.next=temp.next.next;
            temp.next.prev=temp;
         }
         else{
            System.out.println("invalid
position");
         }
      }
   }
```

SOURCE CODE OF DELETE OPERATION:

```
import java.util.Scanner;
class dlldelete
{
   class node
   {
      int data;
      node prev;
      node next;
      node(int data)
      {
         this.data=data;
         this.prev=null;
         this.next=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         head.prev=newnode;
         newnode.next=head;
         head=newnode;
      }
   }

   public void deleteatbeg()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         head=head.next;
         head.prev=null;
      }
   }

   public void deleteatlast()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         node temp=head;
         while(temp.next.next!=null)
         {
            temp=temp.next;
         }
         temp.next=null;
      }
   }

   public void deleteatpos(int pos)
   {
      if(pos<1)
      {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            System.out.println("Empty list");
         }
         else if(head.next==null)
         {
            head=null;
         }
         else
         {
            head=head.next;
            head.prev=null;
         }
      }
      else
      {
         node temp=head;
```

```
        for(int i=1;i<pos-
1&&temp.next!=null;i++)
        {
           temp=temp.next;
        }
        if(temp.next!=null)
        {
           temp.next=temp.next.next;
           temp.next.prev=temp;
        }
        else
        {
           System.out.println("invalid
position");
        }
     }
  }

  public void display()
  {
     if(head==null)
     {
        System.out.println("empty list");
     }
     else
     {
        node temp=head;
        System.out.print("null<-->");
        while(temp!=null)
        {
           System.out.print(temp.data+"<--
>");
           temp=temp.next;
        }
        System.out.println(temp);

     }
  }

  public static void main(String args[])
  {
     dlldelete d=new dlldelete();
     Scanner input=new
Scanner(System.in);
     d.add(1);
     d.add(2);
     d.add(3);
     d.add(4);
     d.add(5);
     d.add(6);
     System.out.println("This is doubly
linked list:");
     d.display();
     int ch;
     do
     {
        System.out.println();
        System.out.println("From where do
you want to delete your newnode?");
        System.out.println("From the
beginning? Press 1");
        System.out.println("From the last?
Press 2");
        System.out.println("From the
specified position? Press 3");
        ch=input.nextInt();
        switch(ch)
        {
           case 1:
           {
              System.out.println("Deleteing
a node from the beginning.");
              d.deleteatbeg();
              d.display();
              System.out.println("********
**********************************");
              break;
           }
           case 2:
           {
              System.out.println("Delete a
node at last.");
              d.deleteatlast();
              d.display();
              System.out.println("********
**********************************");
              break;
           }
           case 3:
           {
              System.out.println("Delete a
node at specified position");
              System.out.println("What is
the position of your newnode?");
              int b=input.nextInt();
              d.deleteatpos(b);
              d.display();
              break;
           }
           default:
```

```
            {                                                   }
                System.out.println("Enter                   }while(ch<=3);
your choice from 1 to 3.");                             }
            }                                       }
```

OUTPUT:

```
This is doubly linked list:
null<-->6<-->5<-->4<-->3<-->2<-->1<-->null

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
1
Deleteing a node from the beginning.
null<-->5<-->4<-->3<-->2<-->1<-->null
************************************************************

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
2
Delete a node at last.
null<-->5<-->4<-->3<-->2<-->null
************************************************************

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
3
Delete a node at specified position
What is the position of your newnode?
-1
pos<1 is invalid
null<-->5<-->4<-->3<-->2<-->null

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
3
```

```
Delete a node at specified position
What is the position of your newnode?
2
null<-->5<-->3<-->2<-->null

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
3
Delete a node at specified position
What is the position of your newnode?
4
invalid position
null<-->5<-->3<-->2<-->null

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
4
Enter your choice from 1 to 3.
```

3.4. PROGRAM 4: Sort the element of the linked list.

PSEUDOCODE:

```
public void sort() {
      node temp;
      node last=null;
      int swap;
      do{
         swap=0;
         temp=head;
         while(temp.next!=last)
         {
            if(temp.data>temp.next.data)
            {
               int t=temp.data;
               temp.data=temp.next.data;
               temp.next.data=t;
               swap=1;
            }
            temp=temp.next;
         }
         last=temp;
      }while(swap!=0);
   }
```

SOURCECODE:

```
class dllsort
{
   class node
   {
      int data;
      node next;
      node prev;
      node(int data)
      {
         this.data=data;
         this.next=null;
         this.prev=null;
      }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
      }
      else
      {
         newnode.next=head;
         head.prev=newnode;
         head=newnode;
      }
   }

   public void sort()
   {
      node temp;
      node last=null;
      int swap;
      do
      {
         swap=0;
         temp=head;
         while(temp.next!=last)
         {
            if(temp.data>temp.next.data)
            {
               int t=temp.data;
               temp.data=temp.next.data;
               temp.next.data=t;
               swap=1;
            }
            temp=temp.next;
         }
         last=temp;
      }while(swap!=0);
   }

   public void display()
   {
      node temp=head;
      System.out.print("null<-->");
      while(temp!=null)
      {
         System.out.print(temp.data+"<-->");
         temp=temp.next;
      }
      System.out.println(temp);
```

```
    }

    public static void main(String args[])
    {
        dllsort d=new dllsort();
        d.add(1);
        d.add(2);
        d.add(3);
       System.out.println("The linked list
before sorting:");
        d.display();
        d.sort();
        System.out.println("The linked list
after sorting:");
       d.display();
    }
}
```

OUTPUT:

```
The linked list before sorting:
null<-->3<-->2<-->1<-->null

The linked list after sorting:
null<-->1<-->2<-->3<-->null
```

4. OUTPUT AND DISCUSSION:

- ✓ In this experiment, the output of program 1 shows the traverse of a node to display the data of the node in the doubly linked list.
- ✓ In the output of program 2, it can be seen that the insertion of a node is done at the beginning of the linked list, after then, when the user press 2, a node is inserted at the last and then when the user press option 3, a node is inserted at a specified position.
- ✓ The output of the Program 3 shows the deletion of a node from the beginning, from the last and from the specified position.
- ✓ In program 4, the sorting program for linked list is done in which the data of the doubly linked list are arranged in ascending order.

5. CONCLUSION:

A Java program to traverse, insert a node at first, last, certain position, delete a node at beginning, at last and at certain position and sorting of the data of doubly linked list is successfully run.

EXPERIMENT: 3

TITLE:

A java program to run various operations of circular singly linked list.

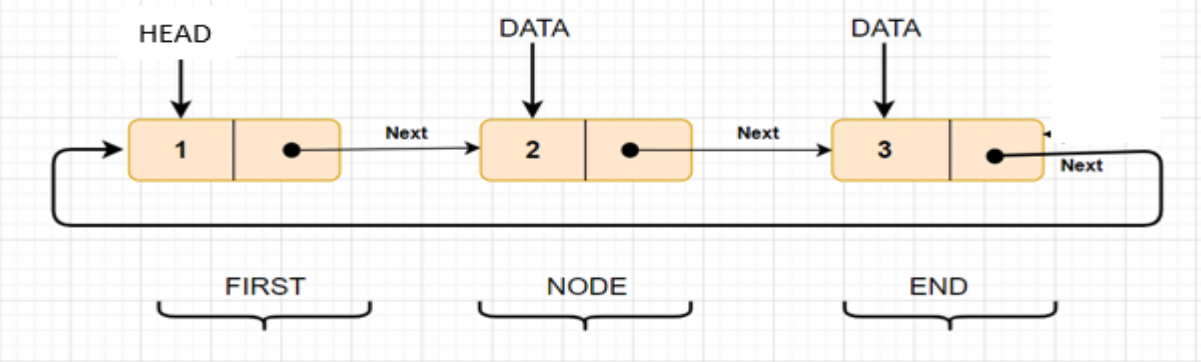
1.OBJECTIVE:

- ✓ To insert a node at beginning, at last and at certain position and display it.
- ✓ To delete a node from the beginning, from last and from certain position.
- ✓ To count the nodes in the circular singly linked list.

2.THEORY:

Circular singly linked list is the data structure used for storing collection of nodes and the node has one data field and a field for reference to next node in a list. The first node is the head node and it has data and points to next node and the last node has data and points to head.

The various operations of circular singly linked list are discussed below:



2.1. Traverse:

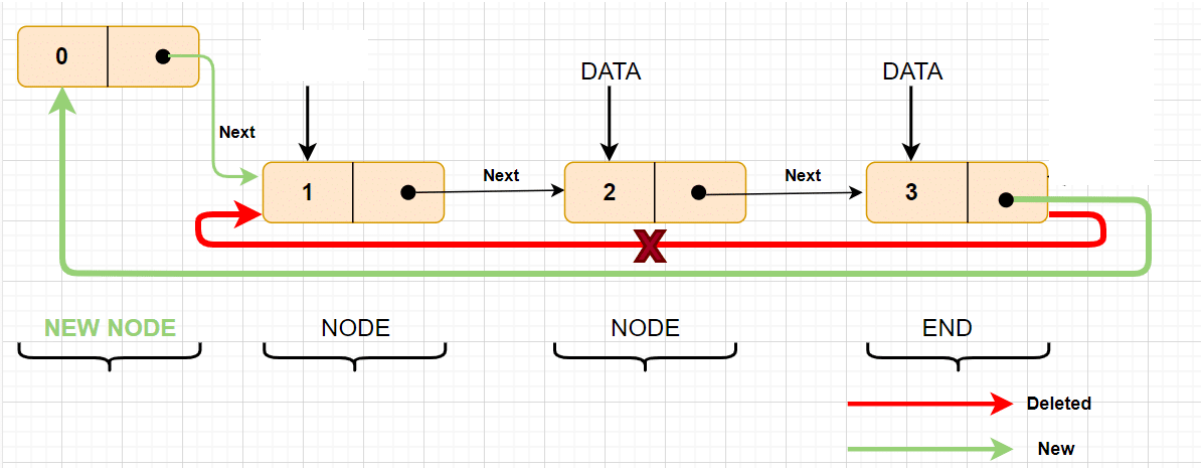
The traverse operation helps to display the content of a circular singly linked list. Here, to run this operation, the temp node is kept moving to the next one and the content is displayed.

2.2.Insertion:

The insertion operation inserts a node in linked list. The insertion operation in circular singly linked list can be run in three different methods. They are:

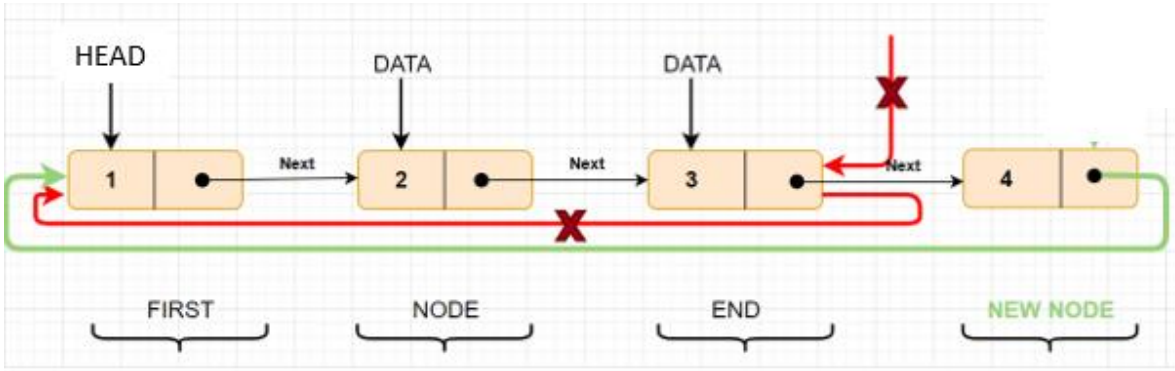
2.2.1.insertatbeg(int x):

In this method, a node is inserted at the beginning of the circular singly linked list.



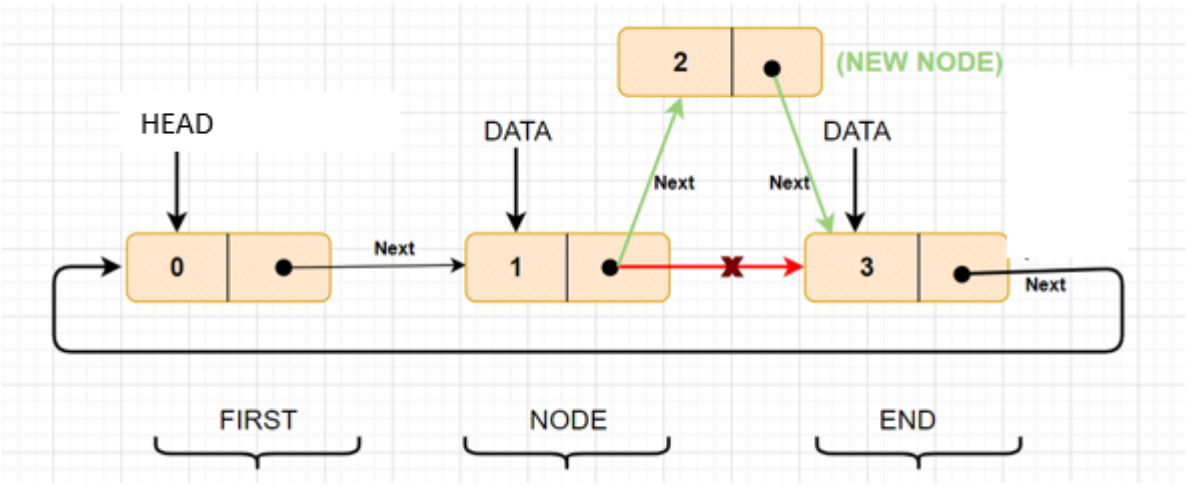
2.2.2.insertatlast(int x):

In this method, a node is inserted at the last of the linked list.



2.2.3.insertatpos(int x, int pos):

This method inserts a node at the specified position of the linked list.

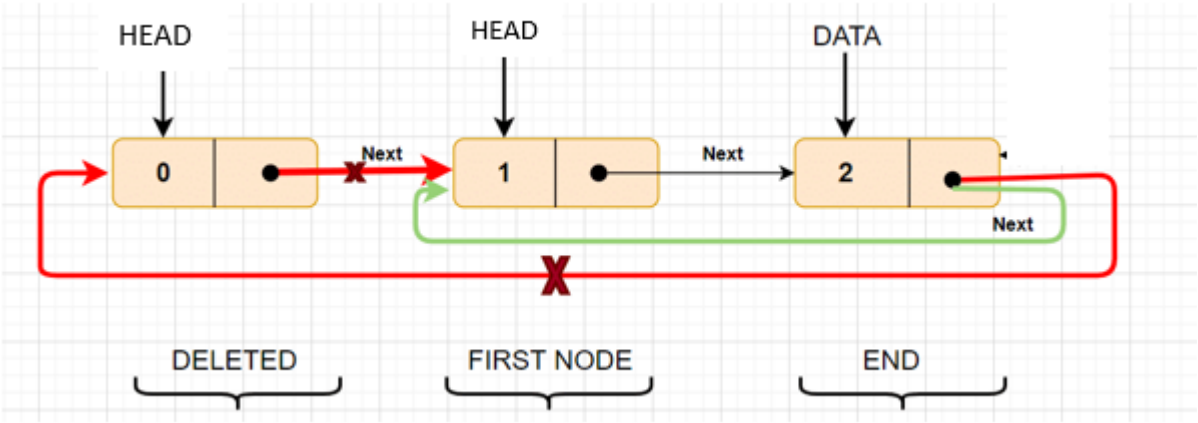


2.3. Deletion:

The deletion operation deletes a node from the linked list. The deletion operation in circular singly linked list can also be run in three different methods. They are:

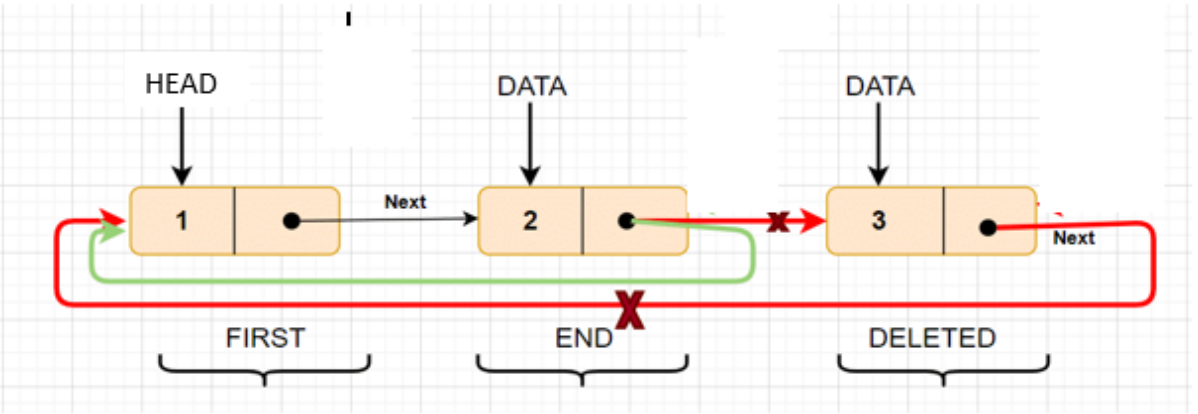
2.3.1.deleteatbeg():

In this method, a node is deleted from the very beginning of the circular singly linked list.



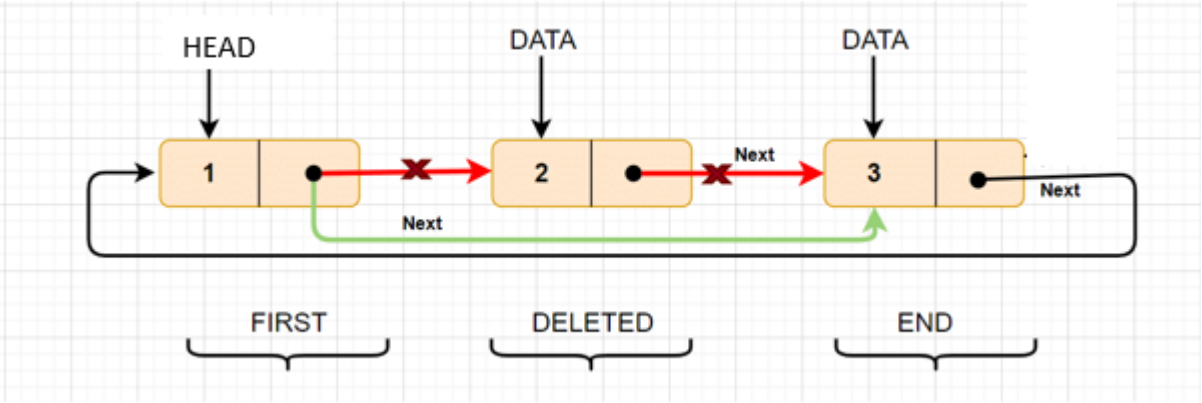
2.3.2.deleteatlast():

In this method, a node is deleted from the last of the circular singly linked list.



2.3.3.deleteatpos(int pos):

In this method, a node is deleted from the specified position of the linked list.



2.4. Count:

This method counts the number of nodes in the given circular singly linked list.

3.IMPLEMENTATION:

The following programs shows the insertion of nodes at the beginning, at last and at certain position in circular singly linked list data structure in java. Similarly, the deletion of node from the beginning, from the last and from the specified position is also shown. Moreover, the source code and output for count operation is also shown.

3.1.PROGRAM 1: Traverse a circular singly linked list.

PSEUDOCODE:

```
public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else{
         node temp=head;
         while(temp.next!=head){
            System.out.print(temp.data+"-->");
            temp=temp.next;
         }
         System.out.print(temp.data);

      }
      node temp=head;
   }
```

SOURCECODE:

```
class csllltraverse
{
   class node
   {
      int data;
      node next;
      node(int data)
      {
         this.data=data;
         this.next=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null){
         head=newnode;
         head.next=head;
      }
      else {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
         head=newnode;
      }
   }
   public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else {
         node temp=head;
         while(temp.next!=head)
         {
            System.out.print(temp.data+"-->");
            temp=temp.next;
         }
         System.out.print(temp.data);

      }
      node temp=head;
   }

   public static void main(String args[])
   {
      csllltraverse d=new csllltraverse();
      System.out.println("A circular singly linked list traversed as:");
      d.add(5);
      d.add(10);
      d.display();
   }

}
```

OUTPUT:

```
A circular singly linked list traversed as:
15-->10-->5
```

3.2 PROGRAM 2: Insert element to a linked list either at the beginning, middle or end of the linked list.

3.2.1. Insert at the beginning:

PSEUDOCODE:

```
public void insertatbeg(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.next=head;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
         head=newnode;
      }
 }
```

3.2.2. Insert at the last:

PSEUDOCODE:

```
public void insertatlast(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.next=head;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
      }
   }
```

3.2.3. Insert at a specified position:

PSEUDOCODE:

```
public void insertatpos(int x,int pos) {
      node newnode=new node(x);
      if(pos<1){
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1) {
         if(head==null){
            head=newnode;
            head.next=head;
         }
         else{
            node temp=head;
            while(temp.next!=head){
               temp=temp.next;
            }
            temp.next=newnode;
            newnode.next=head;
            head=newnode;
         }
      }
      else if(pos>(count()+1)) {
            System.out.println("invalid
position");
         }
      else{
         node temp=head;
         int i;
         for(i=1;i<pos-
1&&i<=count();i++){
            temp=temp.next;
         }
         if(temp.next!=head){
            newnode.next=temp.next;
            temp.next=newnode;
         }
         else {
            temp.next=newnode;
            newnode.next=head;
         }
      }
   }
```

SOURCE CODE OF INSERTION OPERATON:

```
import java.util.Scanner;
class csllinsert
{
   class node
   {
      int data;
      node next;
      node(int data)
      {
         this.data=data;
         this.next=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.next=head;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
         head=newnode;
      }
   }

   public void insertatbeg(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.next=head;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
         head=newnode;
      }
   }

   public void insertatlast(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.next=head;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
      }
   }



   public void insertatpos(int x,int pos)
   {
      node newnode=new node(x);
      if(pos<1)
      {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            head=newnode;
            head.next=head;
         }
         else
         {
            node temp=head;
            while(temp.next!=head)
```

```
            {
               temp=temp.next;
            }
            temp.next=newnode;
            newnode.next=head;
            head=newnode;
         }
      }
      else if(pos>(count()+1))
         {
            System.out.println("invalid
position");
         }
      else
      {
         node temp=head;
         int i;
         for(i=1;i<pos-
1&&i<=count();i++)
         {
            temp=temp.next;
         }
         if(temp.next!=head)
         {
            newnode.next=temp.next;
            temp.next=newnode;
         }
         else
         {
            temp.next=newnode;
            newnode.next=head;
         }

      }
   }

   public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            System.out.print(temp.data+"--
>");
            temp=temp.next;
         }
         System.out.print(temp.data);

      }
   }

   public int count()
   {
      int c=0;
      if(head==null)
      {
         return c;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            c=c+1;
            temp=temp.next;
         }
         return c+1;
      }
   }

   public static void main(String args[])
{
      csllinsert s=new csllinsert();
      System.out.println("A circular singly
linked list is given as:");
      s.add(5);
      s.add(10);
      s.add(15);
      s.display();

      int ch;
      do{
      System.out.println();
      System.out.println("press 1 for
insertion at beginning");
      System.out.println("press 2 for
insertion at last");
      System.out.println("press 3 for
insertion at position");
      System.out.println("enter your
choice");
      Scanner input=new
Scanner(System.in);
      ch=input.nextInt();
      switch(ch)
```

```
    {
        case 1:
        {
            s.insertatbeg(9);
            s.display();
            break;

        }
        case 2:
        {
            s.insertatlast(11);
            s.display();
            break;
        }
        case 3:
        {
            System.out.println("Insertion at
pos<1");
            s.insertatpos(1, -1);
            s.display();
            System.out.println();
            System.out.println("Insertion at
pos=1");
            s.insertatpos(1,1);
            s.display();
            System.out.println();
            System.out.println("Insertion at
position 2");
            s.insertatpos(3,2);
            s.display();
            System.out.println();
            System.out.println("Insertion at
position 10");
            s.insertatpos(10,10);
            s.display();
            break;
        }
        default:
        {
            System.out.println("please enter
your choice properly");
        }
    }
    }while(ch<=3);
  }
}
```

OUTPUT:

```
A circular singly linked list is given as:
15-->10-->5

press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
1
9-->15-->10-->5



press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
2
9-->15-->10-->5-->11
```

```
press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
3
Insertion at pos<1
pos<1 is invalid
9-->15-->10-->5-->11

Insertion at pos=1
1-->9-->15-->10-->5-->11

Insertion at position 2
1-->3-->9-->15-->10-->5-->11

Insertion at position 10
invalid position
1-->3-->9-->15-->10-->5-->11

press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
4
please enter your choice properly
```

3.3. PROGRAM 3: Delete element from a linked list either from the beginning, middle or end of the linked list.

3.3.1. Deletion from the beginning:

PSEUDOCODE:

```
public void deleteatbeg()
  {
     if(head==null){
        System.out.println("Empty list");
     }
     else if(head.next==null)
     {
        head=null;
     }
     else
     {
        node temp=head;
        while(temp.next!=head)
        {
           temp=temp.next;
        }
        temp.next=head.next;
        head=head.next;
     }
  }
```

3.3.2. Deletion from the last:

PSEUDOCODE:

```
public void deleteatlast()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         node temp=head;
         while(temp.next.next!=head)
         {
            temp=temp.next;
         }
         temp.next=head;
      }
   }
```

3.3.3. Deletion from the specified position*:*

PSEUDOCODE:

```
public void deleteatpos(int pos)
   {
      if(pos<1){
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1){
         if(head==null){
            System.out.println("Empty
list");
         }
         else if(head.next==null)
         {
            head=null;
         }
         else{
            node temp=head;
            while(temp.next!=head)
            {
               temp=temp.next;
            }
            temp.next=head.next;
            head=head.next;
         }
      }
      else{
         node temp=head;
         for(int i=1;i<pos-
1&&temp.next!=head;i++)
         {
            temp=temp.next;
         }
         if(temp.next!=head)
         {
            temp.next=temp.next.next;
         }
         else{
            System.out.println("invalid
position");
         }
      }
   }
```

SOURCE CODE FOR DELETION OPERATION:

```
import java.util.Scanner;
class csslldelete
{
   class node
   {
      int data;
      node next;
      node(int data)
      {
         this.data=data;
         this.next=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null){
         head=newnode;
         head.next=head;
      }
      else {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
         head=newnode;
      }
   }

   public void deleteatbeg()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=head.next;
         head=head.next;
      }
   }

   public void deleteatlast()
   {
      if(head==null){
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else {
         node temp=head;
         while(temp.next.next!=head)
         {
            temp=temp.next;
         }
         temp.next=head;
      }
   }

    public void deleteatpos(int pos)
   {
      if(pos<1) {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            System.out.println("Empty list");
         }
         else if(head.next==null)
         {
            head=null;
         }
         else {
            node temp=head;
            while(temp.next!=head) {
               temp=temp.next;
            }
            temp.next=head.next;
            head=head.next;
         }
```

```
        }
        else {
            node temp=head;
            for(int i=1;i<pos-
1&&temp.next!=head;i++)
            {
                temp=temp.next;
            }
            if(temp.next!=head)
            {
                temp.next=temp.next.next;
            }
            else {
                System.out.println("invalid
position");
            }
        }
    }

    public void display()
    {
        if(head==null){
            System.out.println("empty list");
        }
        else{
            node temp=head;
            while(temp.next!=head) {
                System.out.print(temp.data+"←
>");
                temp=temp.next;
            }
            System.out.print(temp.data);

        }
    }

    public static void main(String args[])
    {
        cslldelete d=new cslldelete();
        Scanner input=new
Scanner(System.in);
        d.add(1);
        d.add(2);
        d.add(3);
        d.add(4);
        d.add(5);
        d.add(6);
        System.out.println("This is circular
singly linked list:");
        d.display();
        int ch;
        do{
            System.out.println();
            System.out.println("From where
do you want to delete your newnode?");
            System.out.println("From the
beginning? Press 1");
            System.out.println("From the last?
Press 2");
            System.out.println("From the
specified position? Press 3");
            ch=input.nextInt();
            switch(ch)
            {
                case 1:
                {
                    System.out.println("Deleteing
a node from the beginning.");
                    d.deleteatbeg();
                    d.display();
                    break;
                }
                case 2:
                {
                    System.out.println("Delete a
node at last.");
                    d.deleteatlast();
                    d.display();
                    break;
                }
                case 3:
                {
                    System.out.println("Delete a
node at specified position");
                    System.out.println("What is
the position of your newnode?");
                    int b=input.nextInt();
                    d.deleteatpos(b);
                    d.display();
                    break;
                }
                default:{
                    System.out.println("Enter
your choice from 1 to 3.");
                }
            }
        }while(ch<=3);
    }
}
```

OUTPUT:

```
This is circular singly linked list:
6<-->5<-->4<-->3<-->2<-->1
From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
1
Deleting a node from the beginning.
5<-->4<-->3<-->2<-->1


From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
2
Deletion of a node at last.
5<-->4<-->3<-->2


From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
3
Delete a node at specified position
What is the position of your newnode?
-1
pos<1 is invalid
5<-->4<-->3<-->2

What is the position of your newnode?
5
invalid position
5<-->4<-->3<-->2

What is the position of your newnode?
```

```
1
4<-->3<-->2
What is the position of your newnode?
2
4<-->2


From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
4
Enter your choice from 1 to 3.
```

3.4. PROGRAM 4: Count the element of the linked list.

PSEUDOCODE:

```
public int count()
   {
      int c=0;
      if(head==null)
      {
         return c;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            c=c+1;
            temp=temp.next;
         }
         return c+1;
      }
   }
```

SOURCECODE:

```
class csllcount
{
   class node
   {
      int data;
      node next;
      node(int data)
      {
         this.data=data;
         this.next=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.next=head;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            temp=temp.next;
         }
         temp.next=newnode;
         newnode.next=head;
         head=newnode;
      }
   }
   public int count()
   {
      int c=0;
      if(head==null)
      {
         return c;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            c=c+1;
            temp=temp.next;
         }
         return c+1;
      }
   }

   public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            System.out.print(temp.data+"--
>");
```

```
            temp=temp.next;
        }
        System.out.print(temp.data);
    }
  }

  public static void main(String args[])
  {
    csllcount s=new csllcount();
    System.out.println("A circular singly
linked list is given as:");
    s.add(5);
    s.add(10);
    s.add(15);
    s.display();
    System.out.println();
    System.out.println("The number of
nodes in the linked list are:"+s.count());
  }
}
```

OUTPUT:

```
A circular singly linked list is given as:
15-->10-->5
The number of nodes in the linked list are:3
```

4. OUTPUT AND DISCUSSION:

- ✓ In this experiment, the output of program 1 shows the traverse of a node to display the data of the node in the circular singly linked list.
- ✓ In the output of program 2, it can be seen that the insertion of a node is done at the beginning of the linked list, after then, when the user press 2, a node is inserted at the last and then when the user press option 3, a node is inserted at a specified position.
- ✓ The output of the Program 3 shows the deletion of a node from the beginning, from the last and from the specified position.
- ✓ In program 4, the number of nodes are counted in the given circular singly linked list..

5. CONCLUSION:

A Java program to traverse, insert a node at first, last, certain position, delete a node at beginning, at last and at certain position and counting of the nodes in circular singly linked list is successfully run.

EXPERIMENT: 4

TITLE:

A java program to run various operations of circular doubly linked list.

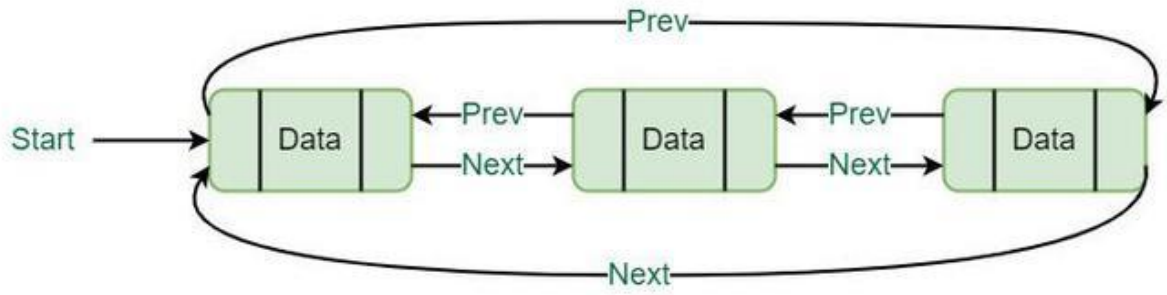
1.OBJECTIVE:

- ✓ To insert a node at beginning, at last and at certain position and display it.
- ✓ To delete a node from the beginning, from last and from certain position.
- ✓ To count the number of nodes in the linked list.

2.THEORY:

Circular doubly linked list is the data structure used for storing collection of nodes and the node has one data field and two fields for reference to previous node and next node in a list. The first node is the head node and its prev field points to the last node and the last node has data and points to head.

The various operations of circular doubly linked list are discussed below:



2.1. Traverse:

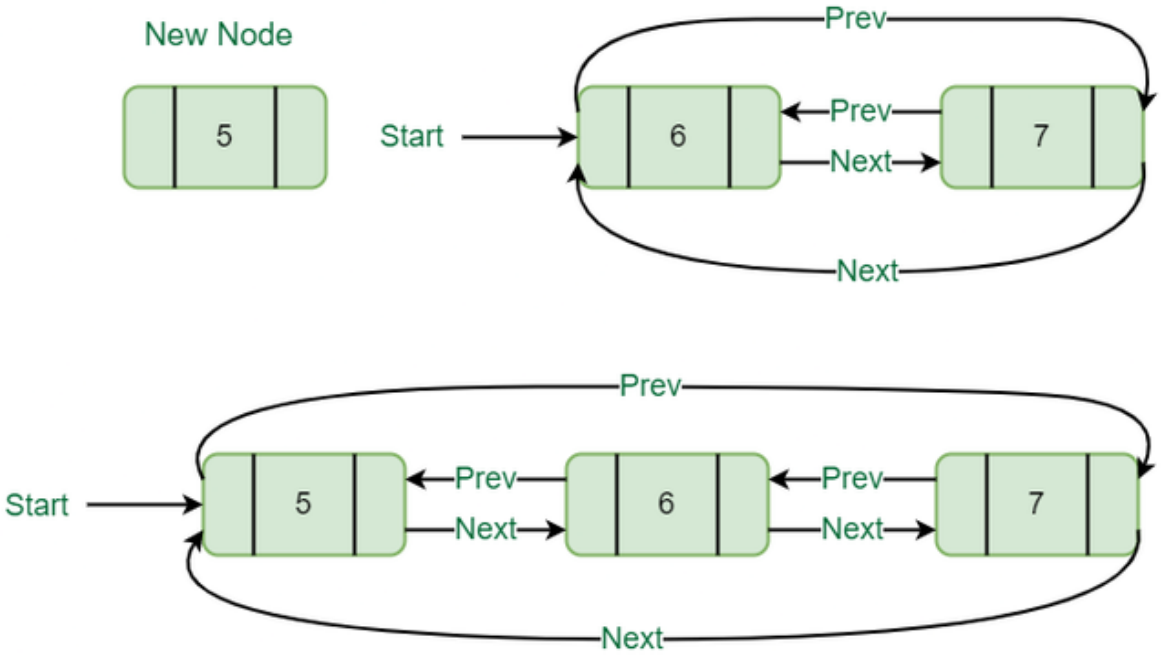
The traverse operation helps to display the content of a circular doubly linked list. Here, to run this operation, the temp node is kept moving to the next one and the content is displayed.

2.2.Insertion:

The insertion operation inserts a node in linked list. The insertion operation in circular doubly linked list can be run in three different methods. They are:

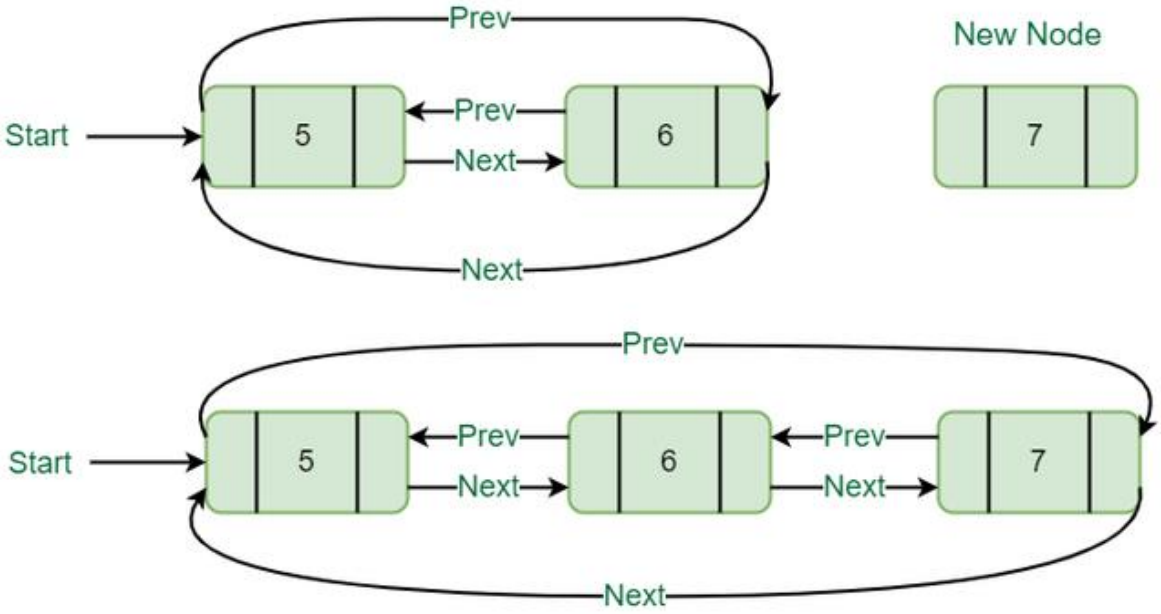
2.2.1.insertatbeg(int x):

In this method, a node is inserted at the beginning of the circular doubly linked list.



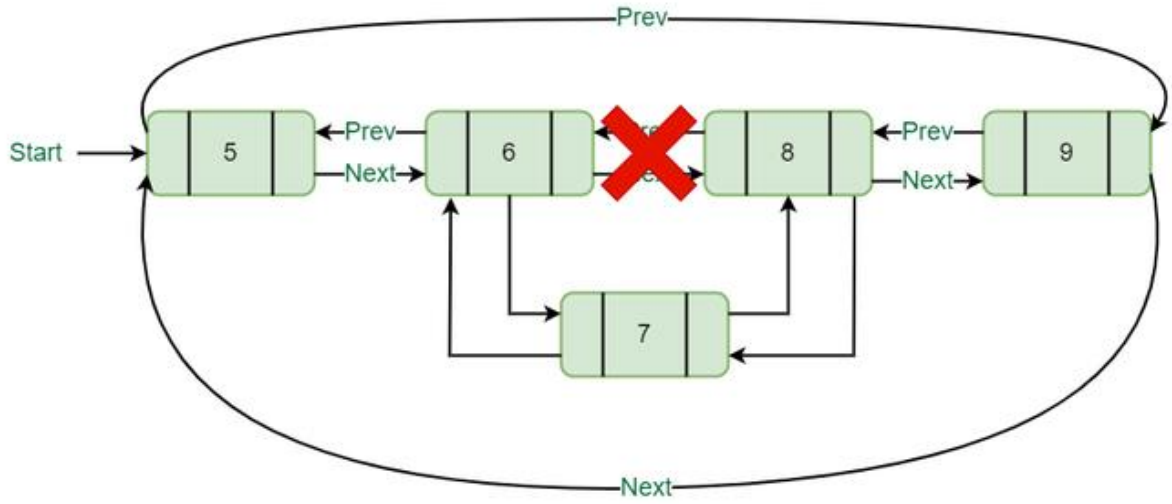
2.2.2.insertatlast(int x):

In this method, a node is inserted at the last of the linked list.



2.2.3.insertatpos(int x, int pos):

This method inserts a node at the specified position of the linked list.

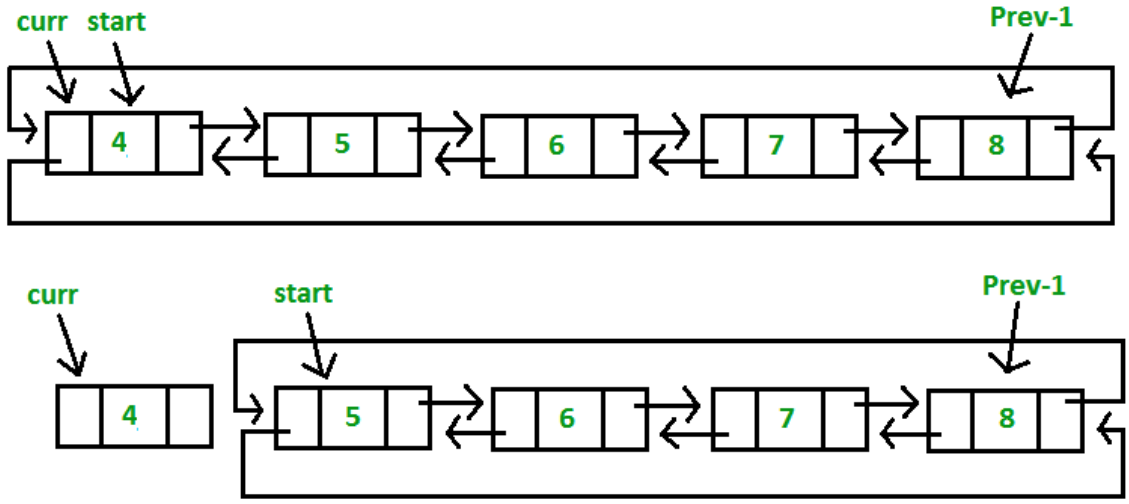


2.3. Deletion:

The deletion operation deletes a node from the linked list. The deletion operation in circular singly linked list can also be run in three different methods. They are:

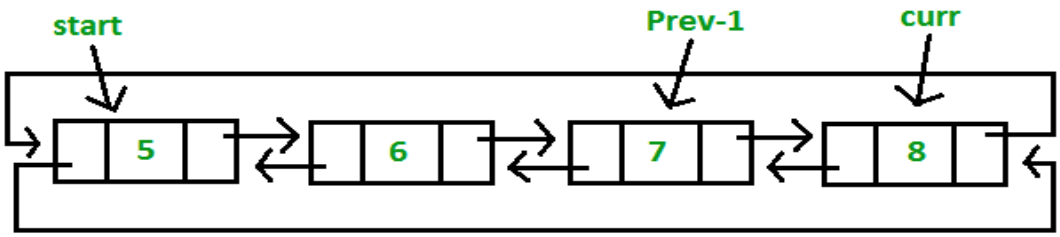
2.3.1.deleteatbeg():

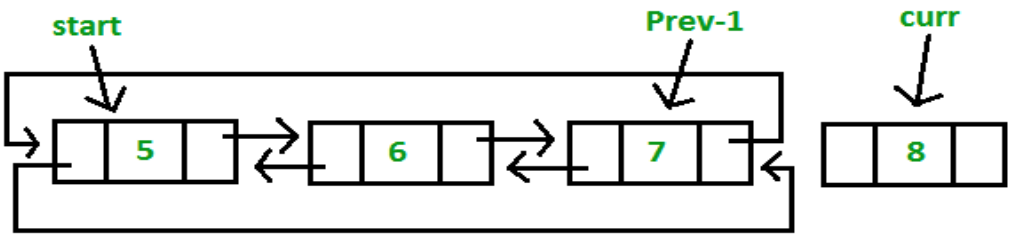
In this method, a node is deleted from the very beginning of the circular singly linked list.



2.3.2.deleteatlast():

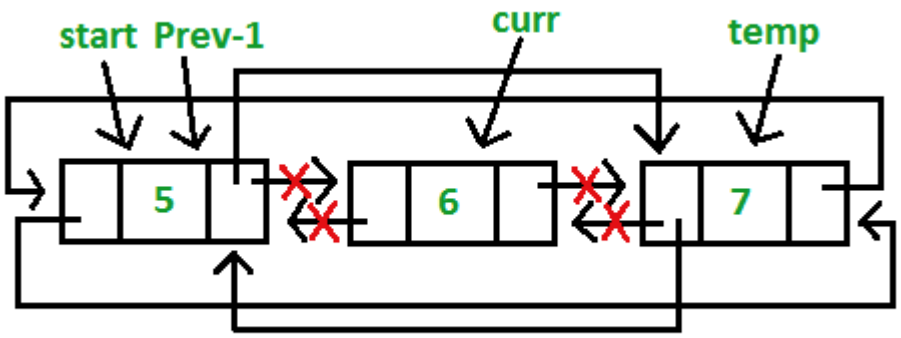
In this method, a node is deleted from the last of the circular singly linked list.





2.3.3.deleteatpos(int pos):

In this method, a node is deleted from the specified position of the linked list.



2.4. Count:

This method counts the number of nodes in the circular doubly linked list.

3.IMPLEMENTATION:

The following programs shows the insertion of nodes at the beginning, at last and at certain position in circular doubly linked list data structure in java. Similarly, the deletion of node from the beginning, from the last and from the specified position is also shown. Moreover, the source code and output for count operation is also shown.

3.1.PROGRAM 1: Traverse a circular doubly linked list.

PSEUDOCODE:

```
public void display()
  {
     if(head==null)
     {
        System.out.println("empty list");
     }
     else
     {
        node temp=head;
        while(temp.next!=head)
        {
           System.out.print(temp.data+"<-->");
           temp=temp.next;
        }
        System.out.print(temp.data);
     }
  }
```

SOURCECODE:

```
class cdlltraverse{
   class node{
      int data;
      node next;
      node prev;
      node(int data){
         this.data=data;
         this.next=null;
         this.prev=null;
;     }
   }
   public node head=null;

   public void add(int x){
      node newnode=new node(x);
      if(head==null){
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else {
         node temp=head.prev;
         temp.next=newnode;
         newnode.next=head;
         newnode.prev=temp;
         head.prev=newnode;
         head=newnode;
      }
   }

   public void display(){
      if(head==null){
         System.out.println("empty list");
      }
      else {
         node temp=head;
         while(temp.next!=head)
         {
            System.out.print(temp.data+"<-->");
            temp=temp.next;
         }
         System.out.print(temp.data);
      }
   }

   public static void main(String args[]){
      cdlltraverse d=new cdlltraverse();
      System.out.println("A circular doubly linked list traversed as:");
      d.add(5);
      d.add(10);
      d.add(15);
      d.display();
   }
}
```

OUTPUT:

```
A circular doubly linked list traversed as:
15<-->10<-->5
```

3.2 PROGRAM 2: Insert element to a linked list either at the beginning, middle or end of the linked list.

3.2.1. Insert at the beginning:

PSEUDOCODE:

```
public void insertatbeg(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else {
         node temp=head.prev;
         temp.next=newnode;
         newnode.next=head;
         newnode.prev=temp;
         head.prev=newnode;
         head=newnode;
      }
   }
```

3.2.2. Insert at the last:

PSEUDOCODE:

```
public void insertatlast(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else
      {
         node temp=head.prev;
         temp.next=newnode;
         newnode.prev=temp;
         newnode.next=head;
         head.prev=newnode;
      }
   }
```

3.2.3. Insert at a specified position:

PSEUDOCODE:

```
public void insertatpos(int x,int pos) {
 node newnode=new node(x);
      if(pos<1) {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1) {
         if(head==null) {
            head=newnode;
            head.prev=head;
            head.next=head;
         }
         else{
            node temp=head.prev;
            temp.next=newnode;
            newnode.next=head;
            newnode.prev=temp;
            head.prev=newnode;
            head=newnode;
         }
      }
      else if(pos>(count()+1)) {
         System.out.println("invalid
position");
      }
      else {
         node temp=head;
         int i;
         for(i=1;i<pos-
1&&i<=count();i++) {
            temp=temp.next;
         }
         if(temp.next!=head) {
            newnode.next=temp.next;
            temp.next.prev=newnode;
            temp.next=newnode;
            newnode.prev=temp;
         }
         else{
            temp.next=newnode;
            newnode.prev=temp;
            newnode.next=head;
            head.prev=newnode;
         } }
   }
```

SOURCECODE OF INSERTION OPERATION:

```
import java.util.Scanner;
class cdllinsert
{
   class node
   {
      int data;
      node next;
      node prev;
      node(int data)
      {
         this.data=data;
         this.next=null;
         this.prev=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else
      {
         node temp=head.prev;
         temp.next=newnode;
         newnode.next=head;
         newnode.prev=temp;
         head.prev=newnode;
         head=newnode;
      }
   }

   public void insertatbeg(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else
      {
         node temp=head.prev;
         temp.next=newnode;
         newnode.next=head;
         newnode.prev=temp;
         head.prev=newnode;
         head=newnode;
      }
   }

   public void insertatlast(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else
      {
         node temp=head.prev;
         temp.next=newnode;
         newnode.prev=temp;
         newnode.next=head;
         head.prev=newnode;
      }
   }



   public void insertatpos(int x,int pos)
   {
      node newnode=new node(x);
      if(pos<1)
      {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            head=newnode;
            head.prev=head;
            head.next=head;
         }
         else
         {
```

```
            node temp=head.prev;
            temp.next=newnode;
            newnode.next=head;
            newnode.prev=temp;
            head.prev=newnode;
            head=newnode;
         }
      }
      else if(pos>(count()+1))
      {
         System.out.println("invalid
position");
      }
      else
      {
         node temp=head;
         int i;
         for(i=1;i<pos-
1&&i<=count();i++)
         {
            temp=temp.next;
         }
         if(temp.next!=head)
         {
            newnode.next=temp.next;
            temp.next.prev=newnode;
            temp.next=newnode;
            newnode.prev=temp;
         }
         else
         {
            temp.next=newnode;
            newnode.prev=temp;
            newnode.next=head;
            head.prev=newnode;
         }
      }
   }

   public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            System.out.print(temp.data+"<--
>");
            temp=temp.next;
         }
         System.out.print(temp.data);

      }
      node temp=head;
   }

   public int count()
   {
      int c=0;
      if(head==null)
      {
         return c;
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            c=c+1;
            temp=temp.next;
         }
         return c+1;
      }
   }

   public static void main(String args[])
   {
      cdllinsert s=new cdllinsert();
      System.out.println("A circular doubly
linked list is given as:");
      s.add(5);
      s.add(10);
      s.add(15);
      s.display();

      int ch;
      do{
      System.out.println();
      System.out.println("press 1 for
insertion at beginning");
      System.out.println("press 2 for
insertion at last");
      System.out.println("press 3 for
insertion at position");
      System.out.println("enter your
choice");
```

```
        Scanner input=new
Scanner(System.in);
        ch=input.nextInt();
        switch(ch)
        {
           case 1:
           {
              s.insertatbeg(9);
              s.display();
              break;

           }
           case 2:
           {
              s.insertatlast(11);
              s.display();
              break;
           }
           case 3:
           {
              System.out.println("Insertion at
pos<1");
              s.insertatpos(1, -1);
              s.display();
              System.out.println();
              System.out.println("Insertion at
pos=1");
              s.insertatpos(1,1);
              s.display();
              System.out.println();
              System.out.println("Insertion at
position 2");
              s.insertatpos(3,2);
              s.display();
              System.out.println();
              System.out.println("Insertion at
position 10");
              s.insertatpos(10,10);
              s.display();
              break;
           }
           default:
           {
              System.out.println("please enter
your choice properly");
           }
        }
        }while(ch<=3);
     }
}
```

OUTPUT:

```
A circular doubly linked list is given as:
15<-->10<-->5

press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
1
9<-->15<-->10<-->5



press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
2
9<-->15<-->10<-->5<-->11
```

```
press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
3
Insertion at pos<1
pos<1 is invalid
9<-->15<-->10<-->5<-->11

Insertion at pos=1
1<-->9<-->15<-->10<-->5<-->11

Insertion at position 2
1<-->3<-->9<-->15<-->10<-->5<-->11

Insertion at position 10
invalid position
1<-->3<-->9<-->15<-->10<-->5<-->11


press 1 for insertion at beginning
press 2 for insertion at last
press 3 for insertion at position
enter your choice
4
please enter your choice properly
```

3.3. PROGRAM 3: Delete element from a linked list either from the beginning, middle or end of the linked list.

3.3.1. Deletion from the beginning:

PSEUDOCODE:

```
public void deleteatbeg()
  {
     if(head==null)
     {
        System.out.println("Empty list");
     }
     else if(head.next==null)
     {
        head=null;
     }
     else
     {
        node temp=head.prev;
        temp.next=head.next;
        head=head.next;
        head.prev=temp;
     }
  }
```

3.3.2. Deletion from the last:

PSEUDOCODE:

```
public void deleteatlast()
  {
     if(head==null)
     {
        System.out.println("Empty list");
     }
     else if(head.next==null)
     {
        head=null;
     }
     else
     {
        node temp=head.prev.prev;
        temp.next=head;
        head.prev=temp;
     }
  }
```

3.3.3. Deletion from the specified position*:*

PSEUDOCODE:

```
public void deleteatpos(int pos)
  {
     if(pos<1)
     {
        System.out.println("pos<1 is
invalid");
     }
     else if(pos==1)
     {
        if(head==null)
        {
           System.out.println("Empty list");
        }
        else if(head.next==null)
        {
           head=null;
        }
        else
        {
           node temp=head.prev;
           temp.next=head.next;
           head=head.next;
           head.prev=temp;
        }
     }

     else
     {
        node temp=head;
        for(int i=1;i<pos-
1&&temp.next!=head;i++)
        {
           temp=temp.next;
        }
        if(temp.next!=head)
        {
           temp.next=temp.next.next;
           temp.prev=temp;
        }
        else
        {
           System.out.println("invalid
position");
        }
     }
  }
```

SOURCECODE FOR DELETION OPERATION:

```
import java.util.Scanner;
class cdlldelete
{
   class node
   {
      int data;
      node next;
      node prev;
      node(int data)
      {
         this.data=data;
         this.next=null;
         this.prev=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else
      {
         node temp=head.prev;
         temp.next=newnode;
         newnode.next=head;
         newnode.prev=temp;
         head.prev=newnode;
         head=newnode;
      }
   }

   public void deleteatbeg()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         node temp=head.prev;
         temp.next=head.next;
         head=head.next;
         head.prev=temp;
      }
   }

   public void deleteatlast()
   {
      if(head==null)
      {
         System.out.println("Empty list");
      }
      else if(head.next==null)
      {
         head=null;
      }
      else
      {
         node temp=head.prev.prev;
         temp.next=head;
         head.prev=temp;
      }
   }



   public void deleteatpos(int pos)
   {
      if(pos<1)
      {
         System.out.println("pos<1 is
invalid");
      }
      else if(pos==1)
      {
         if(head==null)
         {
            System.out.println("Empty list");
         }
         else if(head.next==null)
         {
            head=null;
         }
         else
```

```
        {
            node temp=head.prev;
            temp.next=head.next;
            head=head.next;
            head.prev=temp;
        }
    }

    else
    {
        node temp=head;
        for(int i=1;i<pos-
1&&temp.next!=head;i++)
        {
            temp=temp.next;
        }
        if(temp.next!=head)
        {
            temp.next=temp.next.next;
            temp.prev=temp;
        }
        else
        {
            System.out.println("invalid
position");
        }

    }
  }

  public void display()
  {
    if(head==null)
    {
        System.out.println("empty list");
    }
    else
    {
        node temp=head;
        while(temp.next!=head)
        {
            System.out.print(temp.data+"<--
>");
            temp=temp.next;
        }
        System.out.print(temp.data);

    }
    node temp=head;
  }

  public int count()
  {
    int c=0;
    if(head==null)
    {
        return c;
    }
    else
    {
        node temp=head;
        while(temp.next!=head)
        {
            c=c+1;
            temp=temp.next;
        }
        return c+1;
    }
  }

  public static void main(String args[])
  {
    cdlldelete d=new cdlldelete();
    Scanner input=new
Scanner(System.in);
        d.add(1);
        d.add(2);
        d.add(3);
        d.add(4);
        d.add(5);
        d.add(6);
        System.out.println("This is circular
doubly linked list:");
        d.display();
        int ch;
        do
        {
            System.out.println();
            System.out.println("From where do
you want to delete your newnode?");
            System.out.println("From the
beginning? Press 1");
            System.out.println("From the last?
Press 2");
            System.out.println("From the
specified position? Press 3");
            ch=input.nextInt();
            switch(ch)
            {
                case 1:
```

```
            {
               System.out.println("Deleteing
a node from the beginning.");
               d.deleteatbeg();
               d.display();
               break;
            }
            case 2:
            {
               System.out.println("Delete a
node at last.");
               d.deleteatlast();
               d.display();
               break;
            }
            case 3:
            {
               System.out.println("Deletion
at pos<1");
               d.deleteatpos(-1);
               d.display();
               System.out.println();
               System.out.println("Deletion
at pos=1");
               d.deleteatpos(1);
               d.display();
               System.out.println();
               System.out.println("Deletion
at position 2");
               d.deleteatpos(2);
               d.display();
               System.out.println();
               System.out.println("Deletion
at position 10");
               d.deleteatpos(10);
               d.display();
               break;
            }
            default:
            {
               System.out.println("Enter
your choice from 1 to 3.");
            }
         }
      }while(ch<=3);
   }
}
```

OUTPUT:

```
This is circular doubly linked list:
6<-->5<-->4<-->3<-->2<-->1
From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
1
Deleting a node from the beginning.
5<-->4<-->3<-->2<-->1

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
2
```

```
Delete a node at last.
5<-->4<-->3<-->2
From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
3
Deletion at pos<1
pos<1 is invalid
5<-->4<-->3<-->2

Deletion at pos=1
4<-->3<-->2

Deletion at position 2
4<-->2

Deletion at position 10
invalid position
4<-->2

From where do you want to delete your newnode?
From the beginning? Press 1
From the last? Press 2
From the specified position? Press 3
4
Enter your choice from 1 to 3.
```

3.4. PROGRAM 4: Count the element of the linked list.

PSEUDOCODE:

```
public int count()
  {
     int c=0;
     if(head==null)
     {
        return c;
     }
     else
     {
        node temp=head;
        while(temp.next!=head)
        {
           c=c+1;
           temp=temp.next;
        }
        return c+1;
     }
  }
```

SOURCECODE:

```
class cdllcount
{
   class node
   {
      int data;
      node next;
      node prev;
      node(int data)
      {
         this.data=data;
         this.next=null;
         this.prev=null;
;     }
   }
   public node head=null;

   public void add(int x)
   {
      node newnode=new node(x);
      if(head==null)
      {
         head=newnode;
         head.prev=head;
         head.next=head;
      }
      else
      {
         node temp=head.prev;
         temp.next=newnode;
         newnode.next=head;
         newnode.prev=temp;
         head.prev=newnode;
         head=newnode;
      }
   }

   public void display()
   {
      if(head==null)
      {
         System.out.println("empty list");
      }
      else
      {
         node temp=head;
         while(temp.next!=head)
         {
            System.out.print(temp.data+"<--
>");
            temp=temp.next;
         }
         System.out.print(temp.data);

      }
      node temp=head;
   }

   public int count()
   {
      int c=0;
      if(head==null)
      {
         return c;
      }
```

```
    else
    {
       node temp=head;
       while(temp.next!=head)
       {
          c=c+1;
          temp=temp.next;
       }
       return c+1;
    }
  }

  public static void main(String args[])
  {
     cdllcount s=new cdllcount();
     System.out.println("A circular doubly
linked list is given as:");
     s.add(5);
     s.add(10);
     s.add(15);
     s.display();
     System.out.println();
     System.out.println("The number of
nodes in the linked list are:"+s.count());
  }
}
```

OUTPUT:

```
A circular doubly linked list is given as:
15<-->10<-->5
The number of nodes in the linked list are:3
```

4. OUTPUT AND DISCUSSION:

- ✓ In this experiment, the output of program 1 shows the traverse of a node to display the data of the node in the circular doubly linked list.
- ✓ In the output of program 2, it can be seen that the insertion of a node is done at the beginning of the linked list, after then, when the user press 2, a node is inserted at the last and then when the user press option 3, a node is inserted at a specified position.
- ✓ The output of the Program 3 shows the deletion of a node from the beginning, from the last and from the specified position.
- ✓ In program 4, the number of nodes are counted in the given circular doubly linked list..

5. CONCLUSION:

A Java program to traverse, insert a node at first, last, certain position, delete a node at beginning, at last and at certain position and counting of the nodes in circular doubly linked list is successfully run.