Experiment -1

Write a java program to perform various operations in Singly Linked List.

AIM:

Java program to perform various operations in Singly Linked List.

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a structure of name node and declare structure variables.
Step 4: Create a class name linked list and declare required functions.
Step 5: Initialize start and end to null inside constructor.
Step 6: Create an object for class Singly Linked List.
Step 7: Display all operations on linked list using switch case and choose on operation from the user and call required function.

Step 8: Create node function by creating 2 variables link, data. Assign link as NULL and data as "Zero" and return link and data to display the values of it.

Step 9: Define insertAtStart function by reading the elements to be inserted from the user and create 2 variables start and end and assign the values (nptr) to start, if start is equal to null else assign it to next of start and adjust pointers.

Step 10: Define insertAtEnd function by reading the elements to be Inserted at last from the user and create 2 variables and create new object and traverse the list until null and assign the value to the end.

Step 11: Define insertAtPos function by reading element to be inserted and its position from the user and create 3 variables and create new object and move the pointer to the required position and insert the node at that position.

Step 12: Define delete function by reading position of the value to be deleted and create two variables and read the position and delete the element at the position.

Step 13: Define check Empty function to check whether the given list is empty or not.

Step 14: Define getSize function and create size variable and return the size of the list.

Step 15: Define display funtion by creating a variable and point to the start and traverse it using while loop until it points to NULL and print value at each iteration.

Step 16: Stop

SOURCE CODE:

```
import java.util.Scanner;
 class Node{
    protected int data;
protected Node link;
     public Node()
    {
        link = null;
data = 0;
    }
       public Node(int d,Node n)
    {
        data = d;
        link = n;
    }
 public void setLink(Node n)
    {
         link = n;
    }
    public void setData(int d)
    {
        data = d;
    }
      public Node getLink()
    {
        return link;
    }
    public int getData()
    {
        return data;
    }
}
class linkedList
{
    protected Node start;
    protected Node end ;
    public int size ;
  public linkedList()
   {
        start = null;
        end = null;
        size = 0;
    }
    public boolean isEmpty()
    {
        return start == null;
    }
    public int getSize()
```

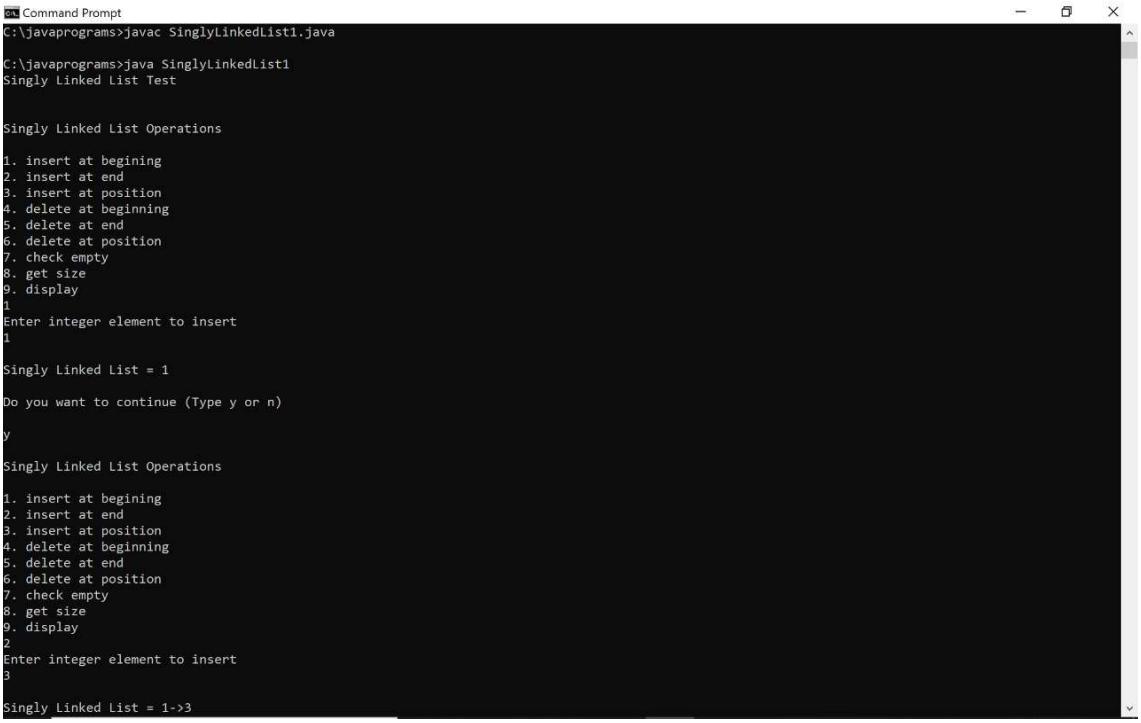
```
{
   return size;
}
public void insertAtStart(int val)
{
   Node nptr = new Node(val, null);
   size++ ;
   if(start == null)
   {
      start = nptr;
      end = start;
   }
   else
   {
      nptr.setLink(start);
      start = nptr;
   }
}
public void insertAtEnd(int val)
{
  Node nptr = new Node(val,null);
   size++ ;
   if(start == null)
   {
      start = nptr;
      end = start;
   }
   else
   {
      end.setLink(nptr);
      end = nptr;
   }
}
public void insertAtPos(int val , int pos)
{
   Node nptr = new Node(val, null);
   Node ptr = start;
   pos = pos - 1 ;
   for (int i = 1; i < size; i++)
   {
      if (i == pos)
      {
         Node tmp = ptr.getLink() ;
         ptr.setLink(nptr);
         nptr.setLink(tmp);
         break;
      }
      ptr = ptr.getLink();
   }
   size++ ;
```

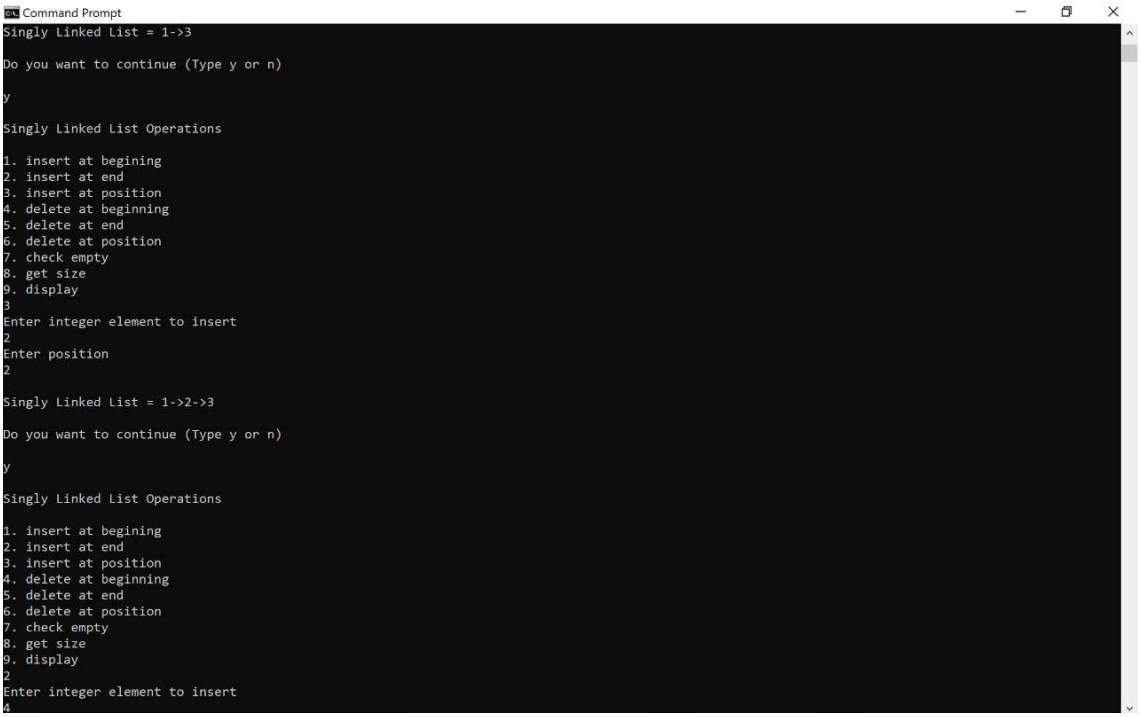
```
}
public void deleteAtPos(int pos)
{
   if (pos == 1)
   {
      start = start.getLink();
      size--;
      return ;
   }
   if (pos == size)
   {
      Node s = start;
      Node t = start;
      while (s != end)
      {
         t = s;
         s = s.getLink();
      }
      end = t;
      end.setLink(null);
      size --;
      return;
   }
   Node ptr = start;
   pos = pos - 1 ;
   for (int i = 1; i < size - 1; i++)
   {
      if (i == pos)
      {
         Node tmp = ptr.getLink();
         tmp = tmp.getLink();
         ptr.setLink(tmp);
         break;
      }
      ptr = ptr.getLink();
   }
   size-- ;
}
public void display()
{
   System.out.print("\nSingly Linked List = ");
   if (size == 0)
   {
      System.out.print("empty\n");
      return;
   }
   if (start.getLink() == null)
   {
      System.out.println(start.getData() );
      return;
```

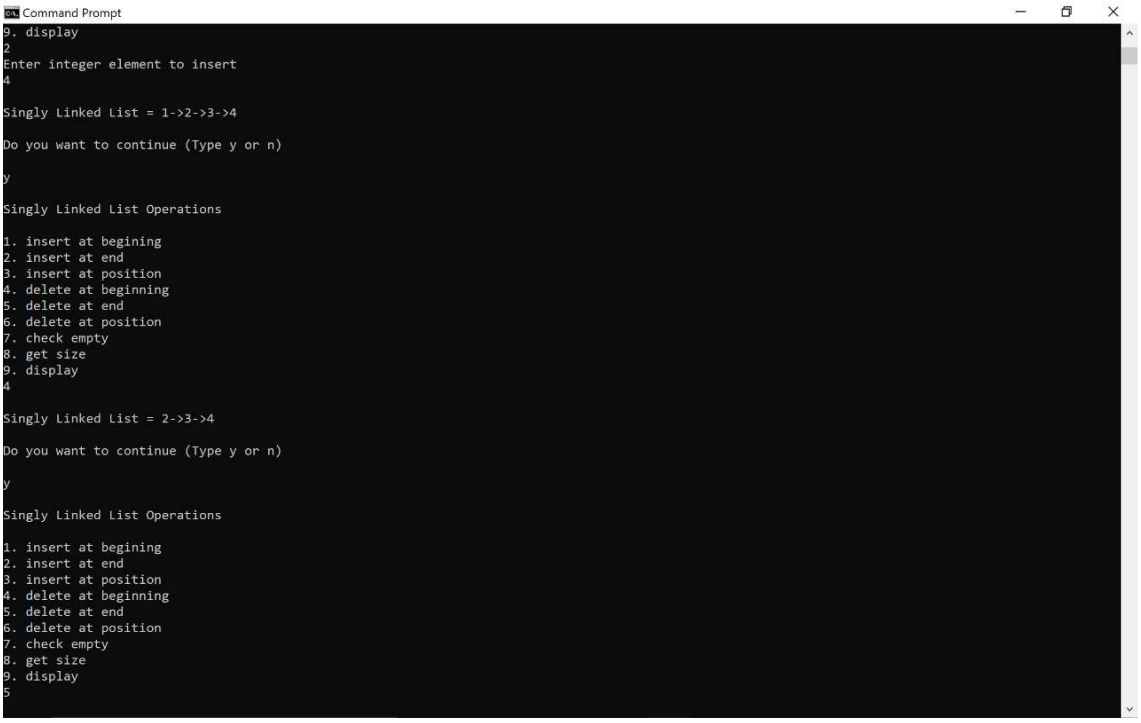
```
        }
        Node ptr = start;
        System.out.print(start.getData()+ "->");
        ptr = start.getLink();
        while (ptr.getLink() != null)
        {
            System.out.print(ptr.getData()+ "->");
            ptr = ptr.getLink();
        }
        System.out.print(ptr.getData()+ "\n");
    }
}
public class SinglyLinkedList1
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        linkedList list = new linkedList();
        System.out.println("Singly Linked List Test\n");
        char ch;
        do
        {
            System.out.println("\nSingly Linked List Operations\n");
            System.out.println("1. insert at begining");
            System.out.println("2. insert at end");
            System.out.println("3. insert at position");
            System.out.println("4. delete at beginning");
            System.out.println("5. delete at end");
            System.out.println("6. delete at position");
            System.out.println("7. check empty");
            System.out.println("8. get size");
            System.out.println("9. display");
            int choice = scan.nextInt();
            switch (choice)  {
            case 1 :
                System.out.println("Enter integer element to insert");
                list.insertAtStart( scan.nextInt() );
                break;
            case 2 :
                System.out.println("Enter integer element to insert");
                list.insertAtEnd( scan.nextInt() );
                break;
            case 3 :
                System.out.println("Enter integer element to insert");
                int num = scan.nextInt() ;
                System.out.println("Enter position");
                int pos = scan.nextInt() ;
                if (pos <= 1 || pos > list.getSize() )
                    System.out.println("Invalid position\n");
                else
```

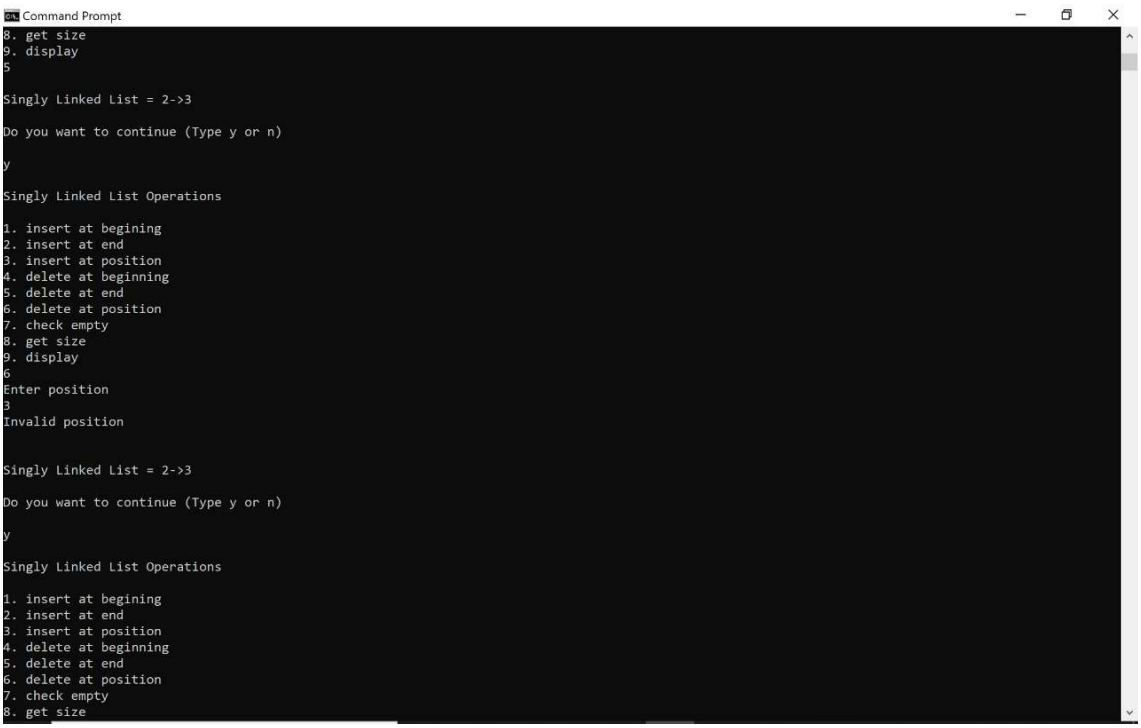
```
                list.insertAtPos(num, pos);
                break;
            case 4 :
                list.deleteAtPos(1);
                break;
            case 5 :
                list.deleteAtPos(list.getSize());
                break;
            case 6 :
                System.out.println("Enter position");
                int p = scan.nextInt() ;
                if (p < 1 || p > list.getSize() )
                    System.out.println("Invalid position\n");
                else
                    list.deleteAtPos(p);
                break;
            case 7 :
                System.out.println("Empty status = "+ list.isEmpty());
                break;
            case 8 :
                System.out.println("Size = "+ list.getSize() +" \n");
                break;
            case 9:  break;
             default :
                System.out.println("Wrong Entry \n ");
                break;
            }
            list.display();
            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
}
```

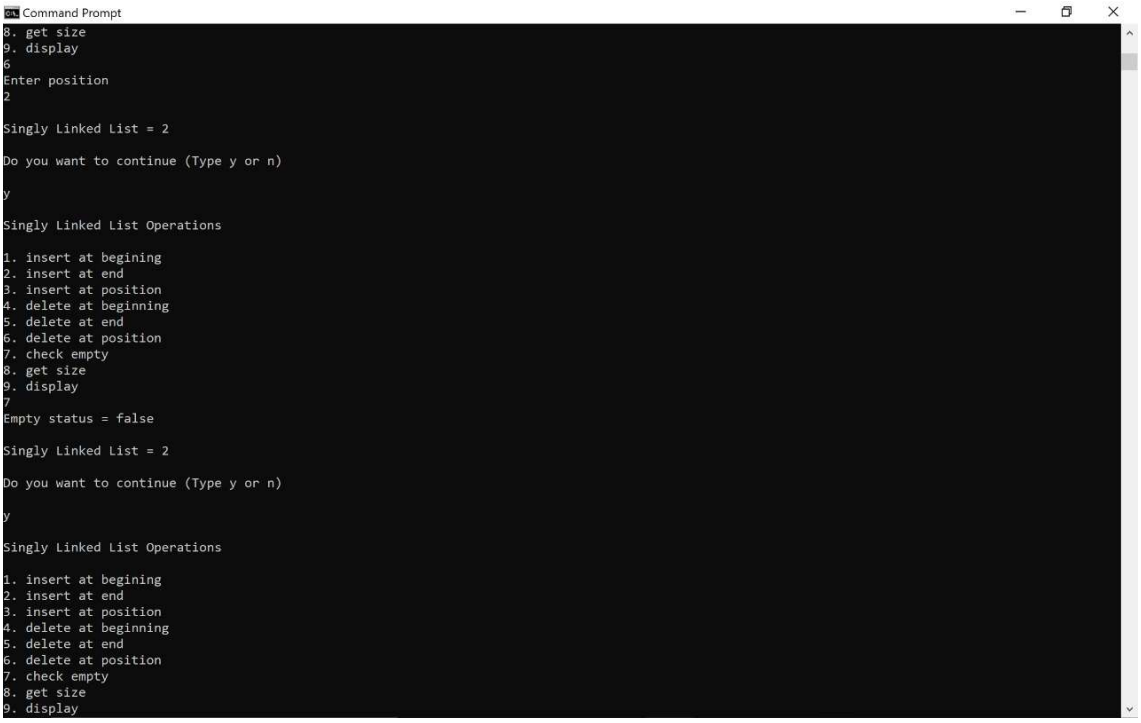
ACTUAL OUTPUT:






Command Prompt
9. display
2
Enter integer element to insert
4
Singly Linked List = 1->2->3->4
Do you want to continue (Type y or n)
y
Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
4
Singly Linked List = 2->3->4
Do you want to continue (Type y or n)
y
Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
5


Command Prompt
8. get size
9. display
5
Singly Linked List = 2->3
Do you want to continue (Type y or n)
y
Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
6
Enter position
3
Invalid position
Singly Linked List = 2->3
Do you want to continue (Type y or n)
y
Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size


Command Prompt
8. get size
9. display
6
Enter position
2
Singly Linked List = 2
Do you want to continue (Type y or n)
y
Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
7
Empty status = false
Singly Linked List = 2
Do you want to continue (Type y or n)
y
Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display


Command Prompt
. insert at position
. delete at beginning
. delete at end
. delete at position
. check empty
. get size
. display
ize = 1
ingly Linked List = 2
o you want to continue (Type y or n)
ingly Linked List Operations
. insert at begining
. insert at end
. insert at position
. delete at beginning
. delete at end
. delete at position
. check empty
. get size
. display
ingly Linked List = 2
o you want to continue (Type y or n)
:\javaprograms>

Experiment -2

Write a java program for the following.

a)Reverse a linkedList

AIM:

Java program to reverse a linked list.

ALGORITHM:

Step 1: Start

Step 2: Declare required header files.

Step 3: Create a class name ReverseList and declare required functions.

Step 4: Create a structure of name Node and declare structure variables (data, link) and initialize data and with data using this pointer and link with null inside the constructor node.

Step 5: And initialize the head 2 tail to null inside the Node Constructor

Step 6: Create a new node using add Node() and check whether the list is empty or not.

Step 7: If the list is empty, assign head and tail to the new node. or else new node will be added after tail such that tail's next will point to new node. Then new Node will become new tail of the list.

Step 8: Define reverse function and assign previous to null and current to head and next to null.And if the current is not equal to null, assign next = current & current to prev & prev to current to reverse the linked link and assign current to next. Next make head as prev.

Step9: Define display function by creating a variable and point to the start and traverse if using while loop until it points to NULL and print value of reversed linked loop at each iteration.

Step10: Stop

EXPECTED OUTPUT:

```
D:\mtech\adsalab>javac ReverseList.java

D:\mtech\adsalab>java ReverseList
Enter size of single linked list
5
Enter single linked list values
2 1 4 8 3
Original list:
2-->1-->4-->8-->3
Reversed list:
3-->8-->4-->1-->2
```

SOURCE CODE:

```
import java.util.Scanner;
public class ReverseList
{
class Node
{
int data;
Node link;
}
public Node(int data)
{
this.data = data;
this.link = null;
}
}
public Node head = null;
public Node tail = null;
public void create(int data)
{
Node newNode = new Node(data);
if(head == null)
{
head = newNode;
tail = newNode;
}
Else
{
tail.link = newNode;
tail = newNode;
}
}
public void reverse()
{
Node prev = null;
Node current = head;
Node next = null;
while (current != null)
{
next = current.link;
current.link = prev;
prev = current;
current = next;
}
head = prev;
}
public void display()
{
Node current = head;
if(head == null)
{
```

```
        System.out.println("List is empty");
return;
}
while(current.link != null)
{
System.out.print(current.data + "-->");
current = current.link;
}
System.out.print(current.data);
System.out.println();
}
public static void main(String[] args)
{
ReverseList sList = new ReverseList();
Scanner scan = new Scanner(System.in);
int size;
System.out.println("Enter size of single linked list");
size=scan.nextInt();
System.out.println("Enter single linked list values");
for(int i=0;i<size;i++)
{
sList.create( scan.nextInt() );
}
System.out.println("Original list: ");
sList.display();
sList.reverse();
System.out.println("Reversed list: ");
sList.display();
}
}
```

ACTUAL OUTPUT:

```
D:\mtech\adsalab>javac ReverseList.java

D:\mtech\adsalab>java ReverseList
Enter size of single linked list
5
Enter single linked list values
2 1 4 8 3
Original list:
2-->1-->4-->8-->3
Reversed list:
3-->8-->4-->1-->2
```

2b) Sort the data in the linked list

AIM:

program to sort the data in the linked list.

ALGORITHM:

Step 1: Start
Step 2: Declare required header files.
Step 3: Create a class name Sort List and declare required functions.
Step 4: Create a structure of name node and declare structure variables.
Step 5: Initialize data with data using this pointer and link with null inside the constructor node.
Step 6: Create a new node using addNode function and check whether the list is empty or not.
Step7: If the list is emply, assign head and tail to the new node or else new node will be added after tail such that fail's next will point to new Node. Then new node will become new tail of the list.
Step8: Define sort List function and consider the size of the elements in the linked list.
Step 9: Considering the index node and the current node, if the current node data is greater than the Index's node data, swap the data between them, using while loop traverse it until the last element.
Step 10: Define display function by creating a variable and point to the start and traverse it using while loop until it point to NULL and print the values of sorted data in each iteration.
Step11: Stop.

EXPECTED OUTPUT:

```
D:\mtech\adsalab>javac SortList.java

D:\mtech\adsalab>java SortList
Enter size of single linked list
5
Enter single linked list values
23 17 5 9 40
Original list:
23-->17-->5-->9-->40
Sorted list:
5-->9-->17-->23-->40
```

SOURCE CODE:

```
import java.util.Scanner;
 public class SortList {
   class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   public Node head = null;
   public Node tail = null;
     public void create(int data) {
      Node newNode = new Node(data);
      if(head == null) {
         head = newNode;
         tail = newNode;
      }
      else {
         tail.next = newNode;
         tail = newNode;
      }
   }
   public void sortList() {
      Node current = head, index = null;
  int temp;
      if(head == null) {
         return;
      }
      else {
         while(current != null) {
              index = current.next;
               while(index != null) {
                 if(current.data > index.data) {
                     temp = current.data;
                     current.data = index.data;
                     index.data = temp;
                 }
                 index = index.next;
             }
             current = current.next;
         }
      }
   }

   public void display() {
      Node current = head;
      if(head == null) {
```

```
            System.out.println("List is empty");
            return;
        }
        while(current.next != null)
            System.out.print(current.data + "-->");
            current = current.next;
        }
                System.out.print(current.data);
        System.out.println();
    }

    public static void main(String[] args) {

        SortList sList = new SortList();
        Scanner scan = new Scanner(System.in);
            int size;
            System.out.println("Enter size of single linked list");
        size=scan.nextInt();
            System.out.println("Enter single linked list values");
            for(int i=0;i<size;i++)
            {
                sList.create( scan.nextInt() );
            }
        System.out.println("Original list: ");
        sList.display();
    sList.sortList();
        System.out.println("Sorted list: ");
        sList.display();
    }
}
```

ACTUAL OUTPUT:

```
D:\mtech\adsalab>javac SortList.java

D:\mtech\adsalab>java SortList
Enter size of single linked list
5
Enter single linked list values
23 17 5 9 40
Original list:
23-->17-->5-->9-->40
Sorted list:
5-->9-->17-->23-->40
```

2c) Removing duplicates

AIM:

Java program to Remove the duplicates in Singly Linked List

ALGORITHM:

Step 1: Start
Step 2: Declare required header files.
Step 3: Create a class name Remove Duplicate and declare required functions.
Step 4: Create a structure of name node and declare structure variables.
Step 5 : Initialize data with data using this pointer and link with null inside the constructor node.
Step 6: Create a new node using addNode function and check whether the list is empty or not.
Step 7: If the list is empty assign head and tail to the new node or else new node will be added after tail such that tail's next will point to new node. Then new node will become tail of the list.
Step 8: Define removeDuplicate function and initialize current node as head and index as null, and traverse it using while loop until the previous node (temp) is equal to the current to (temp) node. If it is equal to the current node remove the node and print the values until the Next becomes null.
Step 9: Define display function by creating a variable and point to the start and traverse it using while loop, until it points to NULL and print the values, after removing the duplicates in each iteration.
Step10: Stop.

EXPECTED OUTPUT:

```
D:\mtech\adsalab>javac RemoveDuplicate.java

D:\mtech\adsalab>java RemoveDuplicate
Enter size of single linked list
6
Enter single linked list values
2 4 1 3 2 1
Originals list:
2--> 4--> 1--> 3--> 2--> 1
List after removing duplicates:
2--> 4--> 1--> 3
```

SOURCE CODE:

```
import java.util.Scanner;
public class RemoveDuplicate {
   class Node{
      int data;
      Node next;
      public Node(int data) {
         this.data = data;
         this.next = null;
      }
   }
   public Node head = null;
   public Node tail = null
   public void create(int data) {
      Node newNode = new Node(data);
      if(head == null) {
         head = newNode;
         tail = newNode;
      }
      else {
         tail.next = newNode;
         tail = newNode;
      }
   }
   public void removeDuplicate()
{
      Node current = head, index = null, temp = null;
      if(head == null) {
         return;
  }
      else
{
   while(current != null){
            temp = current
            index = current.next;
            while(index != null) {
               if(current.data == index.data) {
                  temp.next = index.next;
               }
               else {
                  temp = index;
               }
               index = index.next;
            }
            current = current.next;
         }
      }
   }
   public void display() {
      Node current = head;
```

```
        if(head == null) {
            System.out.println("List is empty");
            return;
        }
        while(current.next != null)
            System.out.print(current.data + "--> ");
            current = current.next;
        }
            System.out.print(current.data);
        System.out.println();
    }
    public static void main(String[] args)
{
        RemoveDuplicate sList = new RemoveDuplicate();
            Scanner scan = new Scanner(System.in);
            int size;
            System.out.println("Enter size of single linked list");
        size=scan.nextInt();
            System.out.println("Enter single linked list values");
            for(int i=0;i<size;i++)
            {
                sList.create( scan.nextInt() );
            }
        System.out.println("Originals list: ");
        sList.display();
        sList.removeDuplicate();
        System.out.println("List after removing duplicates: ");
        sList.display();
    }
}
```

ACTUAL OUTPUT:

```
D:\mtech\adsalab>javac RemoveDuplicate.java

D:\mtech\adsalab>java RemoveDuplicate
Enter size of single linked list
6
Enter single linked list values
2 4 1 3 2 1
Originals list:
2--> 4--> 1--> 3--> 2--> 1
List after removing duplicates:
2--> 4--> 1--> 3
```

2d) Merge two LinkedList

AIM:

Java program to merge two linked lists in single linked list

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Define class MergeList and declare required functions.
Step 4: Create a structure of name node and declare structure variables.
Step 5: Initialize temp data as key and next data with null.
Step 6: Create a function print List to print the linked list before merging the list.
Step 7: Create a function Merge and initially if the first list is empty display the second list. If the second and list is empty display the first list. If the data is present in the both lists, merge the second list by considering the next points from the first list.
Step 8: Initialize the values to list 1 and list 2 and by calling the merge function in the main print the merged list.
Step 9: Stop.

EXPECTED OUTPUT:

```
D:\mtech\adsalab>javac MergeList.java

D:\mtech\adsalab>java MergeList
List 1
1 3 5
List 2
0 2 4
Merged List
0 1 2 3 4 5
```

SOURCE CODE:

```
class MergeList{
   static class Node {
      int data;
      Node next;
   };
   static Node newNode(int key)
   {
      Node temp = new Node();
      temp.data = key;
      temp.next = null;
      return temp;
   }
   static void printList(Node node)
   {
      while (node != null) {
         System.out.printf("%d ", node.data);
         node = node.next;
```

```
    }
  }
  static Node merge(Node h1, Node h2)
  {
    if (h1 == null)
      return h2;
    if (h2 == null)
      return h1;
    if (h1.data < h2.data) {
      h1.next = merge(h1.next, h2);
      return h1;
    }
    else {
      h2.next = merge(h1, h2.next);
      return h2;
    }
  }
  public static void main(String args[])
{
    Node head1 = newNode(1);
    head1.next = newNode(3);
    head1.next.next = newNode(5);
    Node head2 = newNode(0);
    head2.next = newNode(2);
    head2.next.next = newNode(4);
      System.out.println("List 1");
      printList(head1);
      System.out.println("\nList 2");
      printList(head2);
    Node mergedhead = merge(head1, head2);
      System.out.println("\nMerged List");
    printList(mergedhead);
  }
}
```

ACTUAL OUTPUT:

```
D:\mtech\adsalab>javac MergeList.java

D:\mtech\adsalab>java MergeList
List 1
1 3 5
List 2
0 2 4
Merged List
0 1 2 3 4 5
```

Experiment -3

Write a java program to perform various operations in doubly Linked List

AIM:

To perform various operations on doubly linked list using java program.

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a class named dlinked list and create a node named as head and tail and Initialize the variables and pointer in it. Create a construct node and initialize the values for it.
Step 4: Create a function insertBeg and create an object and initialize head and tail to the same object if the head is equal to null else assign it to the next of start and adjust pointers.
Step 5: Define function insertEnd and create an object and by reading the elements to be inserted at end from the user and create a variable temp inserted values user and traverse the list until the null and assign the value to the end.
Step 6: Define function insertPos by reading element to be inserted and its position from the user and create variable and create a object and move the pointer to the required position and insert the node at that pointer.
Step 7: Define function deteleBeg and by Considering the list. if the list is empty throw an exception and if the list is not empty assign head pointer to it's next.
Step 8: Define function deleteEnd and the list entered by the by the user it empty display an empty message if not traverse it until the end and delete the value at end.
Step 9: Define function deletePos and the list entered by the user is empty display an empty message if not traverse until the position entered by the user and delete the value.
Step 10: Define display function by creating a variable and point it to the start and traverse it using while loop until it points to the null and print value at each iteration.
Step 11: Stop.

SOURCE CODE:

```
import java.lang.*;
import java.io.*;
import java.util.Scanner;
class dlinkedlist
{
        node head;
        node tail;
        class node
        {
        int data;
        node next;
        node prev;
        node(int val)
        {
        data=val;
        next=null;
        prev=null;
        }
        }
```

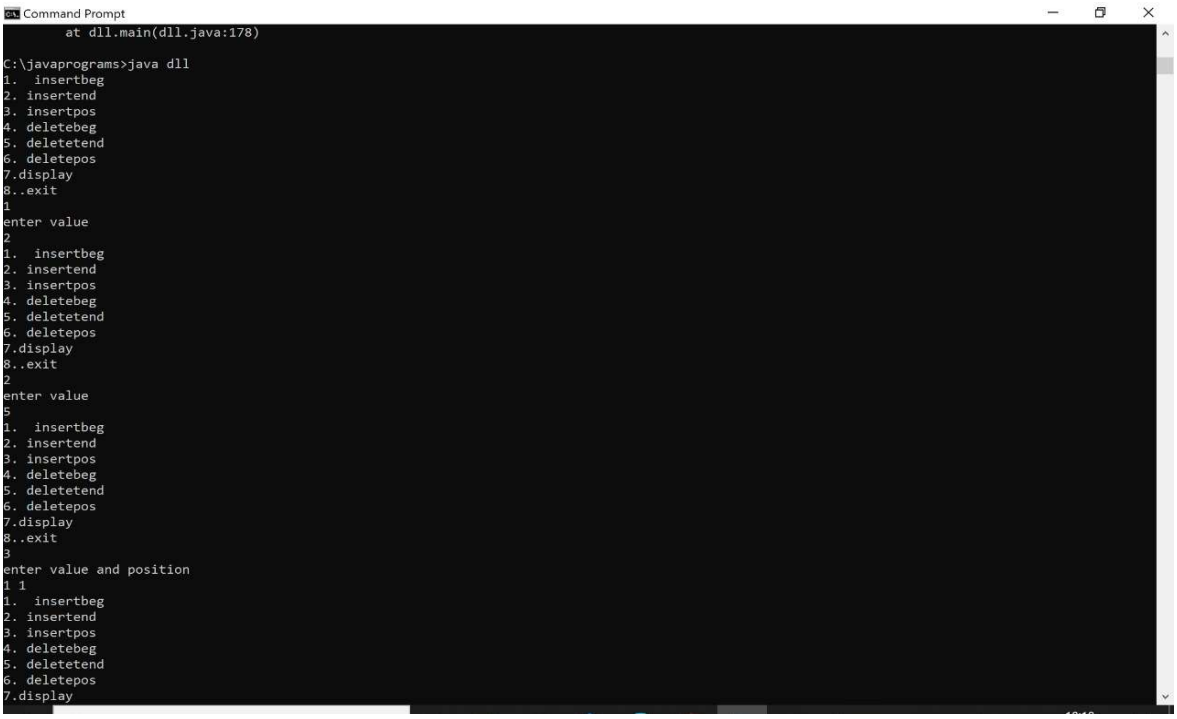
```
public void insertbeg(int x)
{
node p= new node(x);
if( head==null)
{
head=p;
tail=p;
}
else
{
      p.next=head;
       head.prev=p;
      head=p;
}
return;
}
public void insertend(int x)
{
              node p=new node(x);
            node temp;
            if( head==null)
{
            head=p;
            tail=p;
}
else
{
            tail.next=p;
            p.prev=tail;
            tail=p;
}
}
public void insertpos(int x, int pos)
{
            int count=1;
            node temp=head;
            node p= new node(x);
            if(head==null)
{
            head=p;
            tail=p;
}
else
{
            if(pos==1)
            insertbeg(x);
            else
{
while(count<pos-1 && temp.next!=null){
            temp=temp.next;
```

```
count=count+1;
}
            p.next=temp.next;
temp.next.prev=p;
            p.prev=temp;
temp.next=p;
}
}
}
public void display(node t){
node temp=t;
            System.out.print("Start->");
while(temp!=null){
System.out.print(temp.data+"->");
            temp=temp.next;
}
System.out.print("null\n");
}
public int deletebeg(){
int x1=0;
if(head==null){
System.out.println("linked list is empty\n");
}
else{
            x1=head.data;
      head=head.next;
            if(head!=null)
    head.prev=null;
}
return x1;
}
public int deleteend(){
int x1=0;
node temp=null;
node temp1=null;
if(head==null)
System.out.println("linked list is empty");
else{
temp=tail.prev;
x1=tail.data;
temp.next=null;
tail=temp;
}
return x1;
}
public int deletepos(int pos){
int count=1;
node temp=head;
int x1=0;
if(head==null)
```

```
System.out.println("linked list is empty\n");
else{
if(pos==1)
deletebeg();
else{
while(count<pos-1 && temp.next!=null){
temp=temp.next;
count=count+1;
}
x1=temp.next.data;
temp.next=temp.next.next;
if(temp.next!=null)
temp.next.prev=temp;
}
}
return x1;
}
}
class dll{
public static void main(String[] args) {
dlinkedlist l=new dlinkedlist();
int c, x=0;
char ch;
Scanner sc=new Scanner(System.in);
while(true){
System.out.println("1.  insertbeg");
System.out.println("2. insertend");
System.out.println("3. insertpos");
System.out.println("4. deletebeg");
System.out.println("5. deletetend");
System.out.println("6. deletepos");
System.out.println("7.display");
System.out.println("8..exit");
c=sc.nextInt();
switch(c){
case 1:System.out.println("enter value");
          l.insertbeg(sc.nextInt());
                break;
case 2: System.out.println("enter value");
          l.insertend(sc.nextInt());
                 break;
case 3: System.out.println("enter value and position");
                l.insertpos(sc.nextInt(), sc.nextInt());
                 break;
case 4: x=l.deletebeg();
          System.out.println("deleted element is"+x);
          break;
case 5: x=l.deleteend();
           System.out.println("deleted element is"+x);
            break;
```

```
case 6:     System.out.println("enter the position");
             x=l.deletepos(sc.nextInt());
            System.out.println("deleted element is"+x);
               break;
case 7: l.display(l.head);
                  break;
case 8: System.exit(0);
}
}
}
}
```

ACTUAL OUTPUT:




Command Prompt
7.display
8..exit
3
enter value and position
42
2
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos
7.display
8..exit
4
deleted element is1
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos
7.display
8..exit
7
Start->42->2->5->null
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos
7.display
8..exit
5
deleted element is5
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos
7.display


Command Prompt
7.display
8..exit
7
Start->42->2->null
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos
7.display
8..exit
1
enter value
1
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos
7.display
8..exit
6
enter the position
2
deleted element is42
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos
7.display
8..exit
7
Start->1->2->null
1. insertbeg
2. insertend
3. insertpos
4. deletebeg
5. deletetend
6. deletepos

Experiment -4

Write for program to perform various operations on Circular Linked List

AIM:

Java program to perform various operations on circular Linked list

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a structure of name Node and declare structure Variables.
Step 4: Create a class name linked list and declare required functions.
Step 5: Initialize start and end to null inside constructor.
Step 6: Create an object for class CircularSinglyLinked List.
Step 7: Display all operations on linked list using switch case and choose an operation from the user and call required function.
Step 8: Create node function by creating 2 variable link,data. Assign link as null and data as Zero and return link and data to display the values of it.
Step 9: Define insertAtStart function by reading the elements to be inserted from the user and

create 2 variables start and end and assign the values (nptr) to start and assign the start value to the end, if start is equal to null else assign to next of start and adjust pointers.

Step 10: Define insertAtEnd function by reading the elements to be inserted at last from the user and create 2 variables and Create new object and traverse the list until null, and assign the value to the end and assign start value to the end.

Step 11: Define insertAtPos function by reading element to be inserted and its position from the user and create 3 variables and create new object and move the pointer to the required position and insert the node at that position and assign start to end.

Step 12: Define deleteAtBeg function and if the size of the list is one make start and end as null decrease the size or else consider the list and assign it to start and end and decrease the size.

Step 13: Define deleteAtEnd function and create temp(t) &s variable for Structure node and assign start &s to t and end to t if s is not equal to end and then assign t to end.

Step 14: Define deleteAtPos function and if size is equal to 1, assign start and end as null else if pos is equal to 1,assign the values to start and end and decrease the size else if pos is equal to size, use deleteAtEnd function.

Step 15: Define checkEmpty function to check whether the given list is empty or not.

Step 16: Define getSize function and create size variable and return the size of the list.

Step 17: Define display function by creating a variable and point to the start and traverse it using while loop until it points to NULL and print value at each iteration.

Step 18: Stop.

SOURCE CODE:

```
 import java.util.Scanner;
class Node
{
   protected int data;
   protected Node link;
   public Node()
   {
      link = null;
      data = 0;
   }
   public Node(int d,Node n)
   {
      data = d;
      link = n;
   }
   public void setLink(Node n)
   {
      link = n;
   }
   public void setData(int d)
   {
```

```
        data = d;
    }
    public Node getLink()
    {
        return link;
    }
    public int getData()
    {
        return data;
    }
}
class linkedList
{
    protected Node start ;
    protected Node end ;
    public int size ;
    public linkedList()
    {
    start = null;
        end = null;
        size = 0;
    }
    public boolean isEmpty()
    {
        return start == null;
    }
    public int getSize()
    {
        return size;
    }
    public void insertAtStart(int val)
    {
        Node nptr = new Node(val,null);
        nptr.setLink(start);
        if(start == null)
        {
          start = nptr;
            nptr.setLink(start);
            end = start;
        }
        else
        {
            end.setLink(nptr);
            start = nptr;
        }
        size++ ;
    }
    public void insertAtEnd(int val)
    {
        Node nptr = new Node(val,null);
```

```
    nptr.setLink(start);
    if(start == null)
    {
        start = nptr;
        nptr.setLink(start);
        end = start;
    }
    else
    {
        end.setLink(nptr);
        end = nptr;
    }
    size++ ;
}
public void insertAtPos(int val , int pos)
{
    Node nptr = new Node(val,null);
    Node ptr = start;
    pos = pos - 1 ;
    for (int i = 1; i < size - 1; i++)
    {
        if (i == pos)
        {
            Node tmp = ptr.getLink() ;
            ptr.setLink( nptr );
            nptr.setLink(tmp);
            break;
        }
        ptr = ptr.getLink();
    }
    size++ ;
}
public void deleteAtPos(int pos)
{
    if (size == 1 && pos == 1)
    {
        start = null;
        end = null;
        size = 0;
        return ;
    }
    if (pos == 1)
    {
        start = start.getLink();
        end.setLink(start);
        size--;
        return ;
    }
    if (pos == size)
    {
```

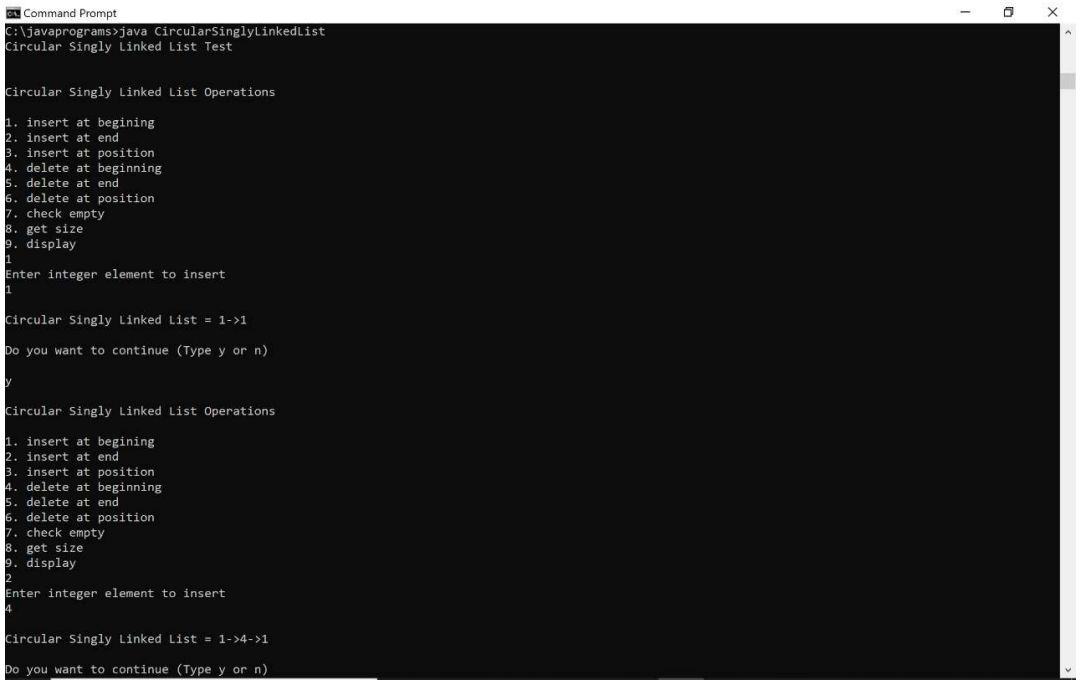
```
        Node s = start;
        Node t = start;
        while (s != end)
        {
            t = s;
            s = s.getLink();
        }
        end = t;
        end.setLink(start);
        size --;
        return;
    }
    Node ptr = start;
    pos = pos - 1 ;
    for (int i = 1; i < size - 1; i++)
    {
        if (i == pos)
        {
            Node tmp = ptr.getLink();
            tmp = tmp.getLink();
            ptr.setLink(tmp);
            break;
        }
        ptr = ptr.getLink();
    }
    size-- ;
}
 public void deleteAtBeg()
{
    if (size == 1)
    {
        start = null;
        end = null;
        size = 0;
        return ;
    }
    else
    {
        start = start.getLink();
        end.setLink(start);
        size--;
        return ;
    }
}
public void deleteAtEnd()
{
            Node s = start;
        Node t = start;
        while (s != end)
        {
```

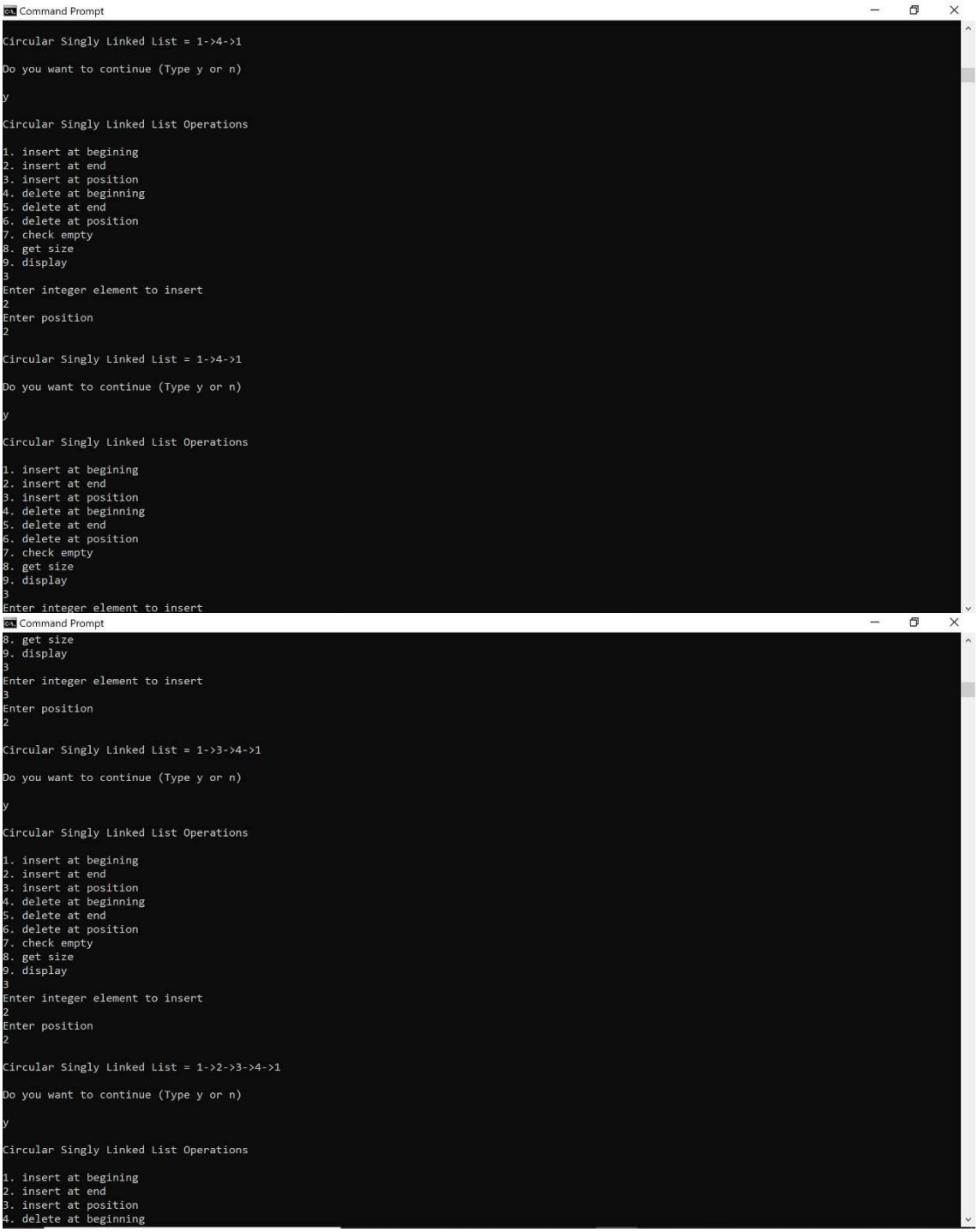
```
            t = s;
            s = s.getLink();
        }
        end = t;
        end.setLink(start);
        size --;
        return;
    }
    public void display()
    {
        System.out.print("\nCircular Singly Linked List = ");
        Node ptr = start;
        if (size == 0)
        {
            System.out.print("empty\n");
            return;
        }
        if (start.getLink() == start){
            System.out.print(start.getData()+ "->"+ptr.getData()+ "\n");
            return;
        }
        System.out.print(start.getData()+ "->");
        ptr = start.getLink();
        while (ptr.getLink() != start) {
            System.out.print(ptr.getData()+ "->");
            ptr = ptr.getLink();
        }
        System.out.print(ptr.getData()+ "->");
        ptr = ptr.getLink();
        System.out.print(ptr.getData()+ "\n");
    }
}
public class CircularSinglyLinkedList
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        linkedList list = new linkedList();
        System.out.println("Circular Singly Linked List Test\n");
        char ch;
        do
        {
            System.out.println("\nCircular Singly Linked List Operations\n");
            System.out.println("1. insert at begining");
            System.out.println("2. insert at end");
            System.out.println("3. insert at position");
            System.out.println("4. delete at beginning");
            System.out.println("5. delete at end");
            System.out.println("6. delete at position");
            System.out.println("7. check empty");
```

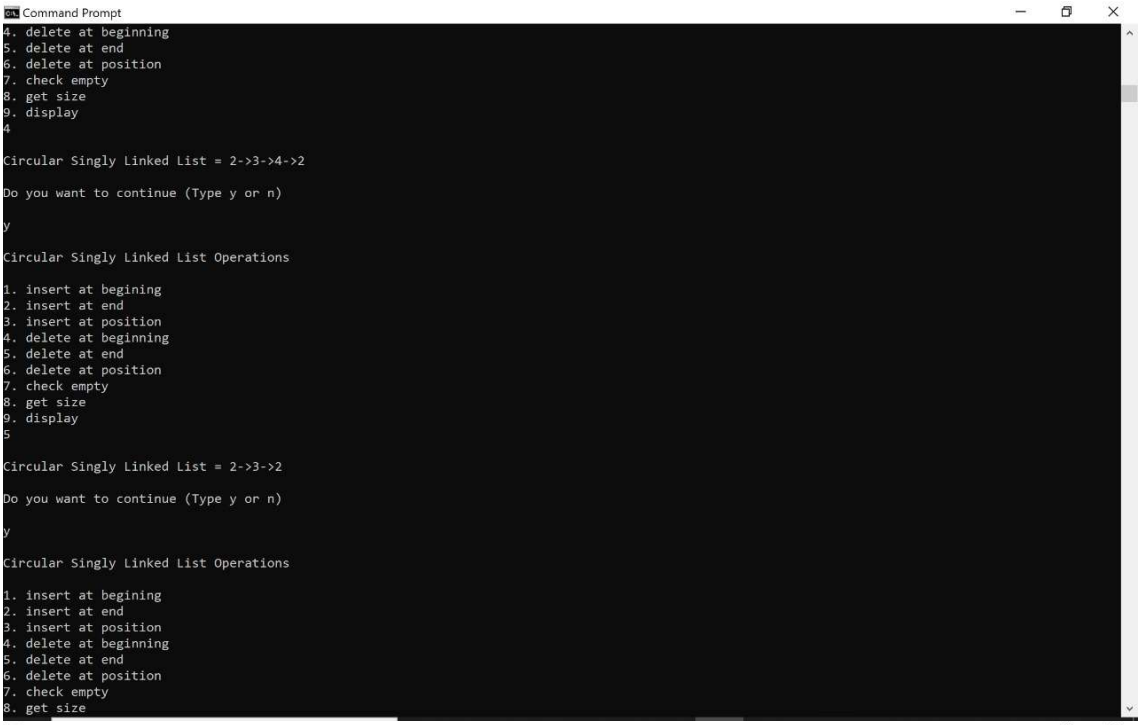
```
System.out.println("8. get size");
System.out.println("9. display");
int choice = scan.nextInt();
switch (choice)
{
case 1 :
   System.out.println("Enter integer element to insert");
   list.insertAtStart( scan.nextInt() );
   break;
case 2 :
   System.out.println("Enter integer element to insert");
   list.insertAtEnd( scan.nextInt() );
   break;
case 3 :
   System.out.println("Enter integer element to insert");
   int num = scan.nextInt() ;
   System.out.println("Enter position");
   int pos = scan.nextInt() ;
   if (pos <= 1 || pos > list.getSize() )
      System.out.println("Invalid position\n");
   else
      list.insertAtPos(num, pos);
   break;
case 4 :
         list.deleteAtBeg();
   break;
case 5 :
   list.deleteAtEnd();
   break;
case 6 :
   System.out.println("Enter position");
   int p = scan.nextInt() ;
   if (p < 1 || p > list.getSize() )
      System.out.println("Invalid position\n");
   else
      list.deleteAtPos(p);
   break;
case 7 :
   System.out.println("Empty status = "+ list.isEmpty());
   break;
case 8 :
         int  s=list.getSize();

   System.out.println("Size = "+ s +" \n");
   break;
   case 9 :
   break;
 default :
   System.out.println("Wrong Entry \n ");
   break;
```

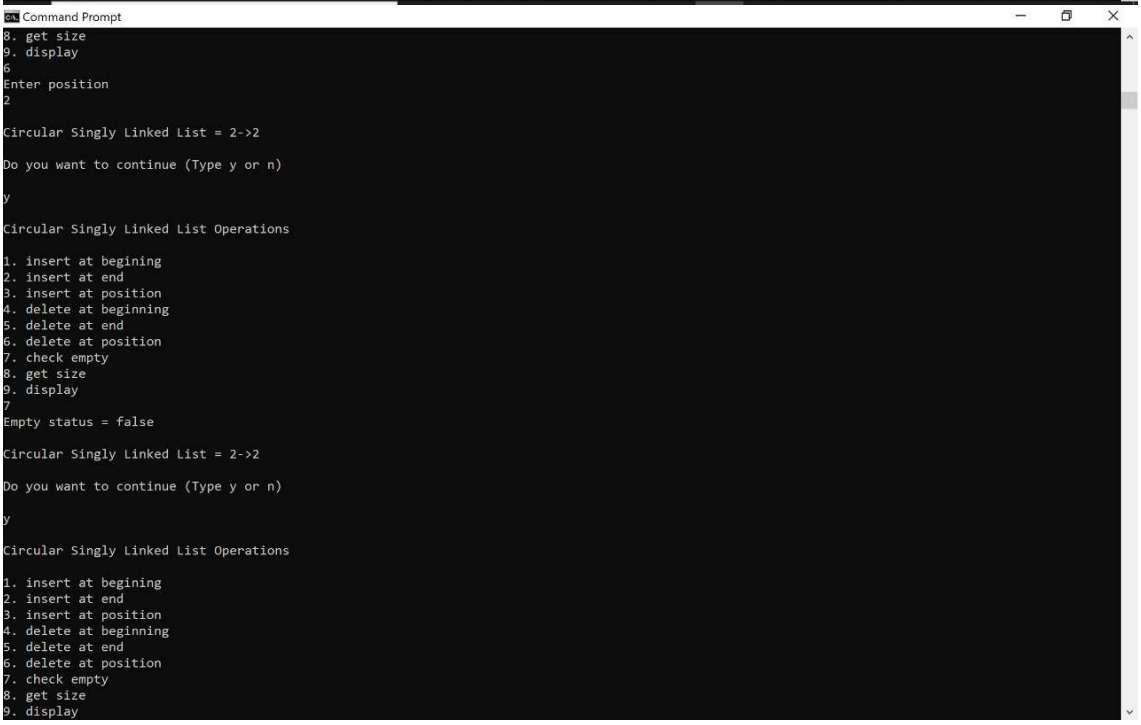
```
            }
            list.display();
            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
}
```

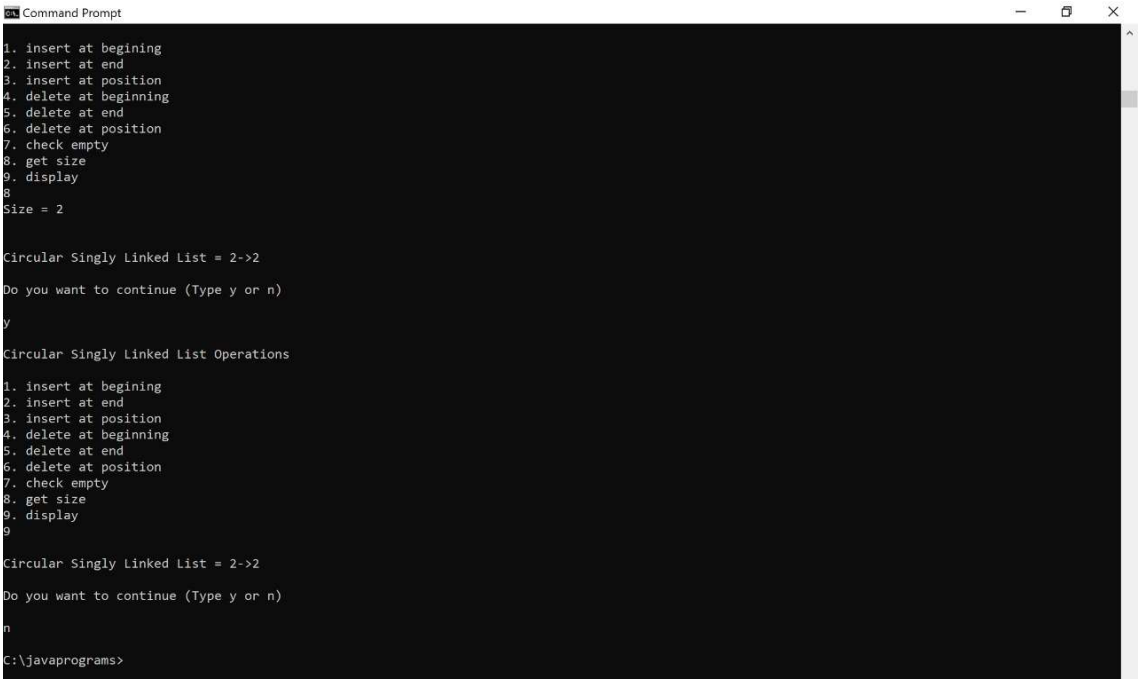
ACTUAL OUTPUT:




Command Prompt
Circular Singly Linked List = 1->4->1
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
3
Enter integer element to insert
2
Enter position
2
Circular Singly Linked List = 1->4->1
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
3
Enter integer element to insert
Command Prompt
8. get size
9. display
3
Enter integer element to insert
3
Enter position
2
Circular Singly Linked List = 1->3->4->1
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
3
Enter integer element to insert
2
Enter position
2
Circular Singly Linked List = 1->2->3->4->1
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning


Command Prompt
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
4
Circular Singly Linked List = 2->3->4->2
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
5
Circular Singly Linked List = 2->3->2
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size


Command Prompt
8. get size
9. display
6
Enter position
2
Circular Singly Linked List = 2->2
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display
7
Empty status = false
Circular Singly Linked List = 2->2
Do you want to continue (Type y or n)
y
Circular Singly Linked List Operations
1. insert at begining
2. insert at end
3. insert at position
4. delete at beginning
5. delete at end
6. delete at position
7. check empty
8. get size
9. display



Experiment -5

Write a java program for performing various operations on stack using linked list

AIM:

Java program for performing various operations on stack using linked list

ALGORITHM:

Step 1: Start.

Step 2: Declare required header files.

Step 3: Create a structure of name node, and declare variables data and link and assign data with zero and link with null (ptr) and also assign values in another constructor, and declare required functions to get the values of data and link and set the values for data and link.

Step 4: Create a classname linkedStack and declare required functions.

Step 5: Initialize top as null and size as zero.

Step 6: Create an object linked Stack for class Linked Stack Implement.

Step 7: Display all operations on linked Stack using switch case and choose an operation from the user and call required function.

Step 8: Define push function and create an object nptr and initialize top as nptr if top is equal to null else the create a element as top for nptr and assign the nptr to top and increase the size.

Step 9: Define pop function and the data is empty in the list. display an underflow exception

or initialize top element to nptr and decrease the size and assign the top element to get the data and return the ptr.

Step 10: Define peek function and if the list empty. throw an underflow exception message and if the data is present in the link display top element using the function getData.

Step 11: Define display function by creating it a variable ptr and traverse it using while loop until it points to null and print value at each iteration.

Step 12: Stop.

SOURCE CODE:

```
import java.util.*;
class Node
{
   protected int data;
   protected Node link;
   public Node()
   {
      link = null;
      data = 0;
   }
   public Node(int d,Node n)
   {
      data = d;
      link = n;
   }
   public void setLink(Node n)
   {
      link = n;
   }
   public void setData(int d)
   {
      data = d;
   }
   public Node getLink()
   {
      return link;
   }
   public int getData()
   {
      return data;
   }
}
class linkedStack
{
   protected Node top ;
   protected int size ;
   public linkedStack()
   {
      top = null;
      size = 0;
```

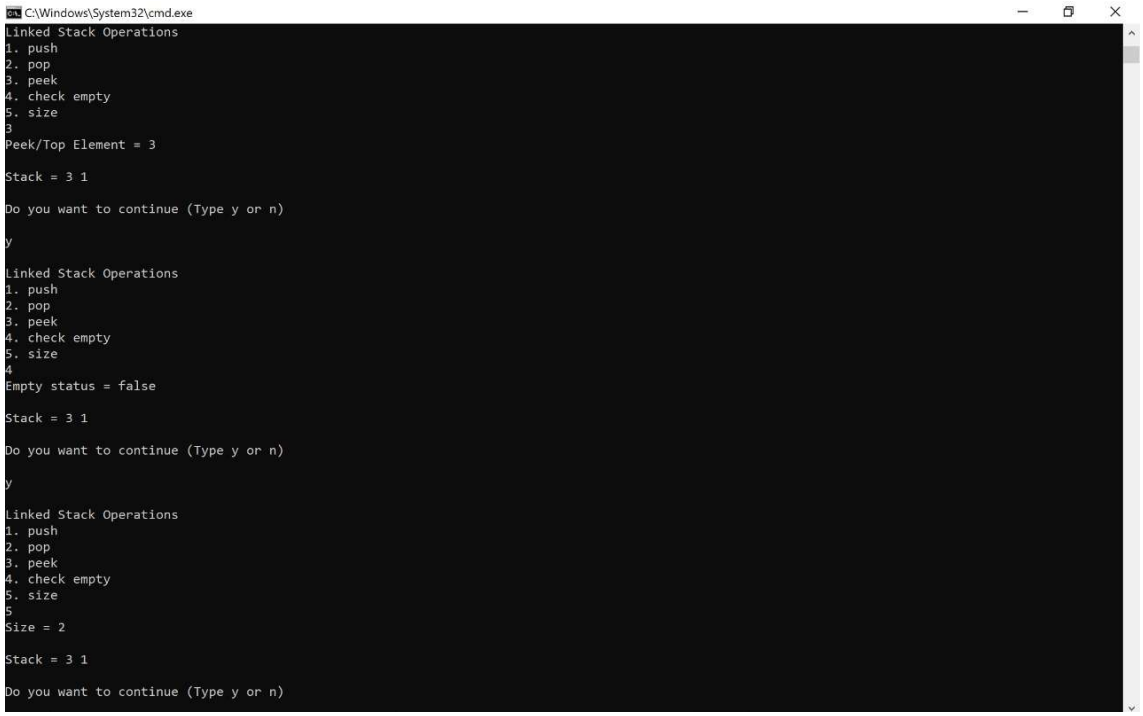
```
}
public boolean isEmpty()
{
   return top == null;
}
public int getSize()
{
   return size;
}
public void push(int data)
{
   Node nptr = new Node (data, null);
   if (top == null)
      top = nptr;
   else
   {
      nptr.setLink(top);
      top = nptr;
   }
   size++ ;
}
public int pop()
{
   if (isEmpty() )
      throw new NoSuchElementException("Underflow Exception") ;
   Node ptr = top;
   top = ptr.getLink();
   size-- ;
   return ptr.getData();
}
public int peek()
{
   if (isEmpty() )
      throw new NoSuchElementException("Underflow Exception") ;
   return top.getData();
}
public void display()
{
   System.out.print("\nStack = ");
   if (size == 0)
   {
      System.out.print("Empty\n");
      return ;
   }
   Node ptr = top;
   while (ptr != null)
   {
      System.out.print(ptr.getData()+" ");
      ptr = ptr.getLink();
   }
```

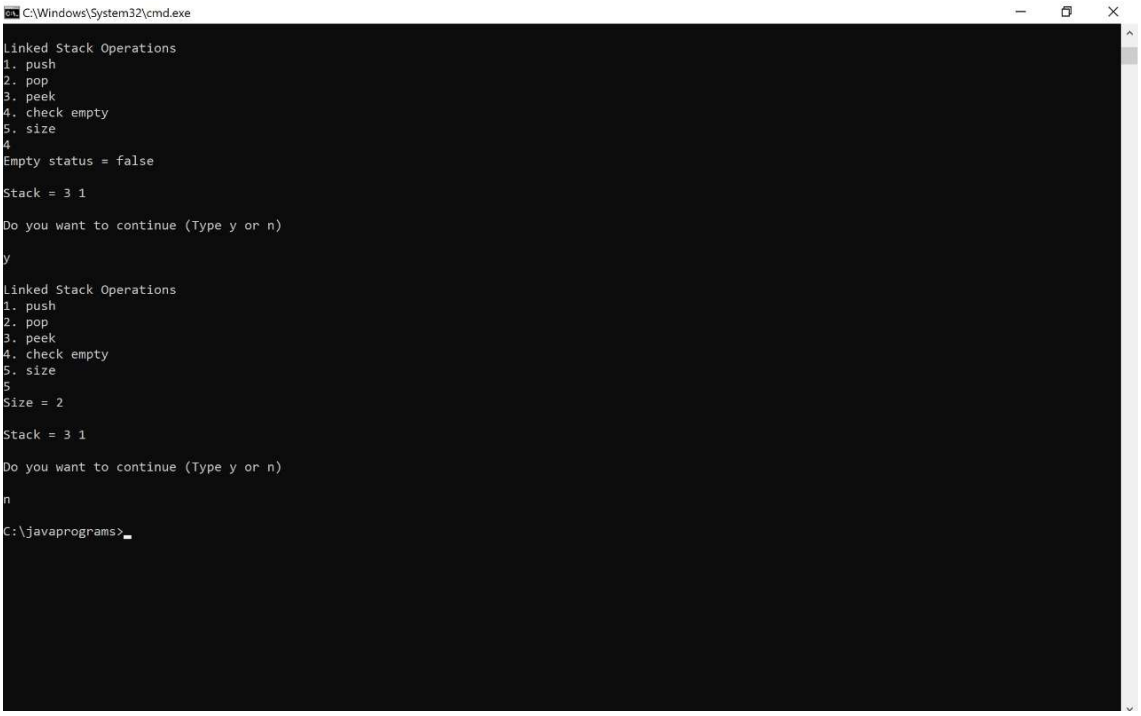
```
        System.out.println();
    }
}
public class LinkedStackImplement
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        linkedStack ls = new linkedStack();
        System.out.println("Linked Stack Test\n");
        char ch;
        do
        {
            System.out.println("\nLinked Stack Operations");
            System.out.println("1. push");
            System.out.println("2. pop");
            System.out.println("3. peek");
            System.out.println("4. check empty");
            System.out.println("5. size");
            int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
                System.out.println("Enter integer element to push");
                ls.push( scan.nextInt() );
                break;
            case 2 :
                try
                {
                    System.out.println("Popped Element = "+ ls.pop());
                }
                catch (Exception e)
                {
                    System.out.println("Error : " + e.getMessage());
                }
                break;
            case 3 :
                try
                {
                    System.out.println("Peek/Top Element = "+ ls.peek());
                }
                catch (Exception e)
                {
                    System.out.println("Error : " + e.getMessage());
                }
                break;
            case 4 :
                System.out.println("Empty status = "+ ls.isEmpty());
                break;
            case 5 :
```

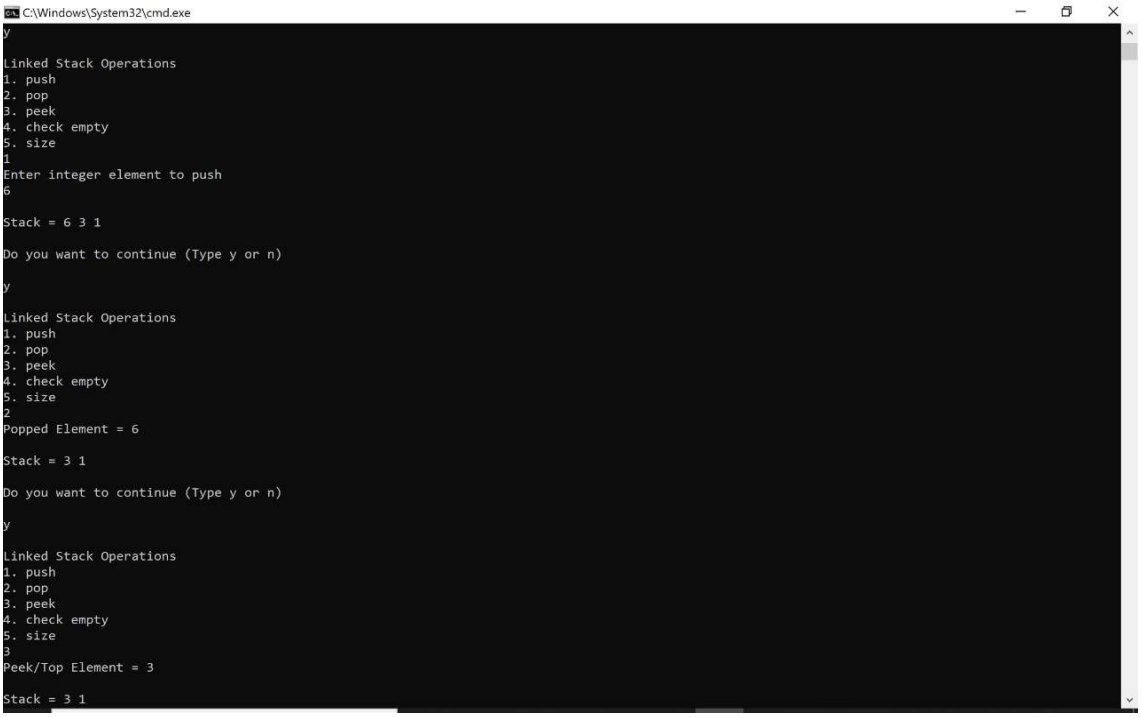
```
            System.out.println("Size = "+ ls.getSize());
            break;
         case 6 :
            System.out.println("Stack = ");
            ls.display();
            break;
         default :
            System.out.println("Wrong Entry \n ");
            break;
         }
         ls.display();
         System.out.println("\nDo you want to continue (Type y or n) \n");
         ch = scan.next().charAt(0);

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

ACTUAL OUTPUT:


C:\Windows\System32\cmd.exe
C:\javaprograms>javac LinkedStackImplement.java
C:\javaprograms>java LinkedStackImplement
Linked Stack Test
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
1
Enter integer element to push
1
Stack = 1
Do you want to continue (Type y or n)
y
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
1
Enter integer element to push
3
Stack = 3 1
Do you want to continue (Type y or n)
y


C:\Windows\System32\cmd.exe
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
3
Peek/Top Element = 3
Stack = 3 1
Do you want to continue (Type y or n)
y
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
4
Empty status = false
Stack = 3 1
Do you want to continue (Type y or n)
y
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
5
Size = 2
Stack = 3 1
Do you want to continue (Type y or n)


C:\Windows\System32\cmd.exe
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
4
Empty status = false
Stack = 3 1
Do you want to continue (Type y or n)
y
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
5
Size = 2
Stack = 3 1
Do you want to continue (Type y or n)
n
C:\javaprograms>


C:\Windows\System32\cmd.exe
y
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
1
Enter integer element to push
6
Stack = 6 3 1
Do you want to continue (Type y or n)
y
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
2
Popped Element = 6
Stack = 3 1
Do you want to continue (Type y or n)
y
Linked Stack Operations
1. push
2. pop
3. peek
4. check empty
5. size
3
Peek/Top Element = 3
Stack = 3 1

Experiment -6

Write a Java program for performing various operations on queue using linked list

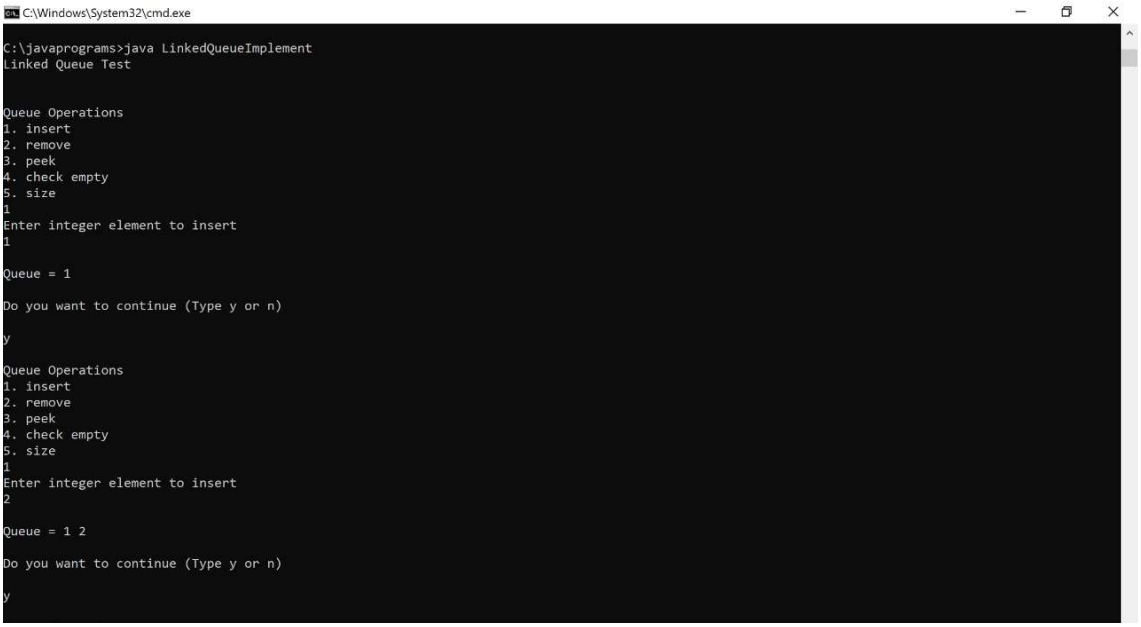
AIM:

Java program for performing various operations queue using linked list

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a structure of name node and declare variables data and link and assign data with zero and link with null (nptr) and also assign values in another constructor and declare required functions to get the values of data and link and (required functions) set the values for data and link.
Step 4: Create a class name linkedQueue and declare required function.
Step 5: Initialize top at null and size at zero.
Step 6: Create an obj linkedQueue for class Linked Queue Implement.
Step 7: Display all operations to on linked Queue using switch case and choose an operation from the user and call required function.
Step 8: Define enqueue function by reading the elements to be inserted by the user and create 2 variables front and rear and assign the nptr value for both by creating an object, if rear is equal to null else assign it to next of the start and adjust the pointers.
Step 9: Define dequeue function to remove the element from front if the list is empty, thrown on underflow exception else the adjust the nptr to remove first element.
Step 10: Define peek function, if the list is empty throw an underflow exception else display the first element using front data.
Step 11: Define display function and by creating a variable and point to the start and traverse it using while loop until it points to null and print value at each iteration.
Step 12: Define checkEmpty function to check whether the given list is empty or not.
Step13: Define getSize function and create size variable and return the size of the list.
Step 14: Stop.

EXPECTED OUTPUT:



SOURCE CODE:

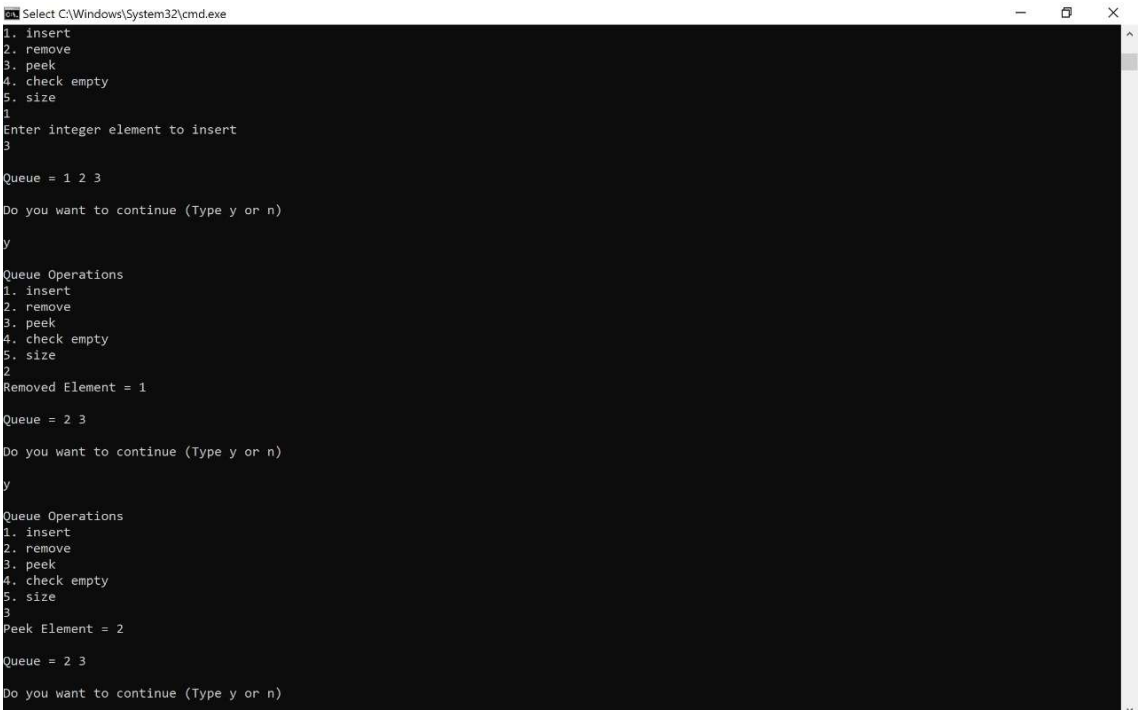
```
import java.util.*;
class Node
{
   protected int data;
   protected Node link;
   public Node()
   {
      link = null;
      data = 0;
   }
   public Node(int d,Node n)
   {
```

```
        data = d;
        link = n;
    }
    public void setLink(Node n)
    {       link = n;
    }
    public void setData(int d)
    {
        data = d;
    }
    public Node getLink()
    {
        return link;
    }
    public int getData()
    {
        return data;
    }
}
class linkedQueue
{
    protected Node front, rear;
    public int size;
    public linkedQueue()
    {
        front = null;
        rear = null;
        size = 0;
    }
    public boolean isEmpty()
    {
        return front == null;
    }
    public int getSize()
    {
        return size;
    }
    public void insert(int data)
    {
        Node nptr = new Node(data, null);
        if (rear == null)
        {
            front = nptr;
            rear = nptr;
        }
        else
        {
            rear.setLink(nptr);
            rear = rear.getLink();
        }
```

```
        size++ ;
    }
    public int remove()
    {
        if (isEmpty() )
            throw new NoSuchElementException("Underflow Exception");
        Node ptr = front;
        front = ptr.getLink();
        if (front == null)
            rear = null;
        size-- ;
        return ptr.getData();
    }
    public int peek()
    {
        if (isEmpty() )
            throw new NoSuchElementException("Underflow Exception");
        return front.getData();
    }
    public void display()
    {
        System.out.print("\nQueue = ");
        if (size == 0)
        {
            System.out.print("Empty\n");
            return ;
        }
        Node ptr = front;
        while (ptr != rear.getLink() )
        {
            System.out.print(ptr.getData()+" ");
            ptr = ptr.getLink();
        }
        System.out.println();
    }
}
public class LinkedQueueImplement
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        linkedQueue lq = new linkedQueue();
        System.out.println("Linked Queue Test\n");
        char ch;
        do
        {
            System.out.println("\nQueue Operations");
            System.out.println("1. insert");
            System.out.println("2. remove");
            System.out.println("3. peek");
```

```
            System.out.println("4. check empty");
            System.out.println("5. size");
            int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
               System.out.println("Enter integer element to insert");
               lq.insert( scan.nextInt() );
               break;
            case 2 :
               try
               {
                  System.out.println("Removed Element = "+ lq.remove());
               }
               catch (Exception e)
               {
                  System.out.println("Error : " + e.getMessage());
               }
               break;
            case 3 :
               try
               {
                  System.out.println("Peek Element = "+ lq.peek());
               }
               catch (Exception e)
               {
                  System.out.println("Error : " + e.getMessage());
               }
               break;
            case 4 :
               System.out.println("Empty status = "+ lq.isEmpty());
               break;

            case 5 :
               System.out.println("Size = "+ lq.getSize());
               break;

            default :
               System.out.println("Wrong Entry \n ");
               break;
            }
            lq.display();

            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);
         } while (ch == 'Y'|| ch == 'y');
      }
   }
```

ACTUAL OUTPUT:



Experiment -7

Write a java program for the following using stack

a) Infix to postfix conversion

AIM:

Java program using stack implement infix to postfix convention

ALGORITHM:

Step 1: Start.

Step 2: Scan the infix notation from left to right one character at a time.

Step 3: If the next symbol scanned as an operand, append it to the postfix string.

Step 4: If the next symbol scanned as an operator the

i) Pop and append to the postfix string every operator on the stack that:

a) Is above the most recently scanned left paranthesis and

b) Has precedence higher than or is a right-associative operator of equal precedence to that of the new operator symbol.

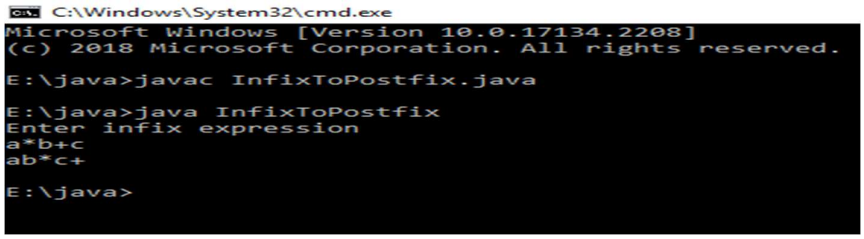
ii) Push the new operator onto the stack.

Step 5: If a left parenthesis is scanned, push it into the stack.

Step 6: If a right paranthesis is scanned, all operators down the most recently scanned left parenthesis must be popped and appended to the postfix string. Futhermore, the pair of parenthesis must be discarded.

Step 7: When the infix string is fully scanned, the stack may contain some operators all the remaining operators should be popped and appended to the postfix String.

EXPECTED OUTPUT:



SOURCE CODE:

```
import java.io.IOException;
import java.util.Stack;
import java.io.*;
class InfixToPostfix{
   static int Prec(char ch) {
 switch (ch)  {
       case '+':
       case '-':
          return 1;
      case '*':
       case '/':
          return 2;
   case '^':
          return 3;
       }
       return -1;
   }
   static String infixToPostfix(String exp)  {
```

```
        String result = new String("");
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i<exp.length(); ++i)  {
           char c = exp.charAt(i);
           if (Character.isLetterOrDigit(c))
              result += c;
           else if (c == '(')
              stack.push(c);
           else if (c == ')'){
              while (!stack.isEmpty() &&
                    stack.peek() != '(')
                 result += stack.pop();
                 stack.pop();
           }
           else   {
              while (!stack.isEmpty() && Prec(c) <= Prec(stack.peek())){
                 result += stack.pop();
            }
              stack.push(c);
           }
        }
        while (!stack.isEmpty()){
           if(stack.peek() == '(')
              return "Invalid Expression";
           result += stack.pop();
         }
        return result;
    }
    public static void main(String[] args)   {
            BufferedReader keyboard = new BufferedReader (new
InputStreamReader(System.in));
            try{
            System.out.println("Enter infix expression ");
            String exp = keyboard.readLine();
            System.out.println(infixToPostfix(exp));
        }
            catch(Exception e){}
    }
}
```

ACTUAL OUTPUT:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17134.2208]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\java>javac InfixToPostfix.java

E:\java>java InfixToPostfix
Enter infix expression
a*b+c
ab*c+

E:\java>
```

7b) Expression evaluation

AIM:

Java program to implement expression evaluation

ALGORITHM:

Step 1: Start

Step 2: Intialize a string consisting of expression and two stacks for storing values and operators.

Step 3: Iterate from 0 to size of string-1. Check if the character at the current index is equal to space start the next iteration. If the character at the current index is equal to '(' insert it in operators stack.

Step 4: If the character at the current index is a digit-Check if there is a number following next to the digit, pick the whole number else if it is a single-digit pick the digit. Insert the digit in the value stack.

Step 5: Else if the character at the current index it ')' iterate while the size of the operators stack is not zero and character at the current index is not equal to '('.

Step 6: After that, remove the 2 elements from the top of the values stack and 1 element from the operator stack.

Step 7: Apply the arithmetic operation on the two popped digits. Insert the answer in a values stack.

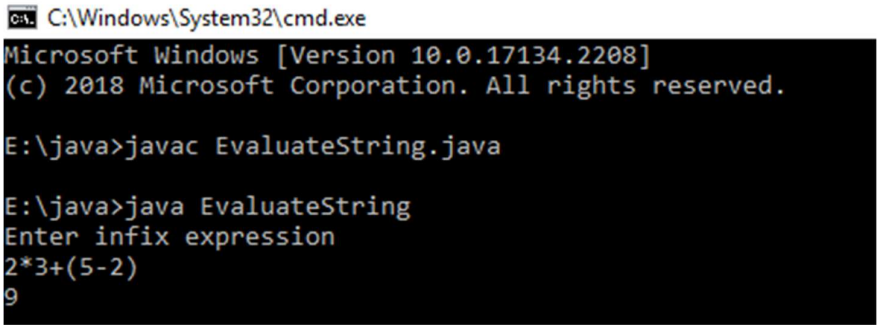
Step 8: Similarly, while the size of the operators stack is not zero, remove the 2 elements at the top of the values stack and an operator from operator stack.

Step 9: Apply the arithmetic operation on the two popped digits.Insert the answer in a value stack.

Step 10: Return the element at the top of the value stack.

Step 11: Stop.

EXPECTED OUTPUT:



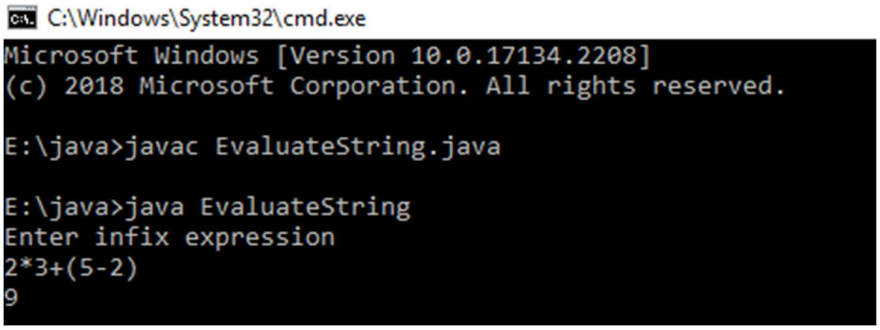
SOURCE CODE:

```
import java.util.Stack;
import java.io.IOException;
import java.io.*;
public class EvaluateString{
   public static int evaluate(String expression)  {
      char[] tokens = expression.toCharArray();
      Stack<Integer> values = new Stack<Integer>();
      Stack<Character> ops = new Stack<Character>();
```

```
    for (int i = 0; i < tokens.length; i++){
        if (tokens[i] == ' ')
            continue;
        if (tokens[i] >= '0' && tokens[i] <= '9')
        {
            StringBuffer sbuf = new StringBuffer();
            while (i < tokens.length && tokens[i] >= '0' && tokens[i] <= '9')
                sbuf.append(tokens[i++]);
            values.push(Integer.parseInt(sbuf. toString()));
             i--;
        }
        else if (tokens[i] == '(')
            ops.push(tokens[i]);
        else if (tokens[i] == ')')
        {
            while (ops.peek() != '(')
              values.push(applyOp(ops.pop(),
                          values.pop(),
                         values.pop()));
            ops.pop();
        }
        else if (tokens[i] == '+' || tokens[i] == '-' ||tokens[i] == '*' ||tokens[i] == '/')
        {
            while (!ops.empty() &&hasPrecedence(tokens[i],ops.peek()))
              values.push(applyOp(ops.pop(),
                          values.pop(),
                         values.pop()));
            ops.push(tokens[i]);
        }
    }
    while (!ops.empty())
        values.push(applyOp(ops.pop(),
                    values.pop(),
                   values.pop()));
    return values.pop();
}
public static boolean hasPrecedence(char op1, char op2)
{
    if (op2 == '(' || op2 == ')')
        return false;
    if ((op1 == '*' || op1 == '/') &&  (op2 == '+' || op2 == '-'))
        return false;
    else
        return true;
}
public static int applyOp(char op,   int b, int a)
{
    switch (op)
    {
    case '+':
```

```
            return a + b;
        case '-':
            return a - b;
        case '*':
            return a * b;
        case '/':
            if (b == 0)
                throw new UnsupportedOperationException( "Cannot divide by zero");
            return a / b;
        }
        return 0;
    }
    public static void main(String[] args)
    {
BufferedReader keyboard = new BufferedReader(newInputStreamReader(System.in));
try{
            System.out.println("Enter infix expression ");
            String exp = keyboard.readLine();
            System.out.println(EvaluateString.evaluate(exp));
        }
            catch(Exception e){}
                  evaluate("10 + 2 * 6"));
        System.out.println(EvaluateString.
                  evaluate("100 * 2 + 12"));
        System.out.println(EvaluateString.
                 evaluate("100 * ( 2 + 12 )"));
        System.out.println(EvaluateString.
             evaluate("100 * ( 2 + 12 ) / 14"));*/
    }
}
```

ACTUAL OUTPUT:



7c) Obtain the binary number for a given decimal number

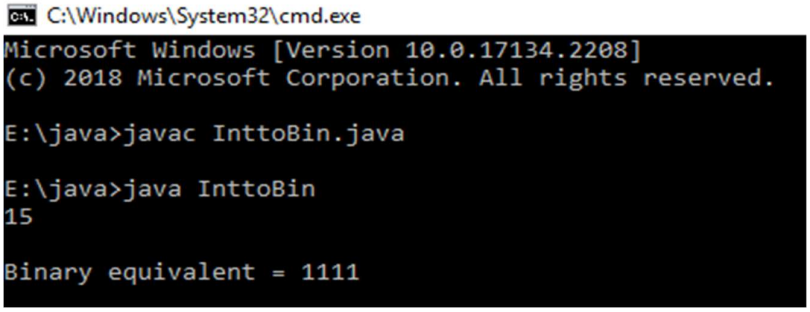
AIM:

Java program to implement the binary number for a given decimal number

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a structure of name node and declare structure variables.
Step 4: Create a class linked Stack and declare required functions.
Step 5: Initialize link and data inside the constructor and declare required functions to get and assign data and link.
Step 6: Define push function and create an object nptr and intialize top as nptr if top its equal to null else create a element as top for nptr and assign the nptr to top and increase the size.
Step 7: Define pop function and the data is empty in the list,display an underflow exception or initialize top element to nptr and decrease the size and assign the top element to get the data and return the ptr.
Step 8: Define peek function and if the list is empty throw an underflow exception message and if the data is present in the link, display top element using the function getData.
Step 9: Define display function by creating a variable ptr and traverse it using while loop until it points to null and print value at each iteration.
Step 10: Create a class InttoBin and by considering input from the user, and perform divide by 2 algorithm and store the value and display it using display function.

EXPECTED OUTPUT:



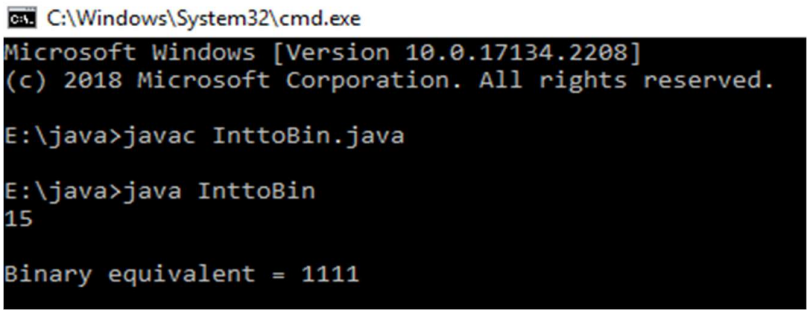
SOURCE CODE:

```
import java.util.*;
class Node{
   protected int data;
   protected Node link;
   public Node()
   {
      link = null;
      data = 0;
   }
   public Node(int d,Node n) {
      data = d;

      link = n;
   }
   public void setLink(Node n) {
```

```
        link = n;
    }
    public void setData(int d){
        data = d;
    }
    public Node getLink() {
        return link;
    }
    public int getData(){
        return data;
    }
}
class linkedStack{
    protected Node top ;
    protected int size ;
    public linkedStack() {
        top = null;
        size = 0;
    }
    public boolean isEmpty(){
        return top == null;
    }
    public int getSize() {
        return size;
    }
    public void push(int data){
        Node nptr = new Node (data, null);
        if (top == null)
            top = nptr;
        else{
            nptr.setLink(top);
            top = nptr;
        }
        size++ ;
    }
    public int pop()  {
        if (isEmpty() )
            throw new NoSuchElementException("Underflow Exception") ;
        Node ptr = top;
        top = ptr.getLink();
        size-- ;
        return ptr.getData();
    }
    public int peek()
    {
        if (isEmpty() )
            throw new NoSuchElementException("Underflow Exception") ;
        return top.getData();
    }
```

```
    public void display() {
        System.out.print("\nStack = ");
        if (size == 0)
        {
            System.out.print("Empty\n");
            return ;
        }
        Node ptr = top;
        while (ptr != null)
        {
            System.out.print(ptr.getData()+" ");
            ptr = ptr.getLink();
        }
        System.out.println();
    }
}
public class InttoBin{
    public static void main(String[] args)  {
        Scanner scan = new Scanner(System.in);
        linkedStack ls = new linkedStack();
                        int num=scan.nextInt();
              while (num != 0){
                        int d = num % 2;
                        ls.push(d);
                        num /= 2;
              }
                        System.out.print("\nBinary equivalent = ");
        while (!(ls.isEmpty() ))
        {
            System.out.print(ls.pop());
        }
        System.out.println();
    }
}
```

ACTUAL OUTPUT:



Experiment -8

Write a java program to implement various, operations on Binary Search Tree using Recursive and Non-Recursive methods

AIM:

Java program to implement various operations on Binary Search Tree using Recursive and Non-Recursive methods

ALGORITHM (Recursive) :

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a class BST Node and declare required variables inside the constructor and initialize it to null.
Step 4: Declare required functions to get and set the node.
Step 5: Create a class BST and declare insert function to insert the data by reading element by the user by considering the data to be inserted at left or right of the BST.
Step 6: Define delete function by considering data from the user and if data is not present in the BST, display message and the data is less than the root, traverse it using the recursive function and the delete element at that particular location, and the data is greater than the root, follow the steps done in left BST.
Step 7: Define count nodes function and by using recursive function of it, traverse each data and store the values and print count until the r becomes null.
Step 8: Define search function and return the search element and by using that function recursively, search the data entered by the user and display found message if found.
Step 9: Define inorder function and return the value of root by calling it recursively initially set the BST as left ,data and right using inorder function until r is not equals to null.
Step 10: Define preorder function and return the value of root and by calling it recursively if it to be inserted in the order data, left and right insert using preorder recursive function is not equals to null.
Step 11: Define post order function and return the value of root and calling it recursively if it be inserted in the order left,right and data insert it using post order recursive function until r is not equals to null.
Step 12: Stop.

SOURCE CODE:

```
import java.util.Scanner;
 class BSTNode
 {
    BSTNode left, right;
    int data;
    public BSTNode()
    {
       left = null;
       right = null;
       data = 0;
    }
    public BSTNode(int n) {
       left = null;
       right = null;
```

```
        data = n;
    }
    public void setLeft(BSTNode n)  {
        left = n;
    }
    public void setRight(BSTNode n){
        right = n;
    }
    public BSTNode getLeft(){
        return left;
    }
    public BSTNode getRight()  {
        return right;
    }
    public void setData(int d){
        data = d;
    }
    public int getData()  {
        return data;
    }
}
class BS {
    private BSTNode root;
    public BST() {
        root = null;
    }
    public boolean isEmpty() {
        return root == null;
    }
    public void insert(int data) {
        root = insert(root, data);
    }
    private BSTNode insert(BSTNode node, int data){
        if (node == null)
            node = new BSTNode(data);
        else {
            if (data <= node.getData())
                node.left = insert(node.left, data);
            else
                node.right = insert(node.right, data);
        }
        return node;
    }
    public void delete(int k) {
        if (isEmpty())
            System.out.println("Tree Empty");
        else if (search(k) == false)
            System.out.println("Sorry "+ k +" is not present");
        else
        {
```

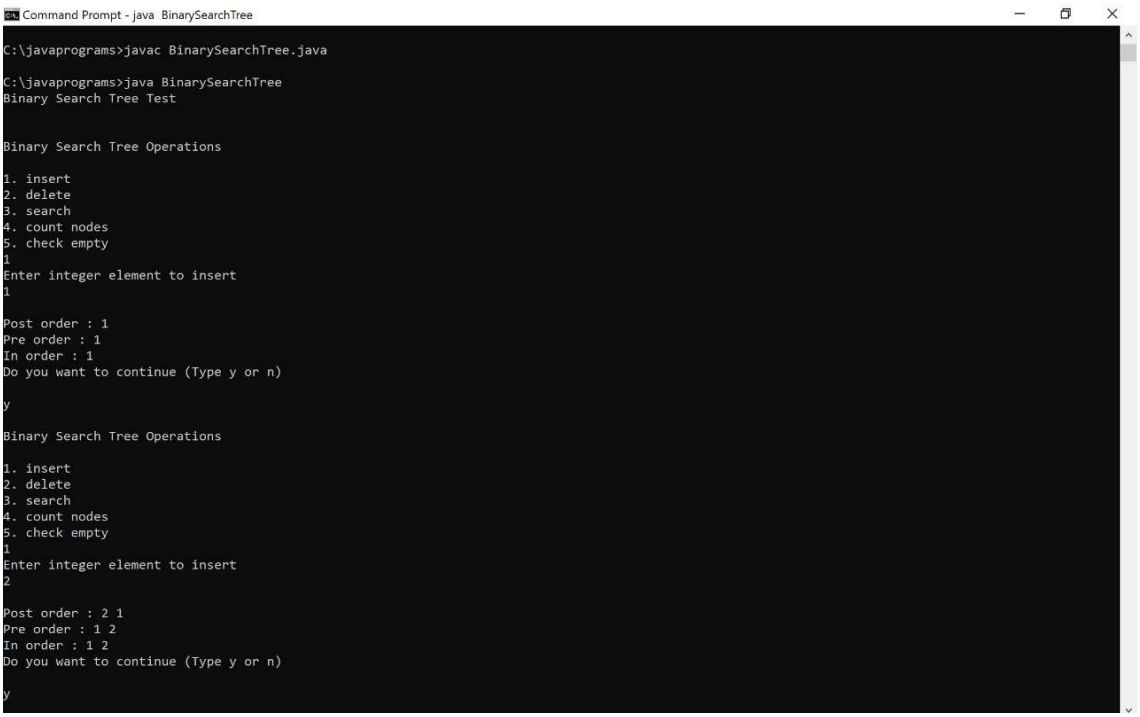
```
            root = delete(root, k);
            System.out.println(k+ " deleted from the tree");
        }
    }
    private BSTNode delete(BSTNode root, int k){
        BSTNode p, p2, n;
        if (root.getData() == k)  {
            BSTNode lt, rt;
            lt = root.getLeft();
            rt = root.getRight();
            if (lt == null && rt == null)
                return null;
            else if (lt == null)  {
                p = rt;
                return p;
            }
            else if (rt == null){
                p = lt;
                return p;
            }
            else  {
                p2 = rt;
                p = rt;
                while (p.getLeft() != null)
                    p = p.getLeft();
                p.setLeft(lt);
                return p2;
            }
        }
        if (k < root.getData()){
            n = delete(root.getLeft(), k);
            root.setLeft(n);
        }
        else
{
            n = delete(root.getRight(), k);
            root.setRight(n);
        }
        return root;
    }
    public int countNodes()  {
        return countNodes(root);
    }
    private int countNodes(BSTNode r){
        if (r == null)
            return 0;
        else
        {
            int l = 1;
            l += countNodes(r.getLeft());
```

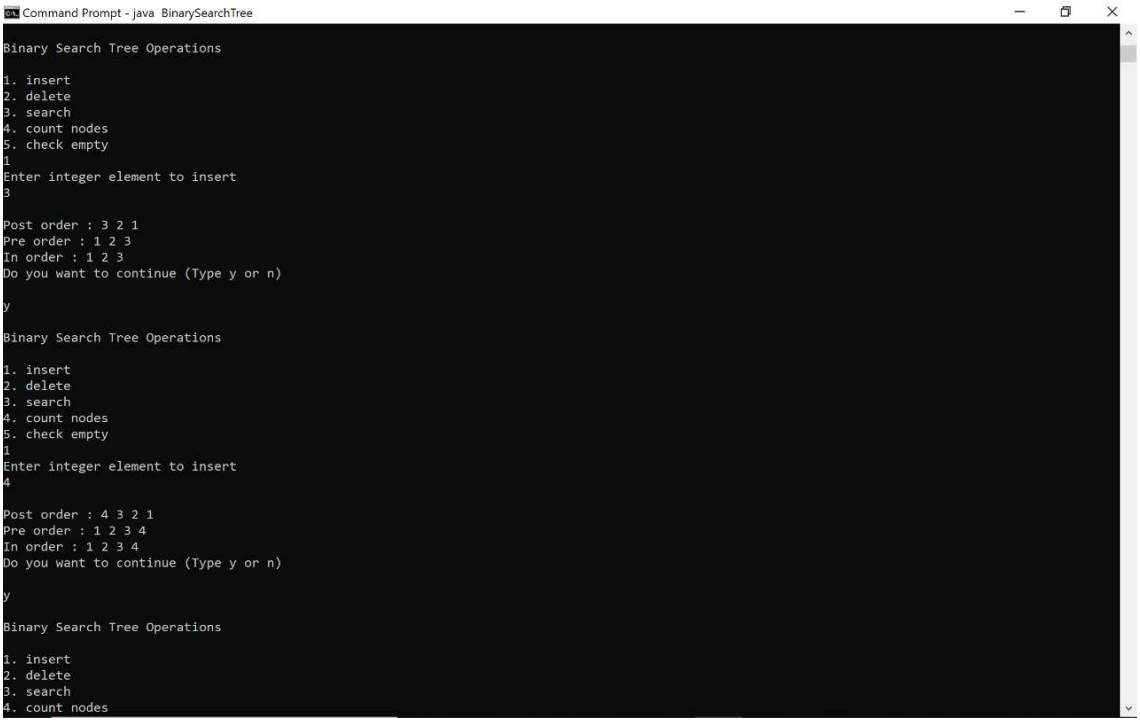
```
        l += countNodes(r.getRight());
        return l;
    }
}
public boolean search(int val){
    return search(root, val);
}
private boolean search(BSTNode r, int val){
    boolean found = false;
    while ((r != null) && !found){
        int rval = r.getData();
        if (val < rval)
            r = r.getLeft();
        else if (val > rval)
            r = r.getRight();
        else
        {
            found = true;
            break;
        }
        found = search(r, val);
    }
    return found;
}
public void inorder()
{
    inorder(root);
}
private void inorder(BSTNode r)
{
    if (r != null)
    {
        inorder(r.getLeft());
        System.out.print(r.getData() +" ");
        inorder(r.getRight());
    }
}
public void preorder()
{
    preorder(root);
}
private void preorder(BSTNode r)
{
    if (r != null)
    {
        System.out.print(r.getData() +" ");
        preorder(r.getLeft());
        preorder(r.getRight());
    }
}
```

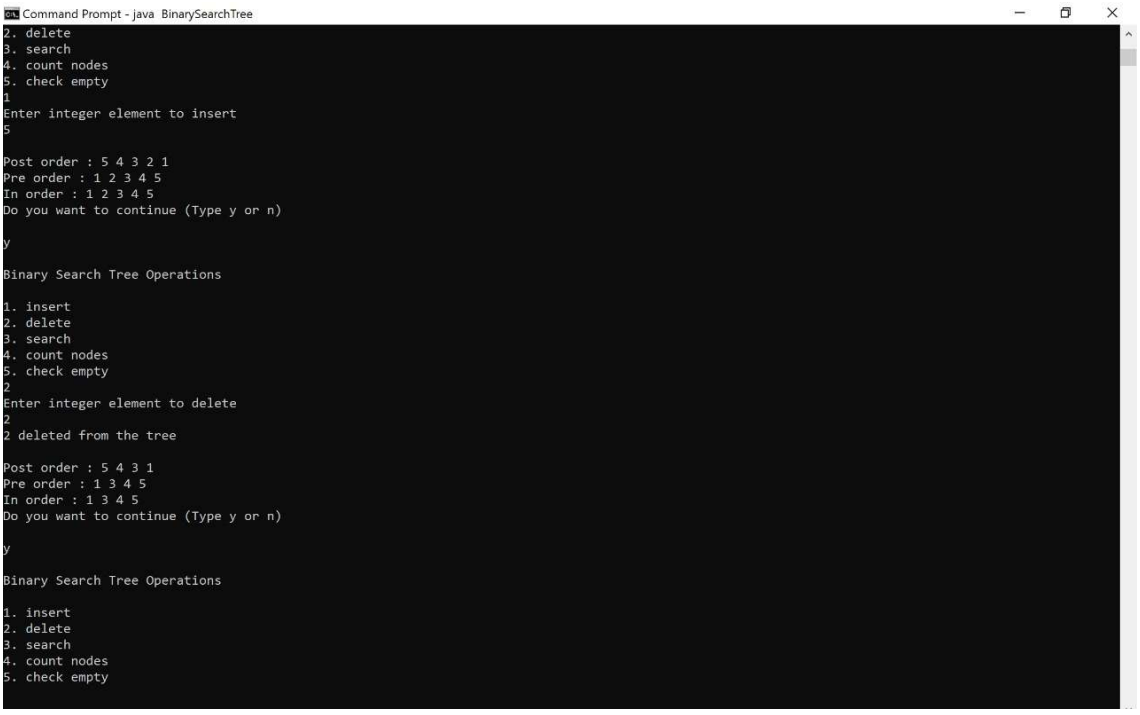
```
    public void postorder()
    {
        postorder(root);
    }
    private void postorder(BSTNode r)
    {
        if (r != null)
        {
            postorder(r.getLeft());
            postorder(r.getRight());
            System.out.print(r.getData() +" ");
        }
    }
}
public class BinarySearchTree
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        BST bst = new BST();
        System.out.println("Binary Search Tree Test\n");
        char ch;
        do
        {
            System.out.println("\nBinary Search Tree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. delete");
            System.out.println("3. search");
            System.out.println("4. count nodes");
            System.out.println("5. check empty");
            int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
                System.out.println("Enter integer element to insert");
                bst.insert( scan.nextInt() );
                break;
            case 2 :
                System.out.println("Enter integer element to delete");
                bst.delete( scan.nextInt() );
                break;
            case 3 :
                System.out.println("Enter integer element to search");
                System.out.println("Search result : "+ bst.search( scan.nextInt() ));
                break;
            case 4 :
                System.out.println("Nodes = "+ bst.countNodes());
                break;
            case 5 :
                System.out.println("Empty status = "+ bst.isEmpty());
```

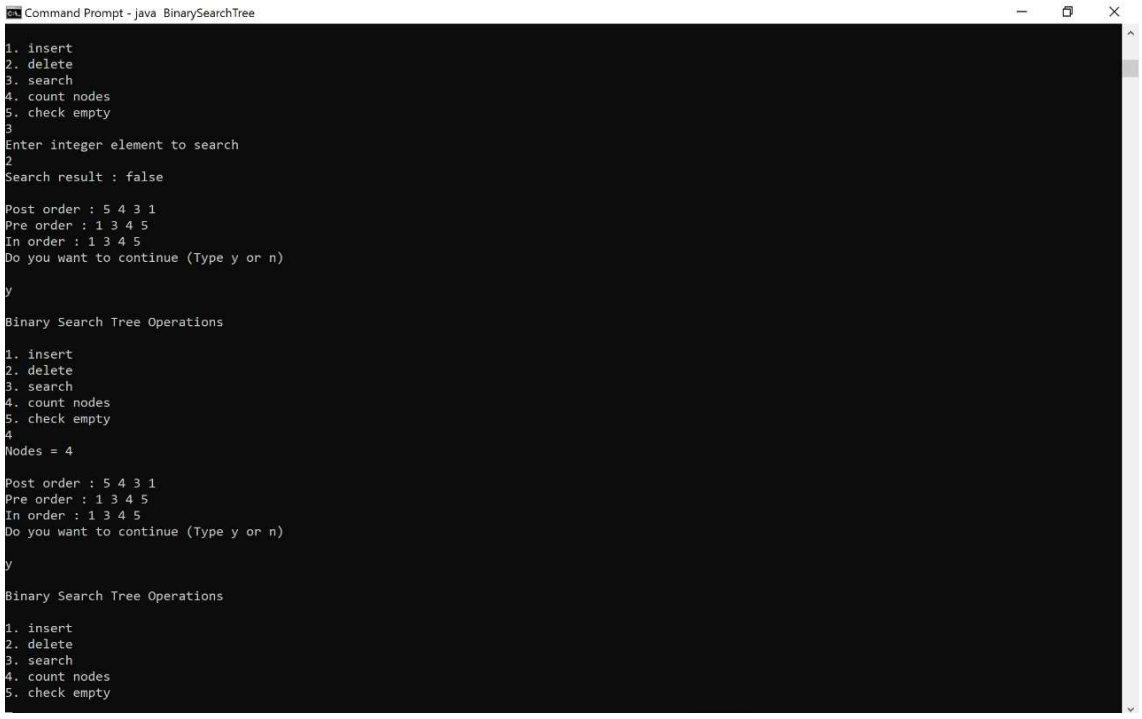
```
            break;
        default :
            System.out.println("Wrong Entry \n ");
            break;
        }
        System.out.print("\nPost order : ");
        bst.postorder();
        System.out.print("\nPre order : ");
        bst.preorder();
        System.out.print("\nIn order : ");
        bst.inorder();

        System.out.println("\nDo you want to continue (Type y or n) \n");
        ch = scan.next().charAt(0);
    } while (ch == 'Y'|| ch == 'y');
  }
}
```

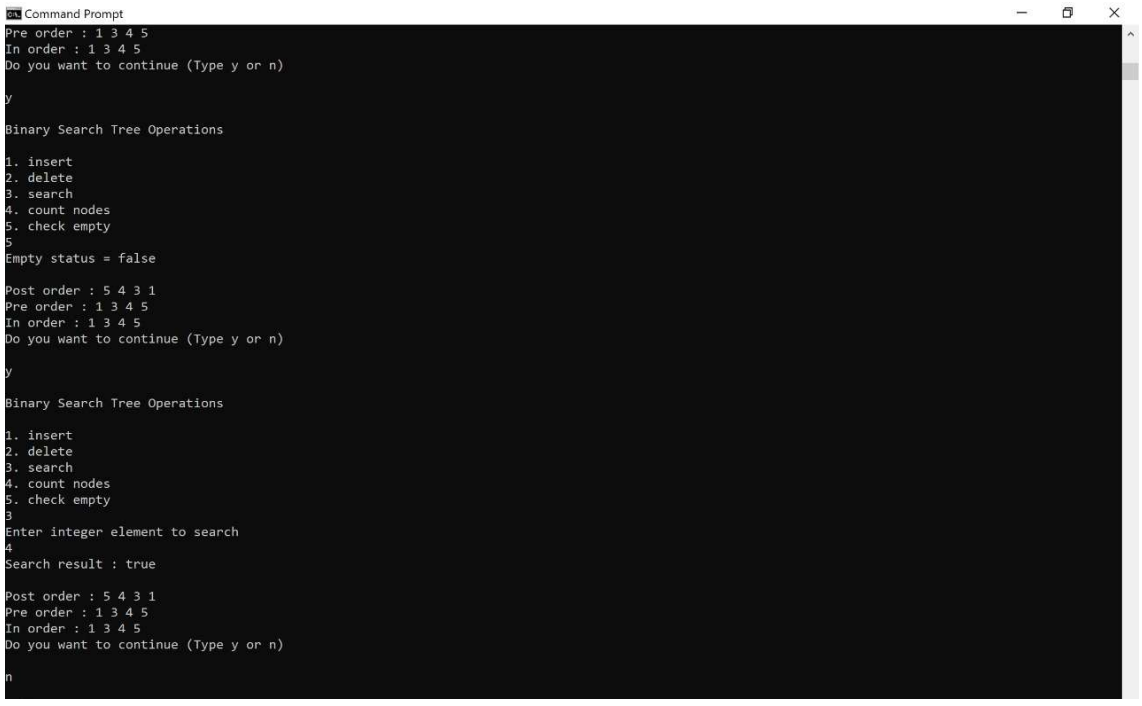
ACTUAL OUTPUT:




Command Prompt - java BinarySearchTree
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
3
Post order : 3 2 1
Pre order : 1 2 3
In order : 1 2 3
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
4
Post order : 4 3 2 1
Pre order : 1 2 3 4
In order : 1 2 3 4
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes


Command Prompt - java BinarySearchTree
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
5
Post order : 5 4 3 2 1
Pre order : 1 2 3 4 5
In order : 1 2 3 4 5
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
2
2 deleted from the tree
Post order : 5 4 3 1
Pre order : 1 3 4 5
In order : 1 3 4 5
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty


Command Prompt - java BinarySearchTree
1. insert
2. delete
3. search
4. count nodes
5. check empty
3
Enter integer element to search
2
Search result : false
Post order : 5 4 3 1
Pre order : 1 3 4 5
In order : 1 3 4 5
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
4
Nodes = 4
Post order : 5 4 3 1
Pre order : 1 3 4 5
In order : 1 3 4 5
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty


Command Prompt
Pre order : 1 3 4 5
In order : 1 3 4 5
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
5
Empty status = false
Post order : 5 4 3 1
Pre order : 1 3 4 5
In order : 1 3 4 5
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
3
Enter integer element to search
4
Search result : true
Post order : 5 4 3 1
Pre order : 1 3 4 5
In order : 1 3 4 5
Do you want to continue (Type y or n)
n

ALGORITHM (Recursive) :

Step 1: Start.

Step 2: Declare required header files.

Step 3: Create a structure name Node and declare required variables let, left and right.

Step 4: Create a function for the node and create an object for it and declare and assign required variables and initiative it to null.

Step 5: Define insert function for the particular node and by considering element from the user and point to start traversing from root and traverse downward path to search where the new node to be inserted. If the root is null for the tree is empty the new node is the root node. If the new nock element/key ft. less than the leaf node key, assign the new node to be it's left child else assign the new node its right child, return the pointer where the bow node is inserted.

Step 6: Define delete function and by considering the root and prev traverse it until the element was found which need to deleted and delete the particular key, if the element was not found to be deleted, display an error message.

Step 7: Define search method and by considering input from the user which element should be found by traveling every location by using stack push every key in it and check whether the key is present or not. if the key found, display in message.

Step 8: Define Inorder and create a stack obj(node) and initialize the root to current data. By considering the input from the user, the data should be inserted in the order of left, key and right.

Step 9: Define Preorder and create a stack object (node) and Initialize the root to current data. By considering the different inputs the to from the user place the keys in the stack and insert in the order of day right, left.

Step 10: Define Postorder and create a stack object (node) and initialize the root to current data and by considering the inputs current by from the user, place the keys in the stack by using push operation to insert and after investing in the tree, delete the key using pop operations in the order of left, right,

Step 11: Define is Empty function to check whether the data present in binary search tree..

Step 12: Display all those functions using main and enter proper choice and data for the proper binary search tree.

SOURCE CODE:

```
import java.lang.*;
import java.util.Stack;
import java.util.Scanner;
class BSTiter
{
static class Node {
   int key;
    Node left, right;
};
static Node newNode(int data)
{
   Node temp = new Node();
```

```
    temp.key = data;
    temp.left = null;
    temp.right = null;
    return temp;
}
static Node insert(Node root, int key)
{
    Node newnode = newNode(key);
    Node x = root;
    Node y = null;
    while (x != null) {
        y = x;
        if (key < x.key)
            x = x.left;
        else
            x = x.right;
    }
    if (y == null)
        y = newnode;
    else if (key < y.key)
        y.left = newnode;
    else
        y.right = newnode;
    return y;
}
static Node delete(Node root,int ele)
{
   Node curr=root;
   Node prev=null;
          while(curr!=null && curr.key!=ele)
          {
             prev=curr;
              if(ele<curr.key)
                  curr=curr.left;
              else
                    curr=curr.right;
          }
          if(curr==null)
          {
                    System.out.println(ele+" Element not found");
                    return root;
          }
          if(curr.left==null||curr.right==null)
          {
                    Node newCurr;
                     if (curr.left == null)
               newCurr = curr.right;
                     else
              newCurr = curr.left;
             if (prev == null)
```

```
        return newCurr;
                if (curr == prev.left)
        prev.left = newCurr;
      else
        prev.right = newCurr;
        }
        else {
      Node p = null;
      Node temp;
      temp = curr.right;
      while (temp.left != null) {
        p = temp;
        temp = temp.left;
      }
      if (p != null)
        p.left = temp.right;
      else
        curr.right = temp.right;

      curr.key = temp.key;
    }
    return root;
}
static void search(Node root,int ele)
{
        int flag=0;

        if(root==null)
        {
                System.out.println("Binary tree is empty");
                return;
        }
        Stack <Node> s=new Stack <Node>();
        Node curr=root;
        while(curr!=null||s.size()>0)
        {
                while(curr!=null)
                {
                        s.push(curr);
                        curr=curr.left;
                }
                curr=s.pop();
                if(curr.key==ele)
                {
                        flag=1;
                        System.out.println("Element "+ele+" found ");
                return;
                }
                curr=curr.right;
        }
```

```
        if(flag==0)
        System.out.println("Element not found");
}
static void Inorder(Node root)
{
        if(root==null)
        {
                return;
        }
        Stack <Node> s=new Stack <Node>();
        Node curr=root;
        while(curr!=null||s.size()>0)
        {
                while(curr!=null)
                {
                        s.push(curr);
                        curr=curr.left;
                }
                curr=s.pop();
                System.out.print(curr.key+" ");
                curr=curr.right;
        }
}
static void Preorder(Node root)
   {
      if (root == null) {
         return;
      }
      Stack<Node> nodeStack = new Stack<Node>();
      nodeStack.push(root);
      while (nodeStack.empty() == false) {
         Node mynode = nodeStack.peek();
         System.out.print(mynode.key + " ");
         nodeStack.pop();
         if (mynode.right != null) {
            nodeStack.push(mynode.right);
         }
         if (mynode.left != null) {
            nodeStack.push(mynode.left);
         }
      }
   }
static void Postorder(Node root)
 {
      Stack<Node> stack = new Stack<Node>();
      while(true) {
         while(root != null) {
            stack.push(root);
            stack.push(root);
            root = root.left;
```

```
            }
            if(stack.empty()) return;
            root = stack.pop();
            if(!stack.empty() && stack.peek() == root) root = root.right;
            else {
                System.out.print(root.key + " "); root = null;
            }
        }
    }
public boolean isEmpty(Node root)
{
  return root==null;
}
public static void main(String args[])
{
/*           50
        /   \
       30      70
       / \   / \
      20 40 60 80*/
          System.out.println("1.Insert\n2.delete\n3.Inorder traversal\n4.Preorder
Traversal\n5.Postorder traversal\n6.Search\n7.IsEmpty\n8.exit");
        Node root = null;
          char ch;
          Scanner scan=new Scanner(System.in);
          while(true)
          {
                    System.out.println("\nEnter your choice");
                    int choice = scan.nextInt();
                   switch (choice)
                   {
                             case 1:
                              System.out.println("Enter element to insert");
                               if(root==null)
                                  root=insert(root,scan.nextInt());
                               else
                                  insert(root,scan.nextInt());
                            Inorder(root);
                              break;
                            case 2:
                            System.out.println("Enter element to delete\n");
                                      root=delete(root,scan.nextInt());
                                      Inorder(root);
                                      break;
                            case 3:
                            System.out.println("Inorder traversal\n");
                                      Inorder(root);
                                      break;
                            case 4:
                            System.out.println("Preorder traversal\n");
```

```
                    Preorder(root);
                    break;
                case 5:
                System.out.println("Postorder traversal\n");
                    Postorder(root);
                    break;
                case 6:
                System.out.println("Enter element to search");
                    search(root,scan.nextInt());
                    break;

                case 7:
                boolean isempty;
                isempty=(root==null);
                System.out.println("Empty status "+isempty);
                    break;
                case 8:
                    System.exit(0);
            }
        }
    }
}
```

ACTUAL OUTPUT:


Select Command Prompt
1 2 3 4 5
Enter your choice
1
Enter element to insert
6
1 2 3 4 5 6
Enter your choice
2
Enter element to delete
3
1 2 4 5 6
Enter your choice
2
Enter element to delete
4
1 2 5 6
Enter your choice
3
Inorder traversal
1 2 5 6
Enter your choice
4
Preorder traversal
1 2 5 6
Enter your choice
5
Postorder traversal
6 5 2 1
Enter your choice
6
Enter element to search
4
Element not found
Enter your choice
6
Enter element to search


Select Command Prompt
3
Inorder traversal
1 2 5 6
Enter your choice
4
Preorder traversal
1 2 5 6
Enter your choice
5
Postorder traversal
6 5 2 1
Enter your choice
6
Enter element to search
4
Element not found
Enter your choice
6
Enter element to search
2
Element 2 found
Enter your choice
7
Empty status false
Enter your choice
8
C:\javaprograms>

Experiment -8

Write a java program to implement the following for
a)BFS graph

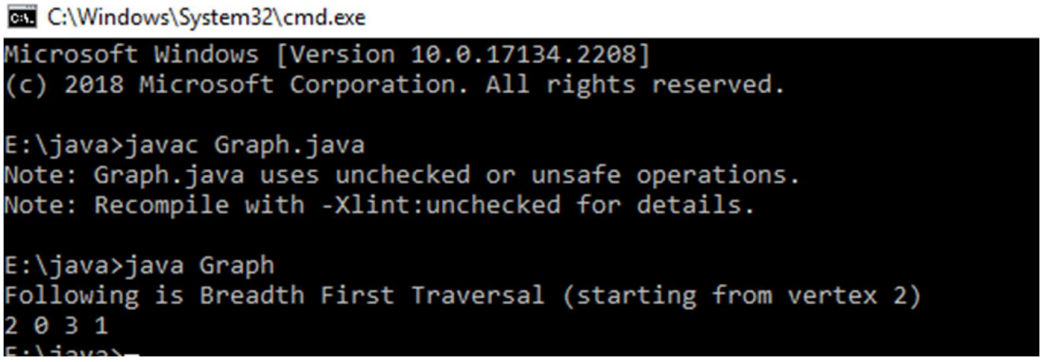
AIM:

To write a java program for implementing the BFS for a graph

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a graph class using adjacent list.
Step 4: Define addEdge to add an edge into the graph.
Step 5: Define BFS for printing BFS traversal from a given source s initially mark all the vertices as not visited (by default set as false).
Step 6: Create a queue for BFS and mark the current node as visited and enqueue it (insert), dequeue it when the size of the queue is not equal to zero.
Step 7: Get all adjacent vertices of the dequeued vertex s , if a adjacent has not been visited, then mark it visited and enqueue it.
Step 8: Import all these methods to main method and traverse through one level of children nodes, then traverse through the level of grandchildren nodes.
Step 9: Stop

EXPECTED OUTPUT:

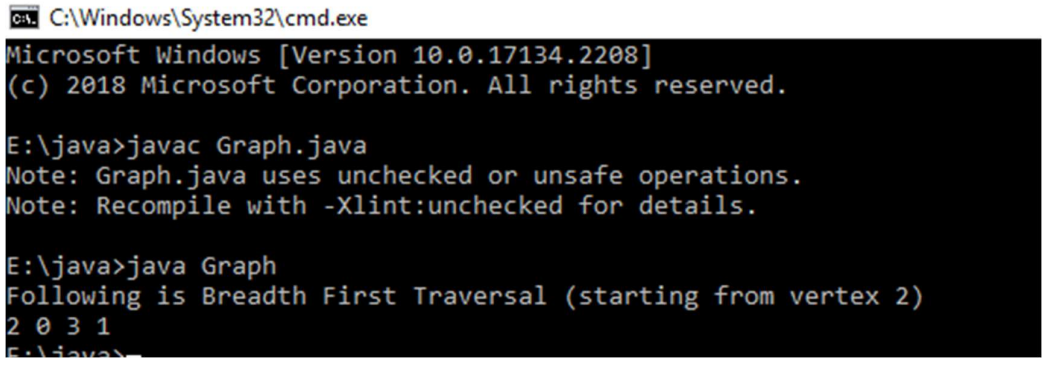


SOURCE CODE:

```
import java.io.*;
import java.util.*;
class Graph
{
   private int V;
   private LinkedList<Integer> adj[];
   Graph(int v)
   {
      V = v;
      adj = new LinkedList[v];
      for (int i=0; i<v; ++i)
         adj[i] = new LinkedList();
   }
   void addEdge(int v,int w)
   {
      adj[v].add(w);
   }
```

```
    void BFS(int s) {
        boolean visited[] = new boolean[V];
        LinkedList<Integer> queue = new LinkedList<Integer>();
        visited[s]=true;
        queue.add(s);
        while (queue.size() != 0)
        {
            s = queue.poll();
            System.out.print(s+" ");
            Iterator<Integer> i = adj[s].listIterator();
            while (i.hasNext())
            {
                int n = i.next();
                if (!visited[n])
                {
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
    }
    public static void main(String args[])  {
        Graph g = new Graph(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);
        System.out.println("Following is Breadth First Traversal "+ "(starting from vertex 2)");
          g.BFS(2);
    }
}
```

ACTUAL OUTPUT:



8b) DFS

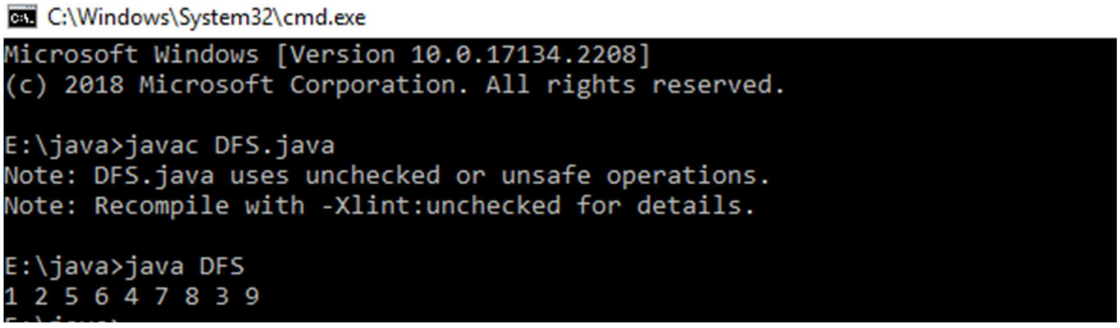
AIM:

To write a Java program for implementing the DFS for a graph

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a class DFS and initialize V (no. of vertices) as we are building graph using adjacency list so we should have linked list for every that list node and store adjacent nodes in that list.
Step 4: Create a constructor for class DFS and create an empty list for every node.
Step 5: Define addEdgesToGraph function for adding edges to graph, and define add function in it for adding vertex to adjacency list of another vertex so that edge can be added to graph.
Step 6: Define DFTraversal and traverse one strong component completely.
Step 7: Define DFSearch and call DFTraverial on disconnected components, it will keep track of visited [ ] array.
Step 8: Define all these functions in the main method and call DFSearch. It will keep track of visited[ ] array.
Step 9: Stop.

EXPECTED OUTPUT:

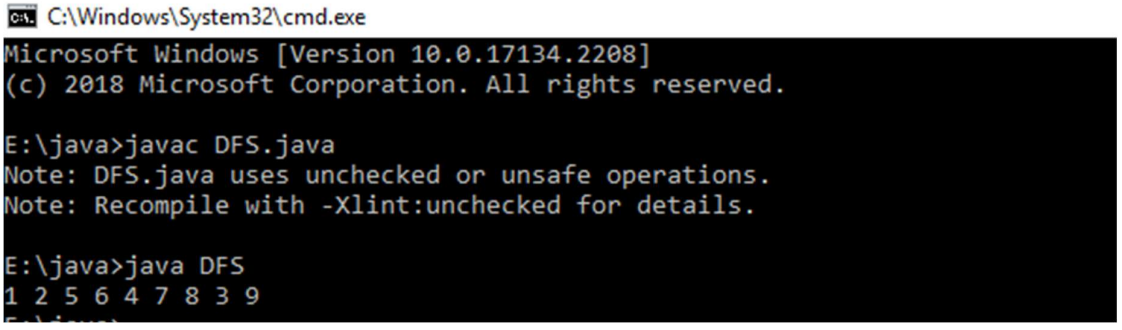


SOURCE CODE:

```
import java.io.*;
import java.util.*;
public class DFS{
   int V
   LinkedList<Integer> adjList[];
   DFS(int v) {
      V = v;
      adjList = new LinkedList[v];
      for (int i=0; i<v; ++i){
         adjList[i] = new LinkedList();
       }
   }
   void addEdgesToGraph(int v, int u){
      adjList[v].add(u);
   }
   void DFTraversal(int v,int visited[])  {
      visited[v] = 1;
```

```
System.out.print(v + " ");
        Iterator<Integer> i = adjList[v].listIterator();
        while (i.hasNext()) {
            int n = i.next();
            if (visited[n]==0)
                DFTraversal(n, visited);
        }
    }
    void DFSearch(int v)
    {
        int visited[] = new int[V];
        DFTraversal(v, visited);
                        for (int i=1;i<V;i++)
                        {
                                if(visited[i]==0)
                                {
                            DFTraversal(i, visited);
                                }
                        }
    }
    public static void main(String args[])
    {
        DFS obj = new DFS(10);
        obj.addEdgesToGraph(1,2);
        obj.addEdgesToGraph(1,4);
        obj.addEdgesToGraph(2,5);
        obj.addEdgesToGraph(2,6);
        obj.addEdgesToGraph(4,7);
        obj.addEdgesToGraph(4,8);
        obj.addEdgesToGraph(3,9);
        obj.addEdgesToGraph(3,4);
        obj.addEdgesToGraph(4,3);
        obj.DFSearch(1);
    }
}
```

ACTUAL OUTPUT:



Experiment -10

Write a Java program to implement Merge & Heap sort of given elements

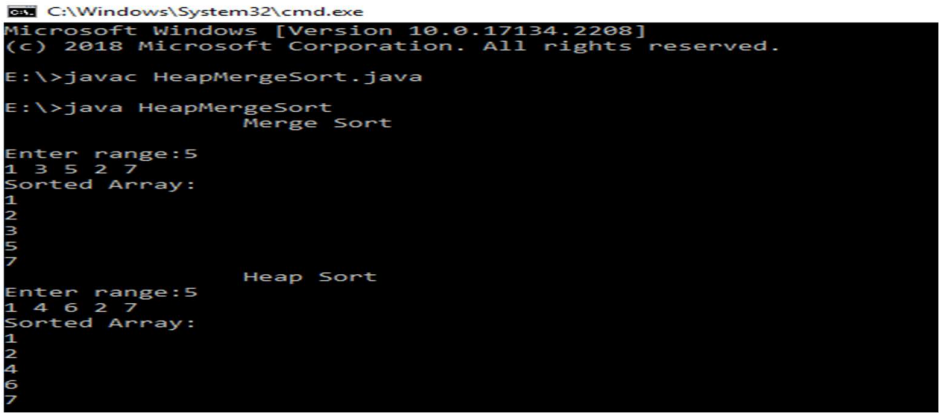
AIM:

To implement Merge and Heap sort of given elements by using java program

ALGORITHM:

Step 1: Start.
Step 2: Declare required header files.
Step 3: Create a class HeapMergeSort and create an obj for it and by considering input from the user, sort the elements using mergesort and heapsort functions
Step 4: Define merge function and declare required variables beg,mid and end and array are [ ] and merge every elements after sorting the elements.
Step 5: Define mergeSort function and if the begining element is less than the end element, make mid as the avg of both the elements and recursively call the function to sort the elements.
Step 6: Define heapSort function and initially consider the length of the array and build heap (rearrange array) by extracting one by one in element from heap taken in array and move current root to end ,call max heapity on the reduced heap. To heapify a subtree rooted with node i which is an index in arr[ ] n is size of heap.
Step 7: Define heapify function and initialize largest as roots if left child is larger than root.
Step 8: If right child larger than largest make r as largest.
Step 9: If largest is not root, recursively heapify the affected subtree.
Step10 : Stop.

EXPECTED OUTPUT:



SOURCE CODE:

```
import java.util.*;
class HeapMergeSort {
   public static void main(String[] args) {
      HeapMergeSort h = new HeapMergeSort();
      Scanner s = new Scanner(System.in);
      System.out.println("\t\tMerge Sort\n");
      int n;
      System.out.print("Enter range:");
      n = s.nextInt();
      int[] arr = new int[n];
      for (int i = 0; i < n; i++) {
         arr[i] = s.nextInt();
      }
```

```
        h.mergesort(arr, 0, n - 1);
        System.out.println("Sorted Array:");
        for (int j = 0; j < arr.length; j++) {
            System.out.println(arr[j]);
        }
        System.out.println("\t\tHeap Sort");
        System.out.print("Enter range:");
        n = s.nextInt();
        int[] arr1 = new int[n];
        for (int i = 0; i < n; i++) {
            arr1[i] = s.nextInt();
        }
        h.heapsort(arr1);
        System.out.println("Sorted Array:");
    for (int j = 0; j < arr1.length; j++) {
            System.out.println(arr1[j]);
        }
        s.close();
     }
     void merge(int arr[], int beg, int mid, int end) {
        int l = mid - beg + 1;
        int r = end - mid;
        int LeftArray[] = new int[l];
        int RightArray[] = new int[r];
        for (int i = 0; i < l; ++i)
            LeftArray[i] = arr[beg + i];
        for (int j = 0; j < r; ++j)
            RightArray[j] = arr[mid + 1 + j];
        int i = 0, j = 0;
        int k = beg;
        while (i < l && j < r) {
            if (LeftArray[i] <= RightArray[j]) {
                arr[k] = LeftArray[i];
                i++;
            } else {
                arr[k] = RightArray[j];
                j++;
            }
            k++;
        }
        while (i < l) {
            arr[k] = LeftArray[i];
            i++;
            k++;
        }
        while (j < r) {
            arr[k] = RightArray[j];
            j++;
            k++;
        }
```

```
    }
    void mergesort(int arr[], int beg, int end) {
        if (beg < end) {
            int mid = (beg + end) / 2;
            mergesort(arr, beg, mid);
            mergesort(arr, mid + 1, end);
            merge(arr, beg, mid, end);
        }
    }
    public void heapsort(int arr[]) {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--)
          heapify(arr, n, i);
        for (int i = n - 1; i >= 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            heapify(arr, i, 0);
        }
    }
    void heapify(int arr[], int n, int i) {
        int largest = i; // Initialize largest as root
        int l = 2 * i + 1; // left = 2*i + 1
        int r = 2 * i + 2; // right = 2*i + 2
        if (l < n && arr[l] > arr[largest])
            largest = l;
        if (r < n && arr[r] > arr[largest])
            largest = r;
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }
}
```

ACTUAL OUTPUT:

```
E:\>javac HeapMergeSort.java

E:\>java HeapMergeSort
                    Merge Sort

Enter range:5
1 3 5 2 7
Sorted Array:
1
2
3
5
7
                    Heap Sort
Enter range:5
1 4 6 2 7
Sorted Array:
1
2
4
6
7
```

Experiment -11

Write a Java program to implement Quick Sort of given elements.

AIM:

Program to implement Quick Sort of given elements using to Java

ALGORITHM:

Step 1: Start.

Step 2: Declare required header files.

Step 3: Create a class QuickSortLinkedList and define required functions.

Step 4: Create a node structure and declare data and next pointer.

Step 5: Create a constructor for node and Initialize using this pointer.

Step 6: Define addNode function and by considering elements from the user, if the head is equal to null, initialize new elements for it and define print list to print all the elements. by considering from first and last node but do not break any links in the whole linked list.

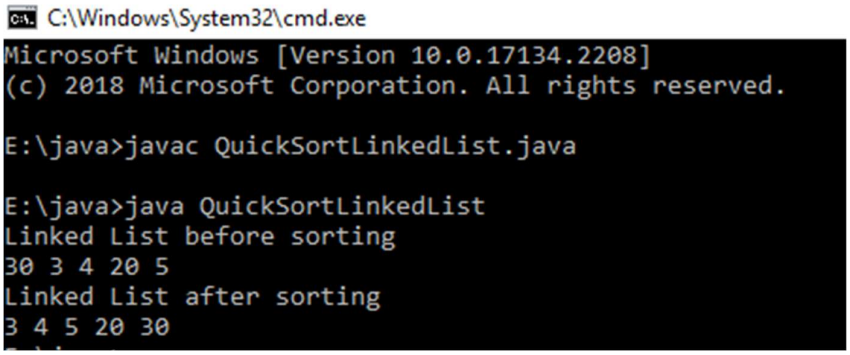
Step 7: Define partionList and iterate till one before the end, no need to iterate till the end because end is pivot keep track of last modified item. Swap the position of current ie. next suitable Index and pivot, and return one previous to current because current is now pointing to pivot.

Step 8: Define sort and split list and partion recurse ,if pivot is picked and moved to the start, that means start and pivot is same, so pick from next of pivot. It pivot is in between of the list, start from next, of pivot, since we have pivot, prev so we move two nodes.

Step 9: Display the sorted elements using main method and by Considering before sorted and after sorted data.

Step10: Stop.

EXPECTED OUTPUT:



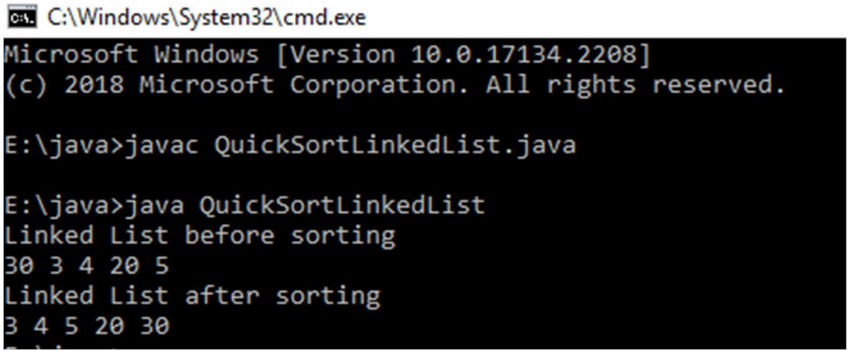
SOURCE CODE:

```
public
class QuickSortLinkedList {
        static class Node {
                int data;
                Node next;
                Node(int d)
                {
                        this.data = d;
```

```
                this.next = null;
        }
}
Node head;
void addNode(int data)
{
        if (head == null) {
                head = new Node(data);
                return;
        }

        Node curr = head;
        while (curr.next != null)
                curr = curr.next;
        Node newNode = new Node(data);
        curr.next = newNode;
}
void printList(Node n)
{
        while (n != null) {
                System.out.print(n.data);
                System.out.print(" ");
                n = n.next;
        }
}
Node paritionLast(Node start, Node end)
{
        if (start == end || start == null || end == null)
                return start;
        Node pivot_prev = start;
        Node curr = start;
        int pivot = end.data;
        while (start != end) {
                if (start.data < pivot) {
                        pivot_prev = curr;
                        int temp = curr.data;
                        curr.data = start.data;
                        start.data = temp;
                        curr = curr.next;
                }
                start = start.next;
        }
        int temp = curr.data;
        curr.data = pivot;
        end.data = temp;
        return pivot_prev;
}
void sort(Node start, Node end)
{
        if(start == null || start == end|| start == end.next )
```

```
                return;
            Node pivot_prev = paritionLast(start, end);
            sort(start, pivot_prev);
            if (pivot_prev != null && pivot_prev == start)
                sort(pivot_prev.next, end);
            else if (pivot_prev != null&& pivot_prev.next != null)
                sort(pivot_prev.next.next, end);
        }
    public static void main(String[] args)
        {
            QuickSortLinkedList list= new QuickSortLinkedList();
            list.addNode(30);
            list.addNode(3);
            list.addNode(4);
            list.addNode(20);
            list.addNode(5);
            Node n = list.head;
            while (n.next != null)
                n = n.next;
            System.out.println("Linked List before sorting");
            list.printList(list.head);
            list.sort(list.head, n);
            System.out.println("\nLinked List after sorting");
            list.printList(list.head);
        }
}
```

ACTUAL OUTPUT:



Experiment -12

Write a java program to implement various operations on AVL trees

AIM:

program to implement various operations on AVL trees

ALGORITHM:

Step 1: Start.

Step 2: Declare required header files.

Step 3: Define class node to create a node which is empty. Declare required variables (data and height), node left and right.

Step 4: Define constructor for class Node and assign initially left and right as null and data, height as zero, and also Initialize data with n in another constructor.

Step 5: Create a class AVLTree 1 and initialize root as null.
Define height function if it, hull return -1 else return the height of the tree using t variable.

Step 6: Define insert function and insert the data by and assign it to root initially.

Step 7: Define insert function with variables and t node, if node t is null, then t is the new node with x in it.else if the date x is less than data in t, then insert the new node of data in left side of t, then if the height of left.Height of right is -2 then if the given data is less than the left data then it is LL(t), else if the is greater than root t node then insert, now node in right side of the t node, then if the height of left - height of right is 2, then if x is greater than data in right mode then t = RR(t) or else RL(t). T height is max height of left & t height of right in +1 and return t.

Step 8: Create node constructor LL. with node k2 as variable. Declare node k1 as k2. left and k2 left is k1and k1 right is k2.

Step 9: Define RR constructor, as k2 is k1 right and right of k1 left of k2 left it k1. height of k is max height of k1 left .

Step10: Define count_nodes, function and return the count node with root to get no. of nodes in the tree.

Step11: Define getbalance to check if the tree is balanced or If it is null, returns 0, or return height of left free - height of right tree.

Step12: Define Node find min to find the minimum element and traverse it to the last hode of left subtree and return the value.

Step13: Define count_nodes with node r and traverse every location of left and right subtree and store the value in l and return l.

Step14: Define inorder function and print left, root and right element preorder function to print root, left and right elements, postorder function to print left, right and root element.

Step15: Define search function of node r and val. If the given element than the root, search it in the left subtree or if the element is greater than the root, traverse every location. than in right subtree until the search element is found.

Step 16: Define delete function and if root is null, return null else if the given element is less than root data traveve to let subtree and delete root of let tree, else if the given element is greater than right subtree, delete root of right subtree else if both roots are

not null.

Step17: If left is null return right it right. I null return left if root is null, when temp is will & the root is temp. Calculate height and declare b at and 11 equal to node,
if bal >1 and getbalance of left <0 then returm L1 (root),
if bal >1 and get bal of left <o, then return LR (root),
if bal <-1 and getbal of right >0, then return RR. (root),
if bal-1 and getbal of right so, then return RL(root). And return root.

Step18: Stop.

SOURCE CODE:

```
import java.util.*;
class Node{
Node left,right;
int data;
int height;
public Node()
{
left=right=null;
data=0;
height=0;
}
public Node(int n)
{
left=right=null;
data=n;
height=0;
}
}
class AVLTree1
{
Node root;
public AVLTree1()
{
root=null;
}
int height(Node t)
{
if(t==null)
return -1;
else
return t.height;
}
public void insert(int data)
{
```

```
root=insert(data,root);
}
private Node insert(int x, Node t)
{
if(t==null)
t=new Node(x);
else if(x<t.data)
{
t.left=insert(x,t.left);
if(height(t.left)-height(t.right)==-2)
{
if(x<t.left.data)
t=LL(t);
else
t=LR(t);
}
}
else if(x>t.data)
{
t.right=insert(x,t.right);
if(height(t.left)-height(t.right)==2)
{
if(x>t.right.data)
t=RR(t);
else
t=RL(t);
}
}
t.height=Math.max(height(t.left),height(t.right))+1;
return t;
}
private Node LL(Node k2)
{
Node k1=k2.left;
k2.left=k1.right;
k1.right=k2;
k2.height=Math.max(height(k2.left),height(k2.right))+1;
k1.height=Math.max(height(k1.left),k2.height)+1;
return k1;
}
private Node RR(Node k1)
{
Node k2=k1.right;
k1.right=k2.left;
```

```
k2.left=k1;
k1.height=Math.max(height(k1.left),height(k1.right))+1;
k2.height=Math.max(height(k2.right), k1.height)+1;
return k2;
}
public Node LR(Node k2)
{
k2.left=RR(k2.left);
return LL(k2);
}
private Node RL(Node k1)
{
k1.right=LL(k1.right);
return RR(k1);
}
public int count_nodes()
{
return count_nodes(root);
}
public int getbalance(Node t)
{
if(t==null)
return 0;
return height(t.left) - height(t.right);
}
Node findmin(Node t)
{
Node temp=t;
while(temp.left!=null)
temp=temp.left;
return temp;
}
private int count_nodes(Node r)
{
int l;
if(r==null)
return 0;
else
{
l=1;
l+=count_nodes(r.left);
l+=count_nodes(r.right);
return l;
}
```

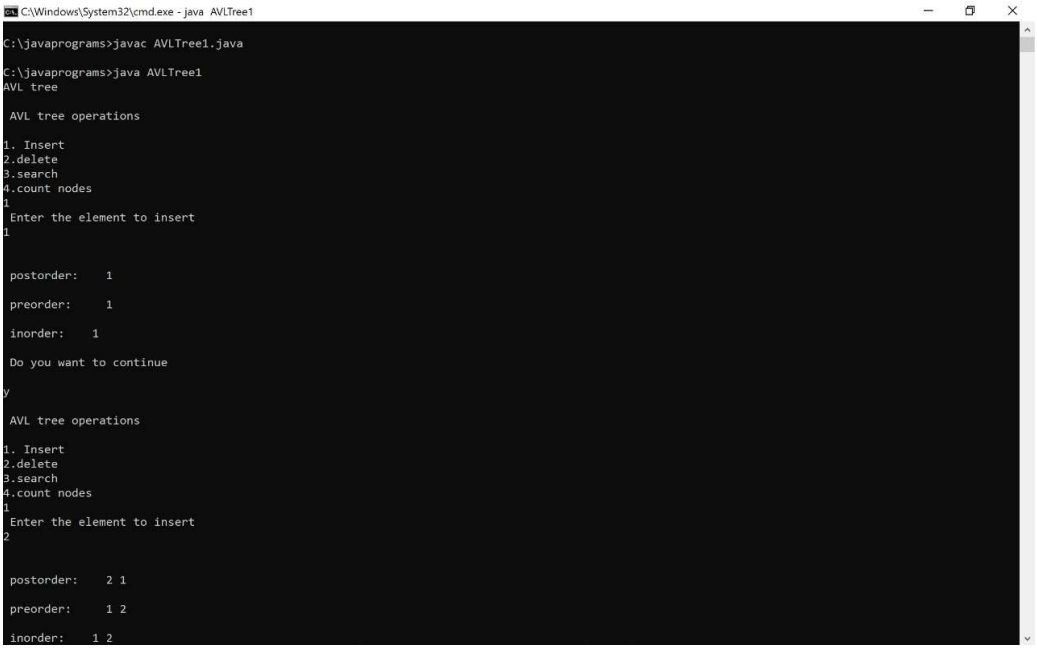
```
}
public void inorder(Node x)
{
if(x!=null)
{
inorder(x.left);
System.out.print(x.data+" ");
inorder(x.right);
}
}
public void postorder()
{
postorder(root);
}
public void postorder(Node x)
{
if(x!=null)
{
postorder(x.left);
postorder(x.right);
System.out.print(x.data+" ");
}
}
public void preorder(Node x)
{
if(x!=null)
{
System.out.print(x.data+" ");
preorder(x.left);
preorder(x.right);
}
}
public boolean search(Node r, int val)
{
boolean found=false;
while((r!=null) &&(!found))
{
int rval;
rval=r.data;
if(val < rval)
r=r.left;
else if(val > rval)
r=r.right;
else
```

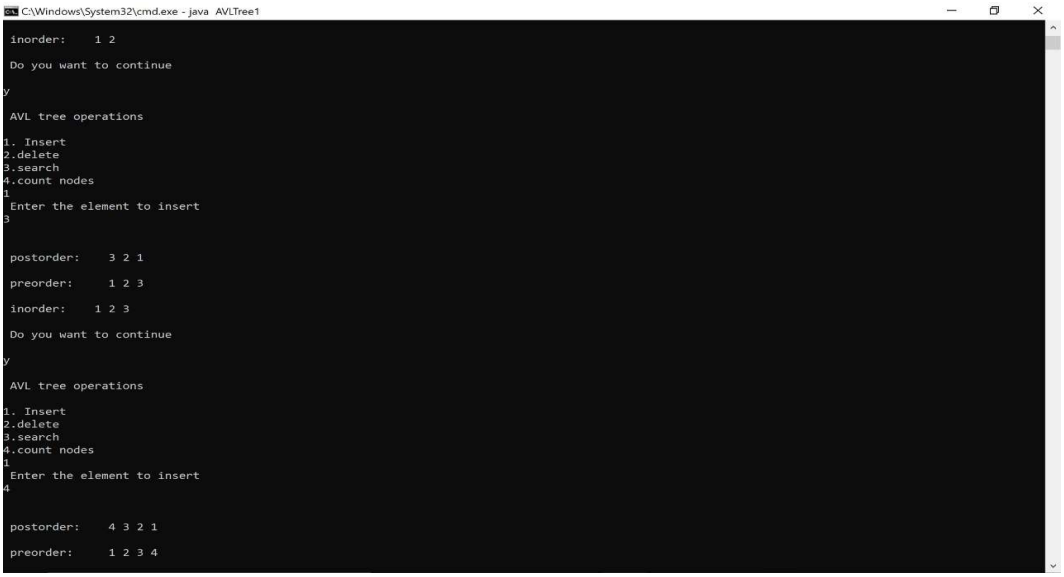
```
{
found=true;
break;
}
found=search( r, val );
}
return found;
}
public Node delete(int x, Node root)
{
Node temp;
if(root==null)
return null;
else if(x < root.data)
root.left=delete(x, root.left);
else if(x > root.data)
root.right=delete(x, root.right);
else if((root.left!=null)&&(root.right!=null))
        {
        temp=findmin(root.right);
        root.data=temp.data;
        root.right=delete(root.data, root.right);
        }
else
{
temp=root;
if(root.left==null)
temp=root.right;
else if(root.right==null)
temp=root.left;
if(temp==null)
{
temp=root;
root=null;
}
else
{
root=temp;
}
}
if(root==null)
return root;
root.height=1+Math.max(height(root.left),height(root.right));
int bal=getbalance(root);
```

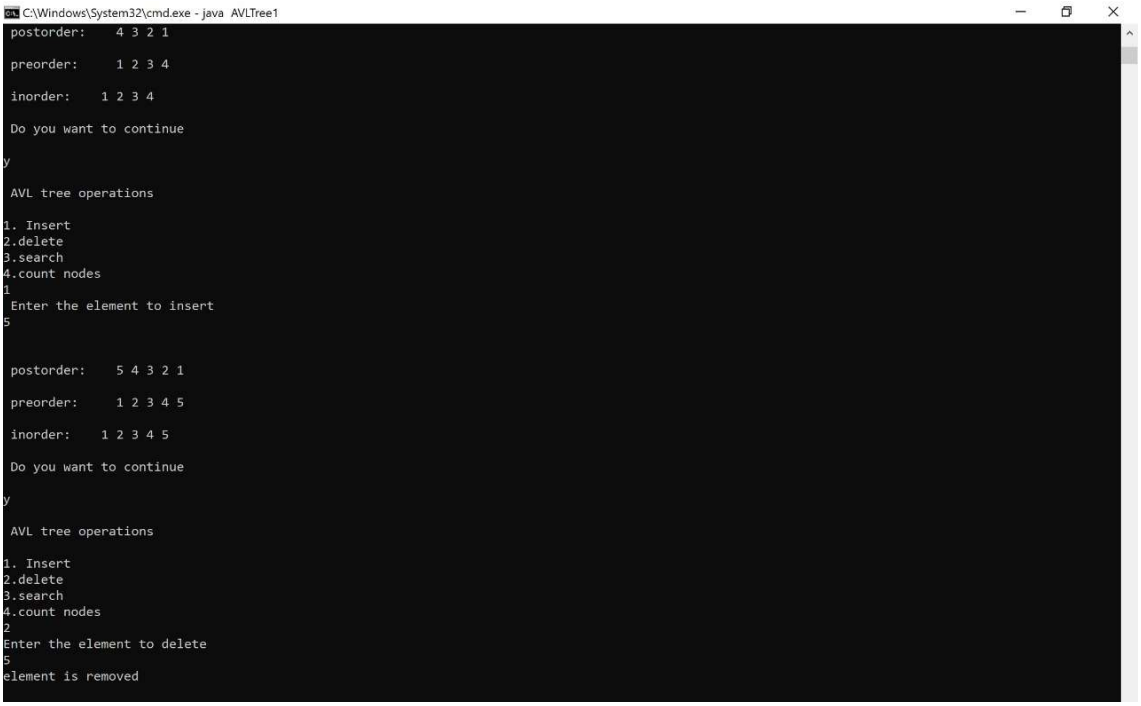
```
if(bal>1 &&getbalance(root.left)>=0)
return LL(root);
if(bal>1 && getbalance(root.left)<0)
return LR(root);
if(bal <-1 && getbalance(root.right)<=0)
return RR(root);
if(bal<-1 &&getbalance(root.right)>0)
return RL(root);
return root;
}
public static void main(String[] args)
{
Node h;
Scanner sc=new Scanner(System.in);
AVLTree1 a=new AVLTree1();
System.out.println("AVL tree");
char ch1;
int ch;
boolean re;
do
{
System.out.println("\n AVL tree operations\n");
System.out.println("1. Insert");
System.out.println("2.delete");
System.out.println("3.search ");
 System.out.println("4.count nodes ");
ch=sc.nextInt();
switch(ch){
case 1:
        System.out.println(" Enter the element to insert");
        a.insert(sc.nextInt());
        break;
case 2: if(a.root==null)
        System.out.println("AVL Tree is empty");
        else{
        System.out.println("Enter the element to delete");
        a.root=a.delete(sc.nextInt(),a.root);
        System.out.println("element is removed");
        }
        break;
case 3:System.out.println(" enter element to search");
        re=a.search(a.root,sc.nextInt());
        if(re==false)
```

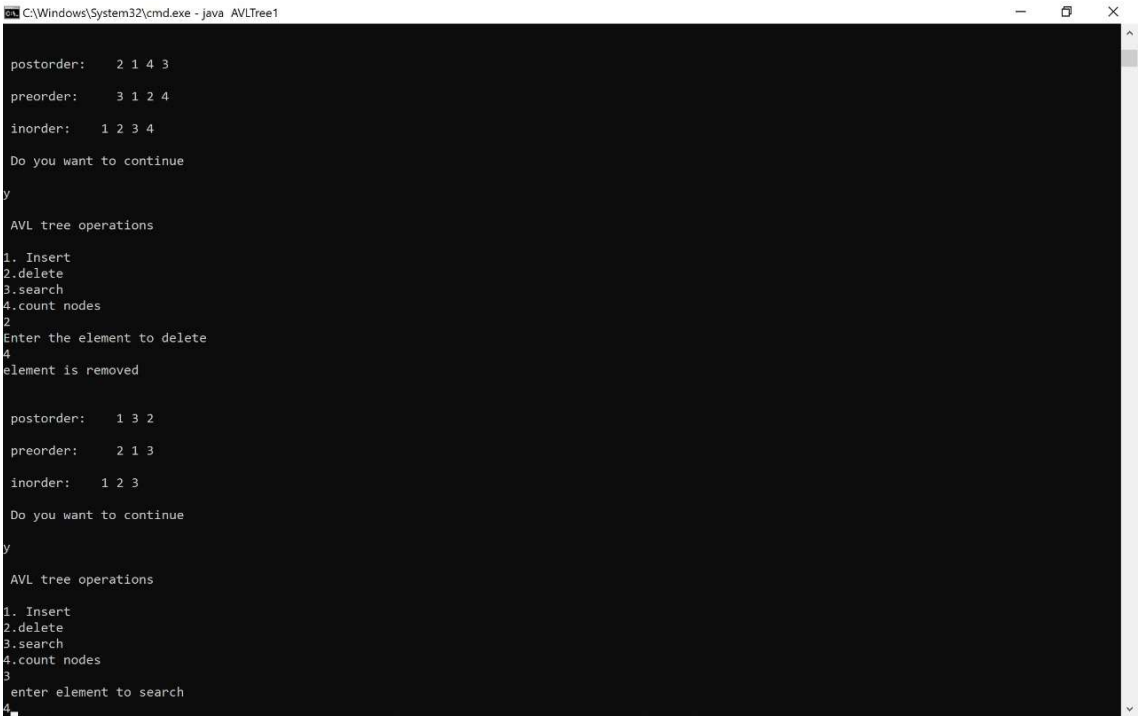
```
            System.out.println("element not found");
            else
            System.out.println("element found");
            break;
case 4: System.out.println("Nodes: "+a.count_nodes());
            break;
            default:
            System.out.println("Wrong entry\n");
            break;
}
System.out.println(" ");
if(a.root!=null){
System.out.print(" \n postorder:    ");
a.postorder(a.root);
System.out.println("");
System.out.print(" \n preorder:     ");
a.preorder(a.root);
System.out.println("");
System.out.print(" \n inorder:    ");
a.inorder(a.root);
}
System.out.println("");
System.out.println(" \n Do you want to continue\n");
ch1=sc.next().charAt(0);
}while(ch1=='y' || ch1=='Y');
}
}
```

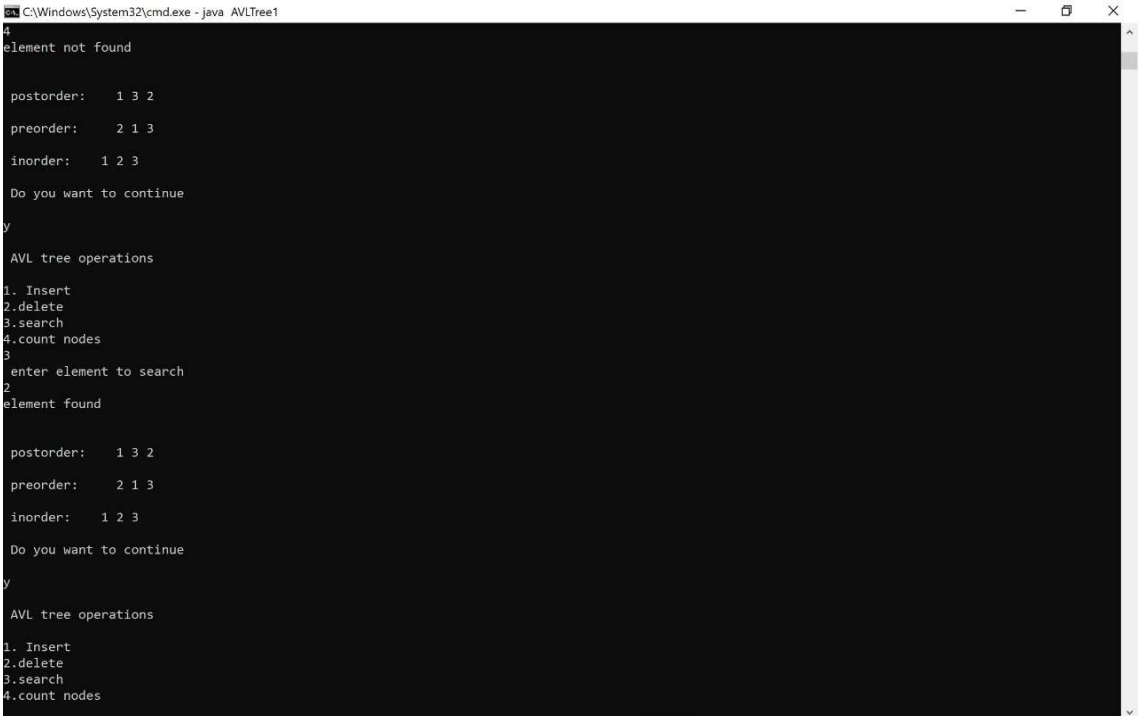
ACTUAL OUTPUT:


C:\Windows\System32\cmd.exe - java AVLTree1
C:\javaprograms>javac AVLTree1.java
C:\javaprograms>java AVLTree1
AVL tree
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
1
Enter the element to insert
1
postorder: 1
preorder: 1
inorder: 1
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
1
Enter the element to insert
2
postorder: 2 1
preorder: 1 2
inorder: 1 2


C:\Windows\System32\cmd.exe - java AVLTree1
inorder: 1 2
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
1
Enter the element to insert
3
postorder: 3 2 1
preorder: 1 2 3
inorder: 1 2 3
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
1
Enter the element to insert
4
postorder: 4 3 2 1
preorder: 1 2 3 4


C:\Windows\System32\cmd.exe - java AVLTree1
postorder: 4 3 2 1
preorder: 1 2 3 4
inorder: 1 2 3 4
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
1
Enter the element to insert
5
postorder: 5 4 3 2 1
preorder: 1 2 3 4 5
inorder: 1 2 3 4 5
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
2
Enter the element to delete
5
element is removed


C:\Windows\System32\cmd.exe - java AVLTree1
postorder: 2 1 4 3
preorder: 3 1 2 4
inorder: 1 2 3 4
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
2
Enter the element to delete
4
element is removed
postorder: 1 3 2
preorder: 2 1 3
inorder: 1 2 3
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
3
enter element to search
4


C:\Windows\System32\cmd.exe - java AVLTree1
4
element not found
postorder: 1 3 2
preorder: 2 1 3
inorder: 1 2 3
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
3
enter element to search
2
element found
postorder: 1 3 2
preorder: 2 1 3
inorder: 1 2 3
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes


C:\Windows\System32\cmd.exe
4.count nodes
3
enter element to search
2
element found
postorder: 1 3 2
preorder: 2 1 3
inorder: 1 2 3
Do you want to continue
y
AVL tree operations
1. Insert
2.delete
3.search
4.count nodes
4
Nodes: 3
postorder: 1 3 2
preorder: 2 1 3
inorder: 1 2 3
Do you want to continue
n

Experiment-13

Write a java program to perform the following operations:

a)Insertion into a B-tree b) Searching in a B-tree

AIM:

Program to implement various operations on a) Insertion intoa B-bree b) Search in a B-tree.

ALGORITHM:

Step 1: Start.

Step 2: Declare required header files.

Step 3: Create a class Btree and declare required functions.

Step 4: Create a class Node and declare key, child and intialize leaf as true.

Step 5: Define insert function and assign in as leaf node. Then we find where k belongs in the array of keys, shift everything over to the left, and stick k in there. If x is not a leaf node, we can't just stick k in because it doesn't have any children, children are really only created when we split node. so we don't get an unbalanced child of where we can insert k.

Step 6: Define search function and perform a binary search on the records in the current node. If a record with the search key is found, then return that record. If the current node is a leaf node and the key is not found, then report an unsuccessful search. Otherwise, follow the proper branch and repeat the process.

Step 7: Define display function by creating a variable and point to the start and traverse it using for loop until it points to null and print value at each iteration.

Step 8: Stop.

SOURCE CODE:

```
import java.util.Stack;
import java.util.*;
public class BTree {
private int T;
  public class Node {
    int n;
    int key[] = new int[2 * T - 1];
    Node child[] = new Node[2 * T];
    boolean leaf = true;
    public int Find(int k) {
      for (int i = 0; i < this.n; i++) {
        if (this.key[i] == k) {
          return i;
        }
      }
      return -1;
    }
  }
  public BTree(int t) {
```

```
  T = t;
  root = new Node();
  root.n = 0;
  root.leaf = true;
 }
static private Node root;
 private Node Search(Node x, int key) {
          int flag=0;
  int i = 0;
  if (x == null)
   return x;
  for (i = 0; i < x.n; i++) {
   if (key < x.key[i]) {
    break;
   }
   if (key == x.key[i]) {
        flag=1;
        System.out.println("Element found");
    return x;
   }
  }
        if(flag==0)
        System.out.println("Element not found");
  if (x.leaf) {
   return null;
  } else {
   return Search(x.child[i], key);
  }
 }
 private void Split(Node x, int pos, Node y) {
  Node z = new Node();
  z.leaf = y.leaf;
  z.n = T - 1;
  for (int j = 0; j < T - 1; j++) {
   z.key[j] = y.key[j + T];
  }
  if (!y.leaf) {
   for (int j = 0; j < T; j++) {
    z.child[j] = y.child[j + T];
   }
  }
  y.n = T - 1;
  for (int j = x.n; j >= pos + 1; j--) {
   x.child[j + 1] = x.child[j];
```

```
  }
  x.child[pos + 1] = z;
  for (int j = x.n - 1; j >= pos; j--) {
    x.key[j + 1] = x.key[j];
  }
  x.key[pos] = y.key[T - 1];
  x.n = x.n + 1;
}
public void Insert(final int key) {
  Node r = root;
  if (r.n == 2 * T - 1) {
    Node s = new Node();
    root = s;
    s.leaf = false;
    s.n = 0;
    s.child[0] = r;
    Split(s, 0, r);
    _Insert(s, key);
  } else {
    _Insert(r, key);
  }
}
final private void _Insert(Node x, int k) {
  if (x.leaf) {
    int i = 0;
    for (i = x.n - 1; i >= 0 && k < x.key[i]; i--) {
      x.key[i + 1] = x.key[i];
    }
    x.key[i + 1] = k;
    x.n = x.n + 1;
  } else {
    int i = 0;
    for (i = x.n - 1; i >= 0 && k < x.key[i]; i--) {
    }
    i++;
    Node tmp = x.child[i];
    if (tmp.n == 2 * T - 1) {
      Split(x, i, tmp);
      if (k > x.key[i]) {
        i++;
      }
    }
    _Insert(x.child[i], k);
  }
```

```
  }
  private void display(Node x) {
    assert (x == null);
    for (int i = 0; i < x.n; i++) {
      System.out.print("  "+x.key[i]);
    }
    if (!x.leaf) {
      for (int i = 0; i < x.n + 1; i++) {
        display(x.child[i]);
      }
    } }
  public static void main(String[] args) {
    BTree b = new BTree(3);
int c, ch1;
char ch;
Scanner sc= new Scanner(System.in);
while(true) {
System.out.println("\n Menu");
System.out.println(" 1. Insert");
System.out.println(" 2.display");
System.out.println("3.search");
System.out.println("4.exit");
System.out.println("enter the choice");
ch1=sc.nextInt();
switch(ch1){
case 1: System.out.println(" enter the element to insert");
          b.Insert(sc.nextInt());
          break;
case 2: System.out.println(" Elements in the B-Tree:");
          b.display(root);
          break;
case 3: System.out.println("Enter element to search");
          b.Search(root,sc.nextInt());
          break;
case 4: System.exit(0);
break;
}
}
}
}
```

ACTUAL OUTPUT:

```
E:\20021D0503>javac BTree.java

E:\20021D0503>java BTree

 Menu
 1. Insert
 2.display
3.search
4.exit
enter the choice
1
 enter the element to insert
1

 Menu
 1. Insert
 2.display
3.search
4.exit
enter the choice
1
 enter the element to insert
3

 Menu
 1. Insert
 2.display
3.search
4.exit
enter the choice
1
 enter the element to insert
5

 Menu
 1. Insert
 2.display
3.search
4.exit
enter the choice
2
 Elements in the B-Tree:
  1  3  5
 Menu
 1. Insert
 2.display
3.search
4.exit
enter the choice
3
Enter element to search
3
Element found
```

```
 Menu
 1. Insert
 2.display
3.search
4.exit
enter the choice
3
Enter element to search
9
Element not found

 Menu
 1. Insert
 2.display
3.search
4.exit
enter the choice
4
```

Experiment-14

Write a java program to implementation of recursive and non-recursive functions to Binary tree Traversals

AIM:

Java program to implement recursive and non-recursive functions to Binary tree traversals

ALGORITHM:

Step 1: Start.

Step 2: Declare required header files.

Step 3: Create a structure of name Node and declare structure variable and create a constructor for Node and initialize key as item, left and right as null.

Step 4: Create a class nr_r_ btt and declare required functions.

Step 5: Display all the functions of recursive and non-recursive inside the main.

Step 6: Define rpostorder function and recursively call function for assigning left, right and then key. by initially checking it the node is null (empty).

Step 7: Define rinorder function and recursively call function by assigning left, key and then right by initially checking t the node is null (empty).

Step 8: Define rpreorder function and recursively call function by assigning key , left and then right by initially checking the node is null (empty).

Step 9: Define inorder function and by reading the elements, assign the elements in the order of key left, key and then right by checking if the current is null, if true then check the current left position is null, if it true. Initialize assign the left element and then insert the key and insert the right element.

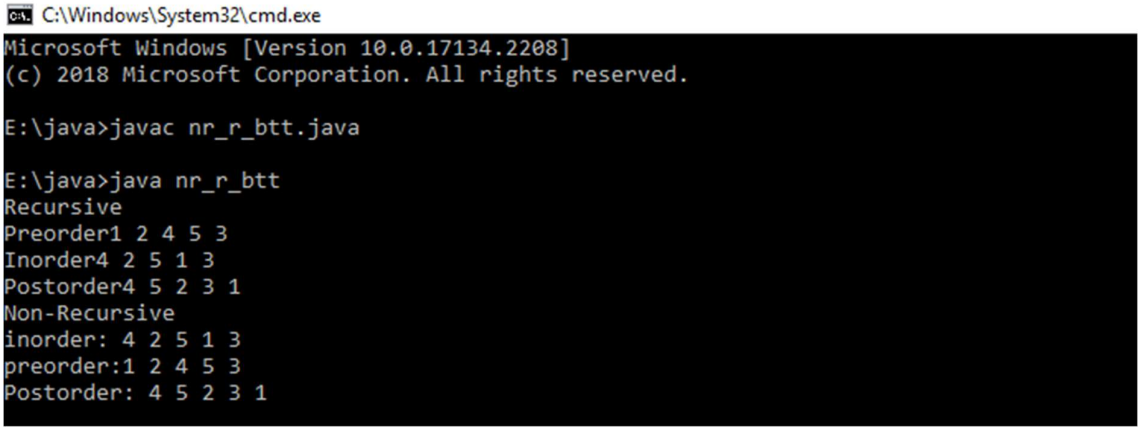
Step10: Define postorder function and by reading the elements, assign the elements in the order of left, right and key by initializing temp to root and if it is null, assign the temp as left and if right is not equal to null; and right is not visited, assign next value to right and next value as key.

Step11: Define preorder function and by reading the elements, assign the elements in the order of key, left and right, by initially checking if the mode is not equal to null, it it is true, assign the node as key, and then traverse to next position insert at left and then right.

Step12: Define an obj for class and call all the functions to display the output using recursive and non-recursive functions.

Step 13: Stop.

EXPECTED OUTPUT:



SOURCE CODE:

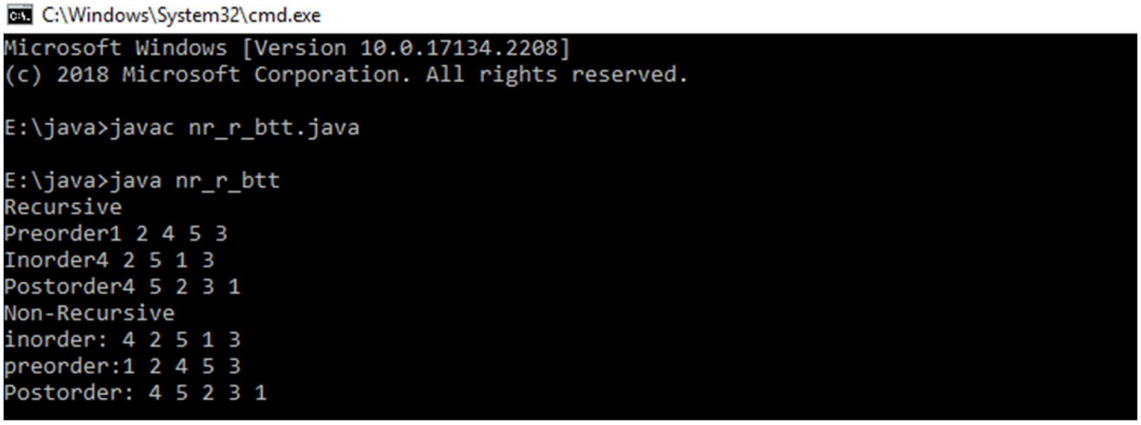
```
import java.util.*;
class Node{
int key;
Node left,right;
boolean visited;
public Node(int item)
{
key=item;
left=right=null;
}
}
class nr_r_btt
{
Node root;
nr_r_btt()
{
root=null;
}
public Node getdata(int a)
{
Node n=new Node(a);
return n;
}
void  rpostorder(Node node)
{
if(node==null)
return;
rpostorder(node.left);
rpostorder(node.right);
System.out.print(node.key+" ");
```

```
}
void  rinorder(Node node)
{
if(node==null)
return;
rinorder(node.left);
System.out.print(node.key+" ");
rinorder(node.right);
}
void  rpreorder(Node node)
{
if(node==null)
return;
System.out.print(node.key+" ");
rpreorder(node.left);
rpreorder(node.right);
}
void inorder(Node ptr)
{
Node curr,pre;
if(root==null)
return;
curr=root;
while(curr!=null){
if(curr.left==null){
System.out.print(curr.key+" ");
curr=curr.right;
}
else{
pre=curr.left;
while((pre.right!=null) &&(pre.right!=curr))
pre=pre.right;
if(pre.right==null){
pre.right=curr;
curr=curr.left;
}
else{
pre.right=null;
System.out.print(curr.key+" ");
curr=curr.right;
}
}
}
System.out.println(" ");
```

```
}
void postorder(Node root){
Node temp=root;
while(temp!=null && temp.visited==false){
if((temp.left!=null)&&(temp.left.visited==false))
temp=temp.left;
else if(temp.right !=null && temp.right.visited==false)
temp=temp.right;
else{
System.out.print(" "+temp.key);
temp.visited=true;
temp=root;
}
}
System.out.println(" ");
}
void preorder(Node node){
while(node!=null){
if(node.left==null){
System.out.print(node.key+" ");
node=node.right;
}
else{
Node curr=node.left;
while((curr.right!=null) && (curr.right!=node)){
curr=curr.right;
}
if(curr.right==node){
curr.right=null;
node=node.right;
}
else
{
System.out.print(node.key+" ");
curr.right=node;
node=node.left;
}
}
}
System.out.println(" ");
}
public static void main(String[] args){
nr_r_btt t= new nr_r_btt();
t.root=t.getdata(1);
```

```
t.root.left=t.getdata(2);
t.root.right=t.getdata(3);
t.root.left.left=t.getdata(4);
t.root.left.right=t.getdata(5);
System.out.println("Recursive");
System.out.print("Preorder");
t.rpreorder(t.root);
System.out.println(" ");
System.out.print("Inorder");
t.rinorder(t.root);
System.out.println(" ");
System.out.print("Postorder");
t.rpostorder(t.root);
System.out.println(" ");
System.out.println("Non-Recursive");
System.out.print("inorder: ");
t.inorder(t.root);
System.out.print("preorder:");
t.preorder(t.root);
System.out.print("Postorder:");
t.postorder(t.root);
}
}
```

ACTUAL OUTPUT:



Experiment-15

Write a java program to implement all the functions of Dictionary (ADT) using Hashing.

AIM:

Java program to implement all the functions of Dictionary using hashing.

ALGORITHM:

Step 1: Start.

Step 2: Declare required header files.

Step 3: Define a class dict_hast and create an obj for It.insert elements in the hash table which was created by Dictionary.

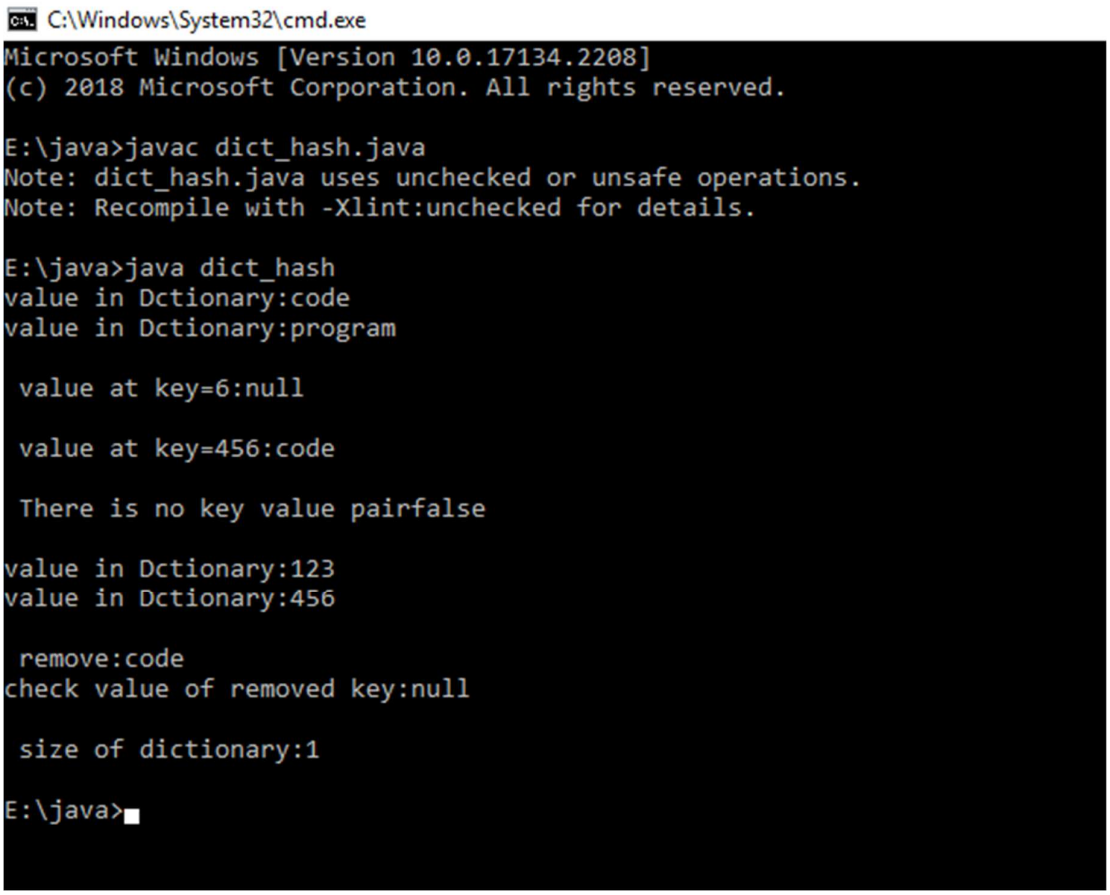
Step 4: The inserted element should be in the order of key-values pairs.

Step 5: Print all the keys by considering the input which was already given. And also check for empty status. If there is no key-value pair, display on message empty is true or false.

Step 6: Print the values of keys by considering the input which was already given. Remove the keys by calling remove function and check value of the removed key. And also check empty status and size of the dictionary after calling all functions.

Step 7: Stop.

EXPECTED OUTPUT:



SOURCE CODE:

```
import java.util.*;
public class dict_hash{
public static void main(String[] args){
Dictionary d= new Hashtable();
d.put("123","code");
d.put("456","program");
for(Enumeration i=d.elements(); i.hasMoreElements(); )
{
System.out.println("value in Dctionary:" + i.nextElement());
}
System.out.println("\n value at key=6:"+d.get("6"));
System.out.println("\n value at key=456:"+d.get("123"));
System.out.println("\n There is no key value pair"+d.isEmpty()+"\n");
for(Enumeration k=d.keys(); k.hasMoreElements();){
System.out.println("value in Dctionary:"+k.nextElement());
}
System.out.println("\n remove:"+d.remove("123"));
System.out.println("check value of removed key:"+d.get("123"));
System.out.println("\n size of dictionary:"+d.size());
}
}
```

ACTUAL OUTPUT:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17134.2208]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\java>javac dict_hash.java
Note: dict_hash.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

E:\java>java dict_hash
value in Dctionary:code
value in Dctionary:program

 value at key=6:null

 value at key=456:code

 There is no key value pairfalse

value in Dctionary:123
value in Dctionary:456

 remove:code
check value of removed key:null

 size of dictionary:1
```