*HARSH KASHYAP  
CSE 4*

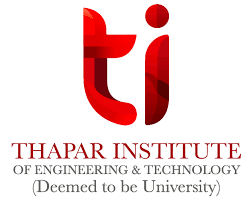
*101917088*

[*hkashyap\_be19@thapar.edu*](mailto:hkashyap_be19@thapar.edu)

A Practical activity Report submitted

for Data Structures (UCS301)

**DATA STRUCTURES**

****

Computer Science and Engineering

Patiala Campus

**2020**

Submitted to

Maninder Kaur

**Assignment 2**

A class to show basic operations, LinkedList. The classes for the questions will inherit the properties and function of this class

//BASIC OPERATIONS OF LINKED LIST

import java.util.\*;

import java.lang.\*;

// class for performing LinkedList operation.

class LinkedList

{

static Scanner scr = new Scanner (System.in);

Node head;

static class Node

{

Node next;

int data;

Node(int d)

{

data =d;

next =null;

}

}

// function to insert at the beginning taking input LinkedList and the val to be inserted and

// returning the LinkedList or the object class created

public static LinkedList insertAtBeginning(LinkedList list, int data)

{

//creating anew node to store the particular value;

Node new\_Node = new Node(data);

//assigning list the head position

new\_Node.next=list.head;

list.head=new\_Node;

return list;

}

// function to insert at the end taking input LinkedList and the val to be inserted and

// returning the LinkedList or the object class created

public static LinkedList insertAtLast(LinkedList list, int data)

{

Node new\_Node=new Node(data);

new\_Node.next=null;

if (list.head==null)

{

list.head = new\_Node;

}

else

{

Node last = list.head;

while(last.next!=null)

{

last = last.next;

}

last.next = new\_Node;

}

return list;

}

//fn to insert before or after a particular value

//Linkedlist in which the operation is to be taken place

//check which stores the node before or after which value is to be inserted

//data to store the value of new node

//index which is either 0 or any other number to insert new node before or after a particular node.

public static LinkedList insertAfterBefore(LinkedList list, int check,int data,int ind)

{

Node new\_Node=new Node(data);

new\_Node.next=null;

Node prev =null;

Node curr=list.head;

boolean ans= false;

if (list.head==null)

{

System.out.println("No element found, didn't insert ");

}

else if (list.head.data==check && ind!=0)

{

ans=true;

return insertAtBeginning(list,data);

}

else

{

while(curr.next!=null)

{

if (curr.data==check)

{

ans=true;

if (ind==0)

{

new\_Node.next= curr.next;

curr.next=new\_Node;

System.out.println(data+" added after Node "+curr.data);

}

else

{

prev.next= new\_Node;

new\_Node.next=curr;

System.out.println(data+" added before Node "+curr.data);

}

break;

}

prev=curr;

curr=curr.next;

}

}

if(!ans)

System.out.println("Node "+check+" not found, didn't insert. ");

return list;

}

//Printing by traversing

public static void print(LinkedList list)

{

Node current = list.head;

System.out.println("\n\tDisplaying the List ");

while (current!=null)

{

if (current.next!=null)

System.out.print(current.data+" -> ");

else

System.out.println(current.data);

current= current.next;

}

}

//fn to delete the value at the end

public static String deleteAtBeginning(LinkedList l1)

{

if (l1.head==null)

return null;

else

{

int val = (l1.head).data;

l1.head= l1.head.next;

return Integer.toString(val);

}

}

//function to delete value at the end

public static String deleteAtEnd(LinkedList l1)

{

if (l1.head==null)

return null;

else

{

Node prev=null,curr =l1.head;

while(curr.next!=null)

{

prev=curr;

curr=curr.next;

}

int val=curr.data;

if (curr==l1.head)

l1.head=null;

else

prev.next= null;

return Integer.toString(val);

}

}

//Delete an element from key passing LinkedList and value

public static LinkedList deleteAtKey(LinkedList list, int key)

{

Node current = list.head;

Node prev = null;

if (current!= null && current.data==key)

{

list.head = current.next;

System.out.println(key +" found and deleted ,was present at the very beginning ");

return list;

}

while (current!=null && current.data!=key)

{

prev= current;

current = current.next;

}

if (current!=null)

{

prev.next = current.next;

System.out.println(key +" found and deleted . ");

}

else if (current==null)

{

System.out.println(key+" not found . ");

}

return list;

}

//function to search a particular value in the linkedlist

public static void search(LinkedList list, int val)

{

Node current = list.head;

int counter=0;

//traversal

while (current!=null)

{

if (current.data == val)

{

System.out.println(val +" is present at index "+counter);

return;

}

current = current.next;

counter++;

}

System.out.println(val+" not found in list. ");

}

}

**Question 1**

**Develop a menu-driven program for the following operations of on a Singly Linked List.**

**(a) Insertion at the beginning.**

**(b) Insertion at the end.**

**(c) Insertion in between (before or after a node having a specific value, say 'Insert a new Node 35 before/after the Node 30').**

**(d) Deletion from the beginning.**

**(e) Deletion from the end.**

**(f) Deletion of a specific node, say 'Delete Node 60').**

**(g) Search for a node and display its position from the head.**

**(h) Display all the node values.**

**SOLUTION CODE**

**/\***

**Develop a menu-driven program for the following operations of on a Singly Linked**

**List.**

**(a) Insertion at the beginning.**

**(b) Insertion at the end.**

**(c) Insertion in between (before or after a node having a specific value, say 'Insert a**

**new Node 35 before/after the Node 30').**

**(d) Deletion from the beginning.**

**(e) Deletion from the end.**

**(f) Deletion of a specific node, say ('Delete Node 60').**

**(g) Search for a node and display its position from the head.**

**(h) Display all the node values.**

**\*/**

**import java.util.\*;**

**// child class which can use basic operation of parent class**

**//TO USE THIS PLEASE CALL THE CLASS LinkedList TOO**

**class a2ques1 extends LinkedList**

**{**

**static Scanner scr= new Scanner(System.in);**

**static int menu()**

**{**

**System.out.println( );**

**System.out.println( "--------——LINKED LIST----------");**

**System.out.println( "\t——MENU——- \n1.Insertion at the beginning.\n2.Insertion at the end.\n3.Insertion in between ");**

**System.out.println( "4.Deletion from the beginning.\n5.Deletion from the end. \n6.Deletion of a specific node");**

**System.out.println( "7.Search for a node and display its position from head.\n8.Display all node values.\n9.Exit \nEnter your choice ");**

**int ch = scr.nextInt();**

**return ch;**

**}**

**public static void main(String[] args)**

**{**

**LinkedList list = new LinkedList();**

**int val;**

**int ch=0;**

**do {**

**ch = menu();**

**System.out.println();**

**switch (ch)**

**{**

**case 1:**

**System.out.print("Enter value to be inserted at the beginning : ");**

**val =scr.nextInt();**

**insertAtBeginning(list, val);**

**break;**

**case 2:**

**System.out.print("Enter value to be inserted at the end : ");**

**val =scr.nextInt();**

**insertAtLast(list, val);**

**break;**

**case 3:**

**System.out.print("Enter value to be inserted : ");**

**val =scr.nextInt();**

**System.out.print(val+" to be inserted around which node : ");**

**int no =scr.nextInt();**

**System.out.println(val+" to be inserted before/ after "+no+".\nType 0 for inserting after "+no+". Any other number for inserting before ");**

**int cross =scr.nextInt();**

**insertAfterBefore(list, no, val, cross);**

**break;**

**case 4:**

**System.out.print("Deleting item form beginning :" +deleteAtBeginning(list));**

**break;**

**case 5:**

**System.out.print("Deleting item form end :" +deleteAtEnd(list));**

**break;**

**case 6:**

**System.out.print("Enter value to be deleted :");**

**val =scr.nextInt();**

**deleteAtKey(list, val);**

**break;**

**case 7:**

**System.out.print("Enter value to be searched :");**

**val =scr.nextInt();**

**search(list, val);**

**break;**

**case 8:**

**print(list);**

**break;**

**case 9:**

**System.out.println("We are done ");**

**break;**

**default:**

**System.out.println("Not an option \n");**

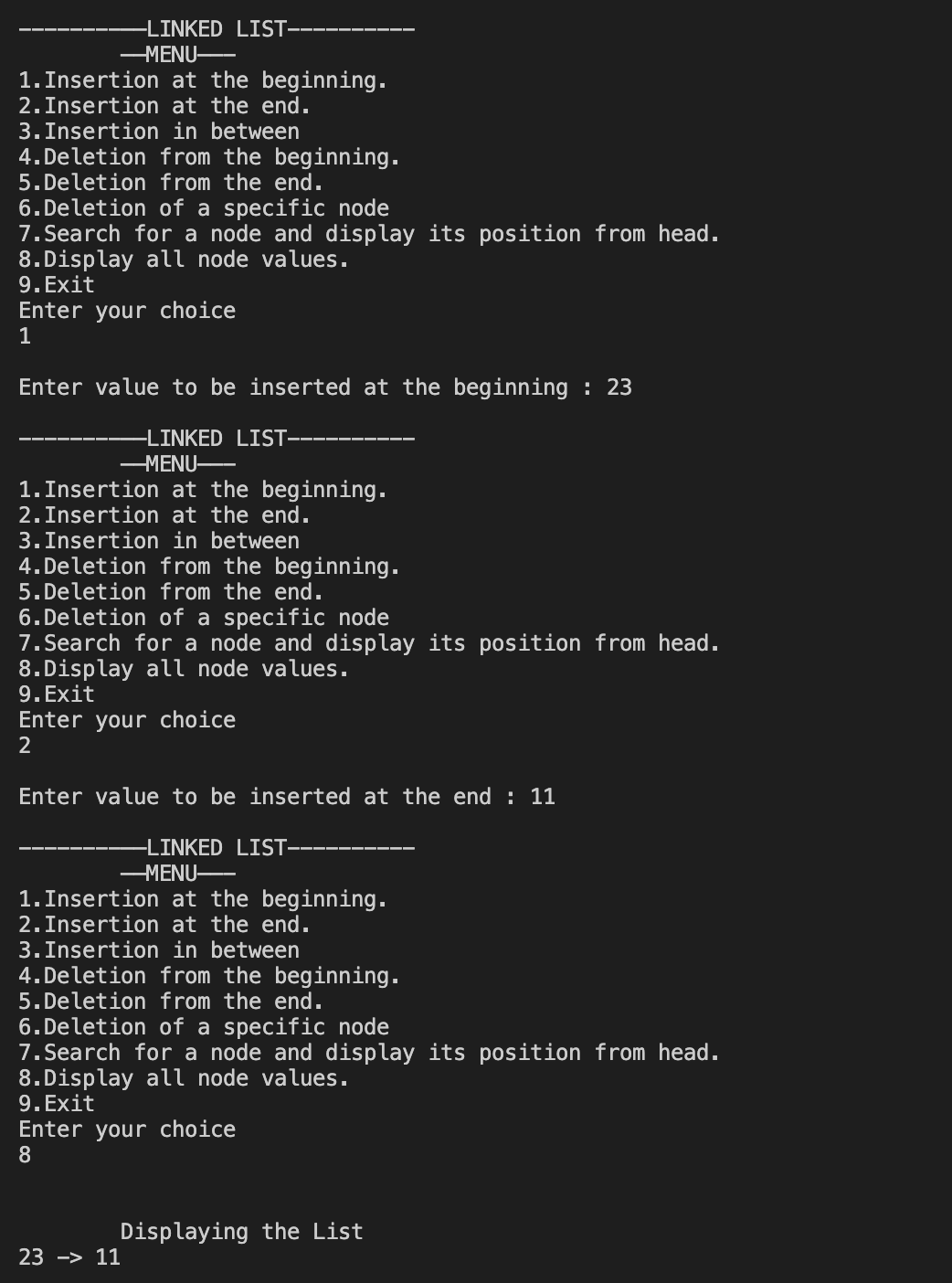
**}**

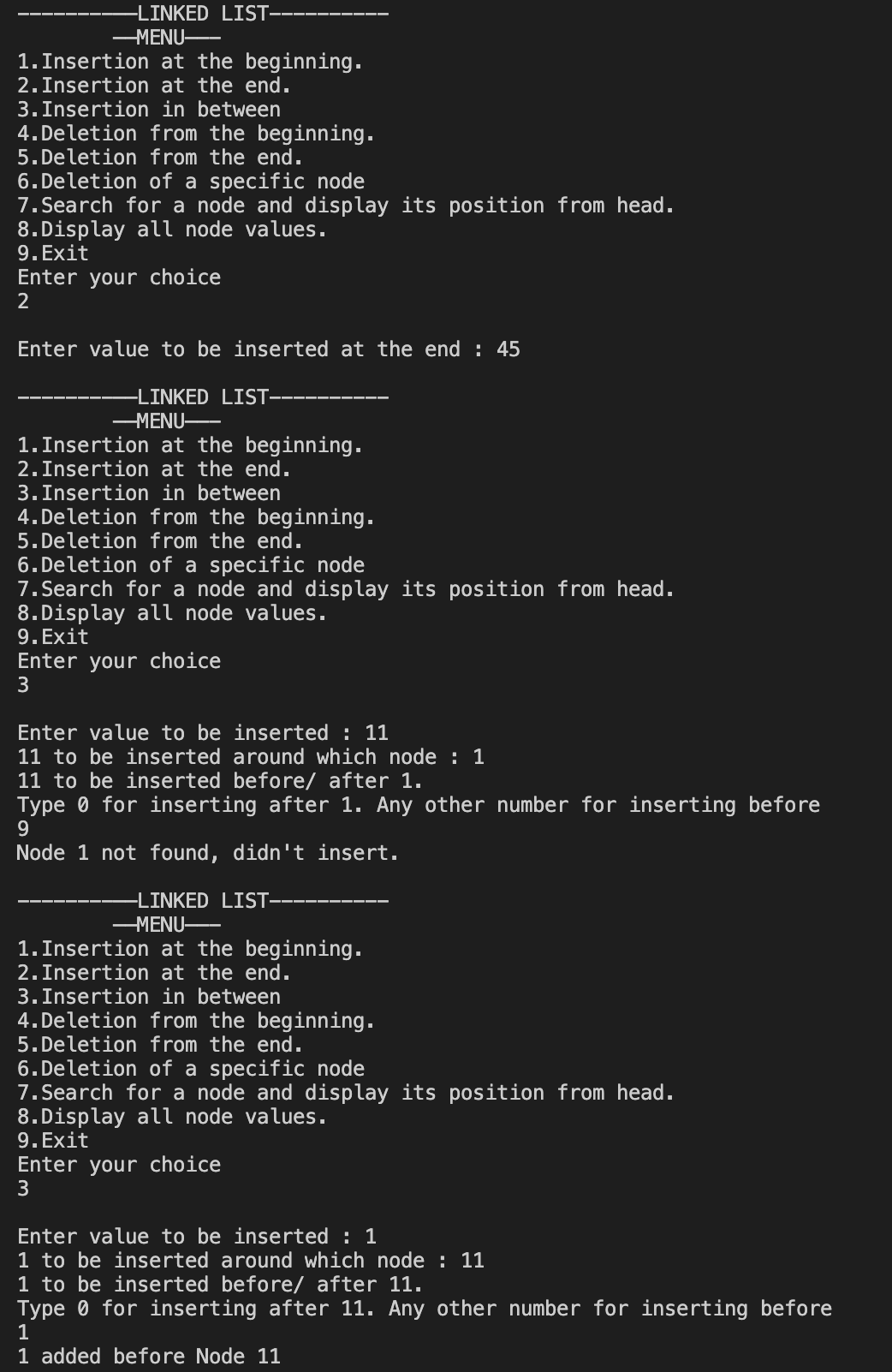
**} while (ch != 9);**

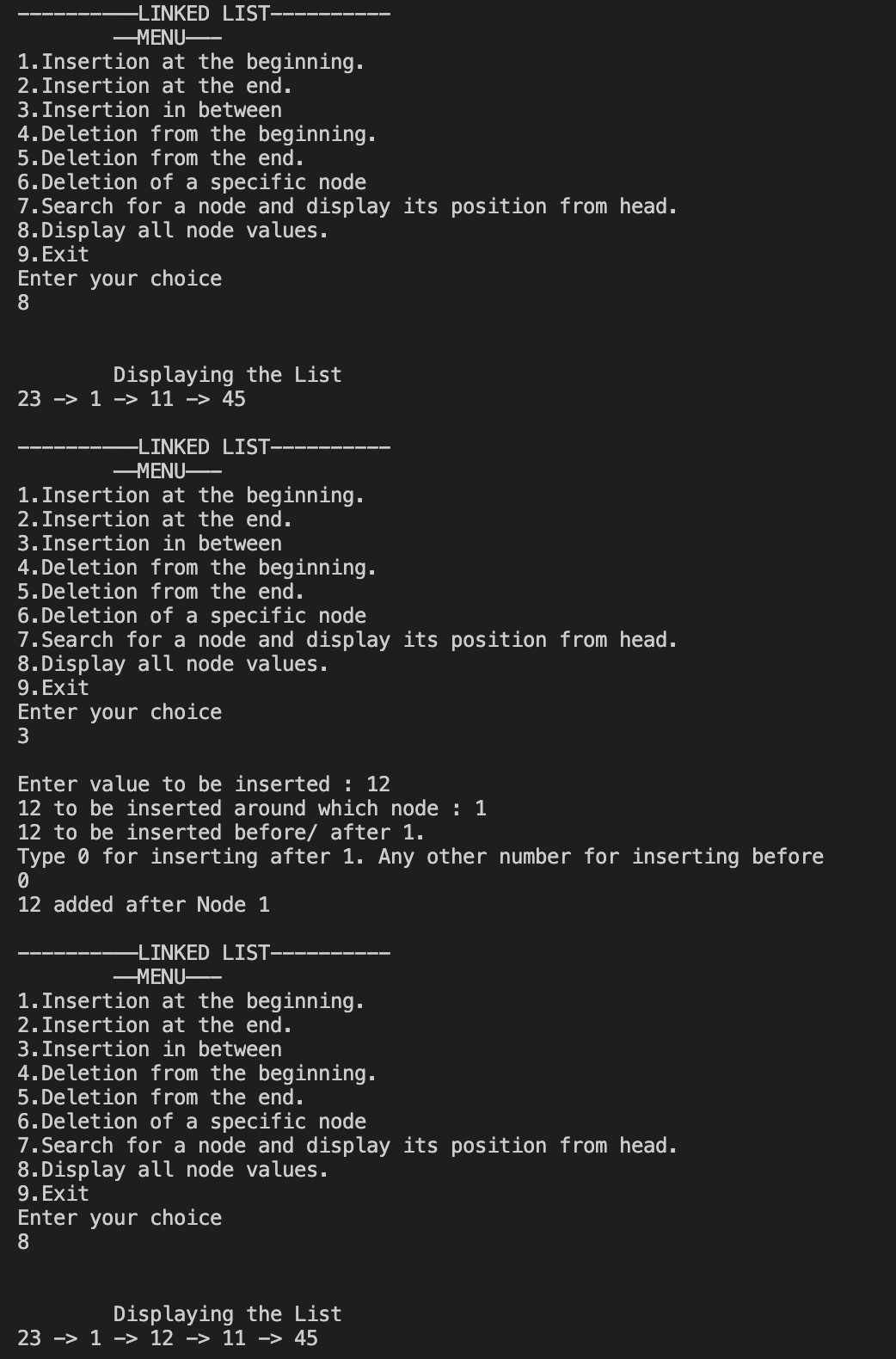
**}**

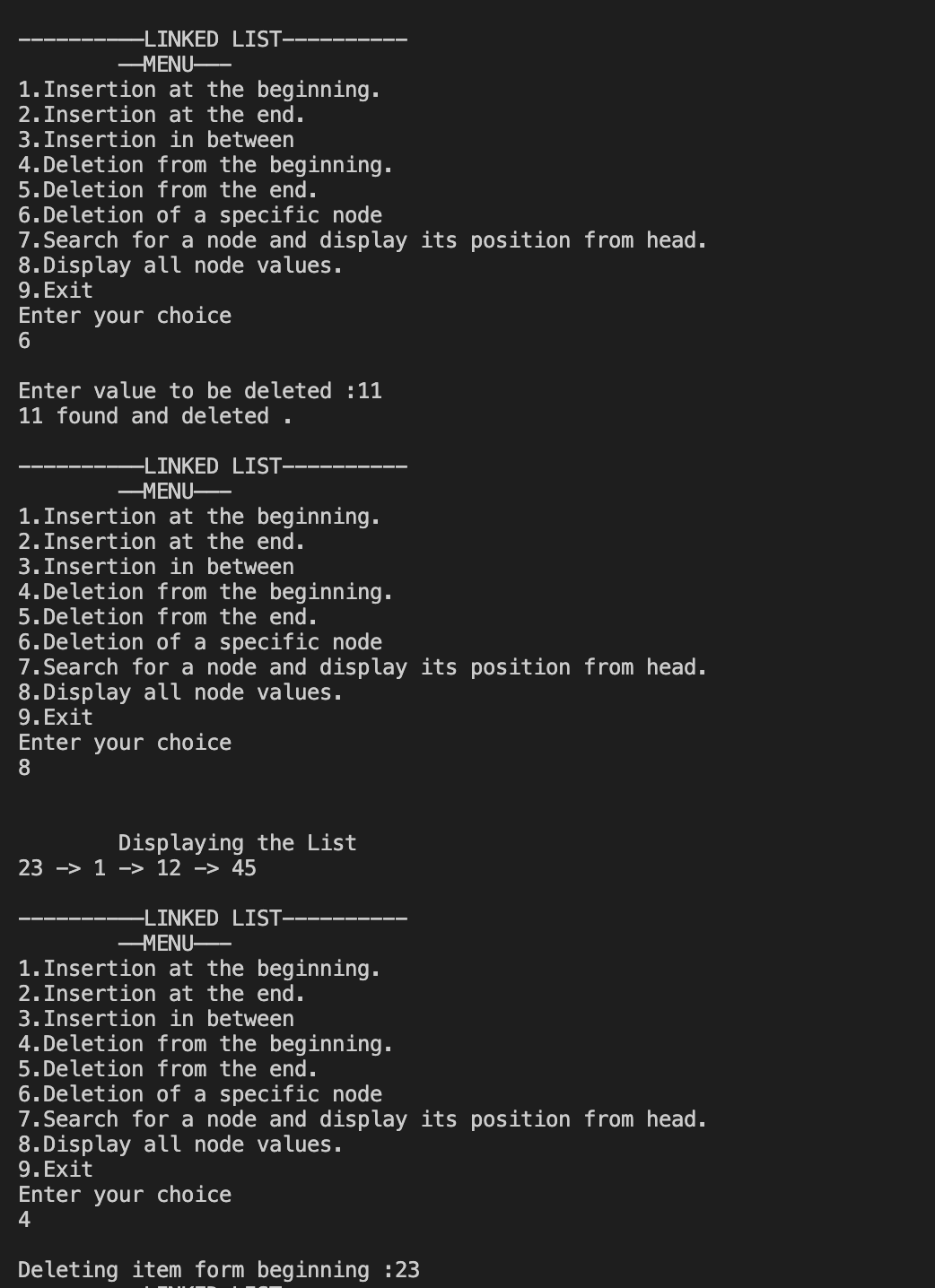
**}**

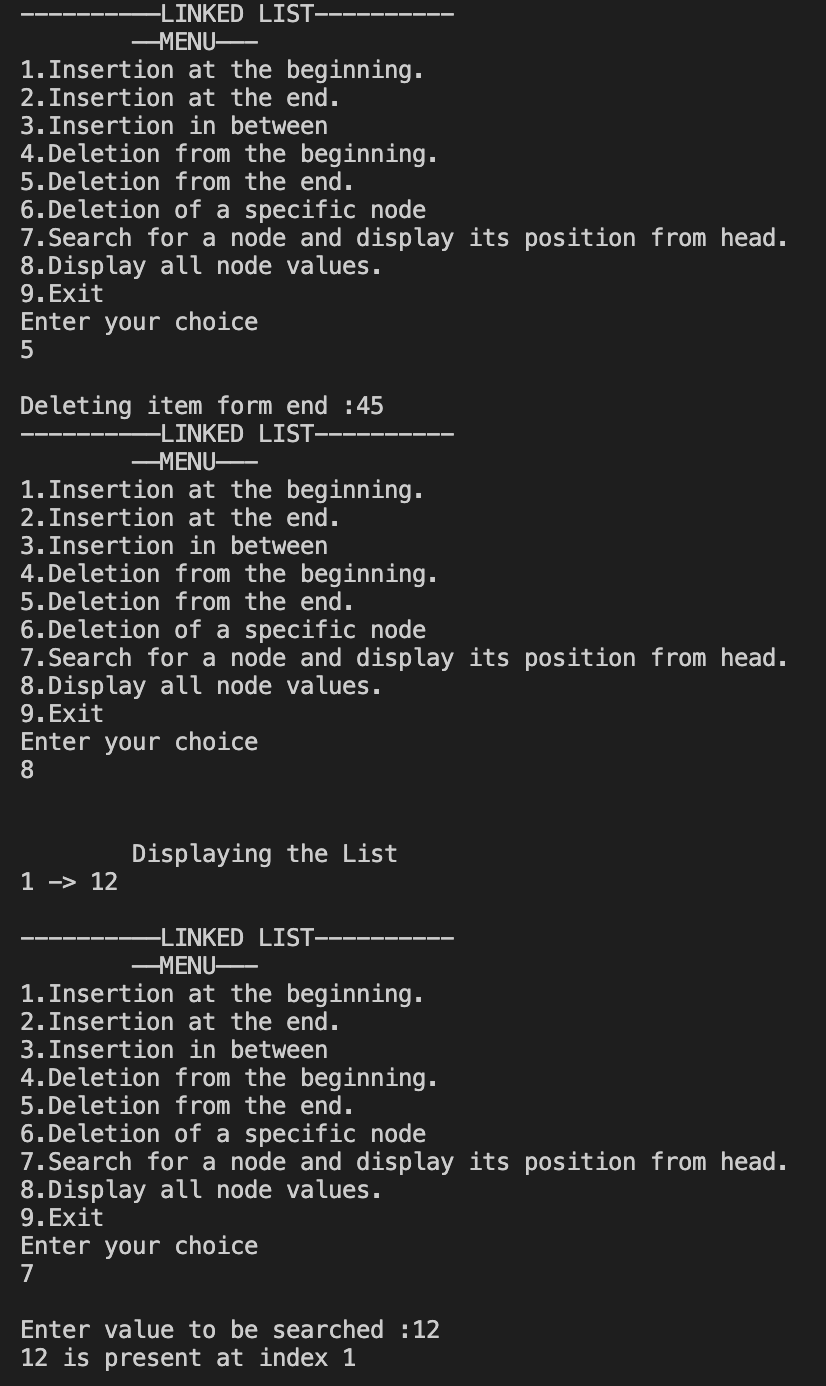
**OUTPUT**

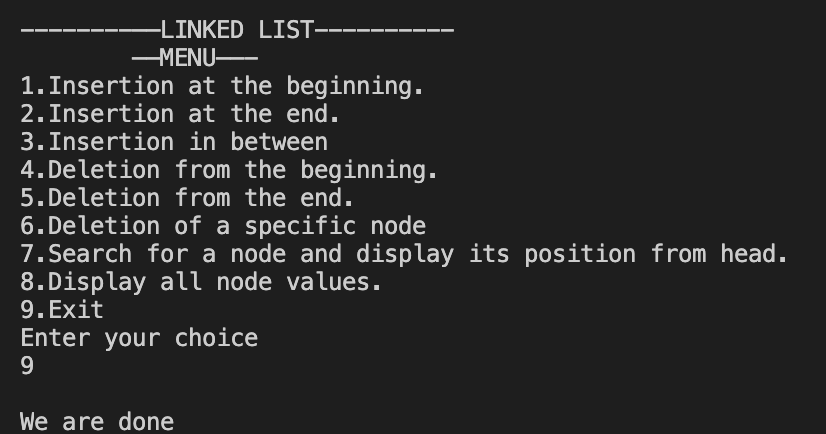












**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Question 2**

**Write a program to count the number of occurrences of a given key in a singly linked list and then delete all the occurrences. For example, if the given linked list is 1->2->1- >2->1->3->1 and given key is 1, then the output should be 4. After deletion of all the occurrences of 1, the linked list is 2->2->3.**

**SOLUTION CODE**

**/\***

**Write a program to count the number of occurrences of a given key in a singly linked**

**list and then delete all the occurrences. For example, if given linked list is 1->2->1-**

**>2->1->3->1 and given key is 1, then output should be 4. After deletion of all the**

**occurrences of 1, the linked list is 2->2->3.**

**\*/**

**import java.util.\*;**

**// child class which can use basic operation of parent class**

**//TO USE THIS PLEASE CALL THE CLASS LinkedList TOO**

**class a2ques2 extends LinkedList**

**{**

**static Scanner scr= new Scanner(System.in);**

**static void input(LinkedList list)**

**{**

**System.out.println("Enter number of elements ");**

**int n = scr.nextInt();**

**for (int i = 0; i < n; i++)**

**{**

**System.out.println("Enter value of " + i + "th element ");**

**int inp= scr.nextInt();**

**insertAtLast(list,inp);**

**}**

**}**

**// to remove duplicate and print**

**static void duplicate (LinkedList l,int val)**

**{**

**int dup=0;**

**while (l.head.data==val)**

**{**

**l.head=l.head.next;**

**}**

**Node prev = l.head;**

**Node curr= l.head;**

**while (curr!=null)**

**{**

**if (curr.data==val)**

**{**

**dup++;**

**prev.next=curr.next;**

**}**

**prev=curr;**

**curr=curr.next;**

**}**

**if (dup==0)**

**System.out.println("No node found with "+val+" value.");**

**else**

**System.out.println("\nNode with "+val+" values are : "+dup);**

**print(l);**

**}**

**//main function**

**public static void main(String[] args)**

**{**

**LinkedList list = new LinkedList();**

**input(list);**

**System.out.print("Enter value to be removed : ");**

**int n=scr.nextInt();**

**System.out.print("Elements before removal : ");**

**print(list);**

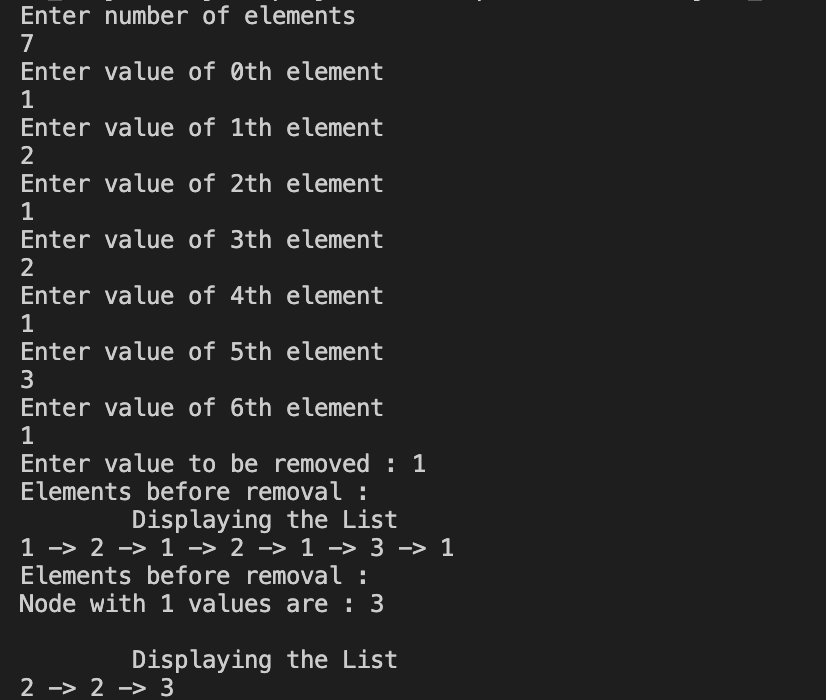
**System.out.print("Elements before removal : ");**

**duplicate(list, n);**

**}**

**}**

**OUTPUT**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Question 3**

**Write a program to find the middle of a linked list**

**SOLUTION CODE**

**/\***

**Write a program to find the middle of a linked list\**

**\*/**

**// child class which can use basic operation of parent class**

**//TO USE THIS PLEASE CALL THE CLASS LinkedList TOO**

**import java.util.\*;**

**class a2ques3 extends LinkedList**

**{**

**static Scanner scr= new Scanner(System.in);**

**static void input(LinkedList list)**

**{**

**System.out.println("Enter number of elements ");**

**int n = scr.nextInt();**

**for (int i = 0; i < n; i++)**

**{**

**System.out.println("Enter value of " + i + "th element ");**

**int inp= scr.nextInt();**

**insertAtLast(list,inp);**

**}**

**}**

**public static String getMiddle(LinkedList l)**

**{**

**if (l.head == null)**

**return "NOT FOUND";**

**Node slow = l.head, fast = l.head;**

**while (fast.next != null && fast.next.next != null)**

**{**

**slow = slow.next;**

**fast = fast.next.next;**

**}**

**return Integer.toString(slow.data);**

**}**

**public static void main(final String[] args)**

**{**

**LinkedList l1= new LinkedList();**

**input(l1);**

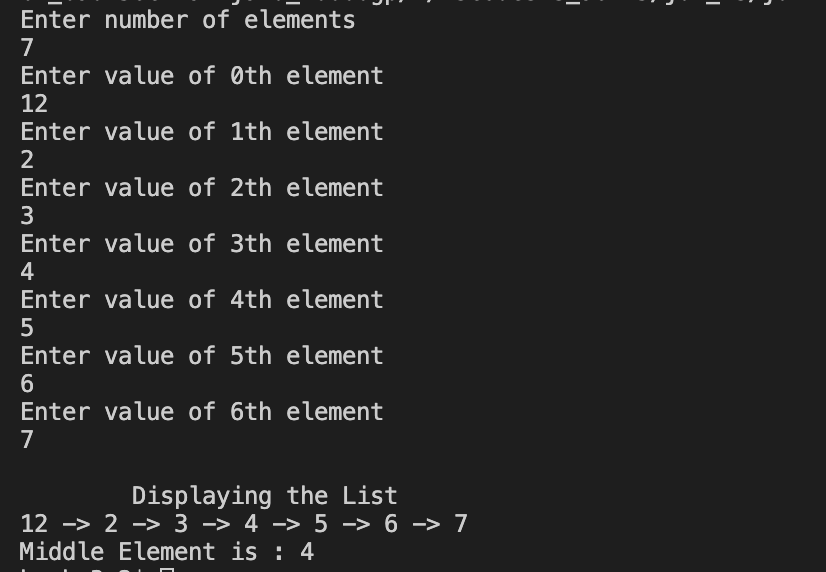
**print(l1);**

**System.out.println("Middle Element is : "+getMiddle(l1));**

**}**

**}**

***OUTPUT***

******

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Question 4**

**Write a program to reverse a linked list.**

**SOLUTION CODE**

**//Write a program to reverse a linked list.**

**import java.io.\*;**

**import java.util.\*;**

**class a2ques4 extends LinkedList**

**{**

**static Scanner scr=new Scanner (System.in);**

**//insertion**

**static void input(LinkedList list)**

**{**

**System.out.println("Enter number of elements ");**

**int n = scr.nextInt();**

**for (int i = 0; i < n; i++)**

**{**

**System.out.println("Enter value of " + i + "th element ");**

**int inp= scr.nextInt();**

**insertAtLast(list,inp);**

**}**

**}**

**//reverse**

**public static void reverse(LinkedList list)**

**{**

**Node pointer = list.head;**

**Node previous = null, current = null;**

**while (pointer != null)**

**{**

**current = pointer;**

**pointer = pointer.next;**

**// reverse the link**

**current.next = previous;**

**previous = current;**

**list.head = current;**

**}**

**}**

**public static void main(final String[] args)**

**{**

**LinkedList l1= new LinkedList();**

**input(l1);**

**print(l1);**

**System.out.println("\nReverse elements: ");**

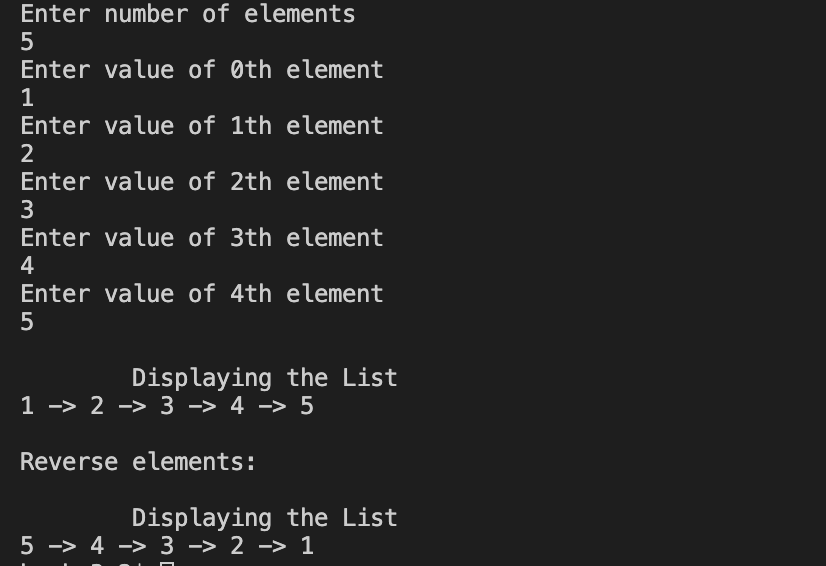
**reverse(l1);**

**print(l1);**

**}**

**}**

***OUTPUT***

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**EXTRA**

**Question**

**Rotate a linked list to n places**

**If the linked list is 1 -> 2 -> 3 -> 4 -> 5.**

**Then rotated linked list with k=2 rotation (let’s suppose) is 4 -> 5 -> 1 -> 2 -> 3**

**SOLUTION CODE**

**/\***

**Given a list, rotate the list to the right by k places, where k is non-negative.**

**For example:**

**Given 1->2->3->4->5->NULL and k = 2,**

**return 4->5->1->2->3->NULL.**

**\*/**

**import java.io.\*;**

**import java.util.\*;**

**class a2rotate extends LinkedList**

**{**

**static Scanner scr=new Scanner (System.in);**

**static int n;**

**//insertion**

**static void input(LinkedList list)**

**{**

**System.out.println("Enter number of elements ");**

**n = scr.nextInt();**

**for (int i = 0; i < n; i++)**

**{**

**System.out.println("Enter value of " + i + "th element ");**

**int inp= scr.nextInt();**

**insertAtLast(list,inp);**

**}**

**}**

**public static void main(final String[] args)**

**{**

**LinkedList l1= new LinkedList();**

**input(l1);**

**print(l1);**

**System.out.print("Enter places to be rotated : ");**

**int k= scr.nextInt();**

**int x= Math.abs(n-k);**

**Node temp=l1.head;**

**while(x>1)**

**{**

**x--;**

**temp=temp.next;**

**}**

**Node newHead=temp.next;**

**temp.next=null;**

**temp=newHead;**

**while(temp.next!=null)**

**{**

**temp=temp.next;**

**}**

**temp.next=l1.head;**

**l1.head=newHead;**

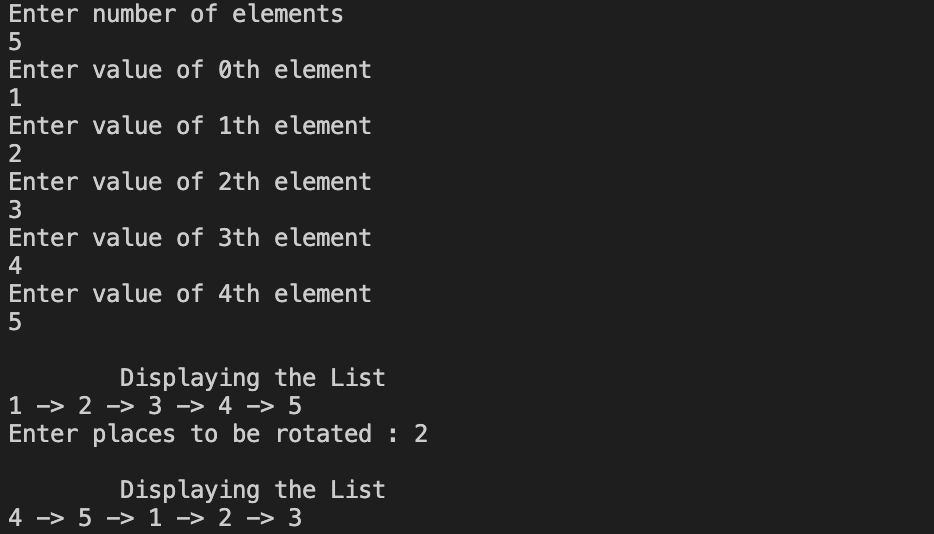
**temp=l1.head;**

**print(l1);**

**}**

**}**

***OUTPUT***

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Question**

**Reverse a linked list from position m to n. Do it in-place and in one-pass. For example: Given 1->2->3->4->5->NULL, m = 2 and n = 4, return 1->4->3->2->5->NULL.**

**SOLUTION CODE**

**/\***

**Reverse a linked list from position m to n. Do it in-place and in one-pass.**

**For example:**

**Given 1->2->3->4->5->NULL, m = 2 and n = 4,**

**return 1->4->3->2->5->NULL.**

**\*/**

**import java.io.\*;**

**import java.util.\*;**

**class a2reverse extends LinkedList**

**{**

**static Scanner scr=new Scanner (System.in);**

**//insertion**

**static void input(LinkedList list)**

**{**

**System.out.println("Enter number of elements ");**

**int n = scr.nextInt();**

**for (int i = 0; i < n; i++)**

**{**

**System.out.println("Enter value of " + i + "th element ");**

**int inp= scr.nextInt();**

**insertAtLast(list,inp);**

**}**

**}**

**public static void reverse(LinkedList list)**

**{**

**Node pointer = list.head;**

**Node previous = null, current = null;**

**while (pointer != null)**

**{**

**current = pointer;**

**pointer = pointer.next;**

**// reverse the link**

**current.next = previous;**

**previous = current;**

**list.head = current;**

**}**

**}**

**public static void main(final String[] args)**

**{**

**LinkedList l1= new LinkedList();**

**input(l1);**

**print(l1);**

**System.out.print("Enter initial index m : ");**

**int m = scr.nextInt();**

**System.out.print("Enter final index n : ");**

**int n = scr.nextInt();**

**Node prev=l1.head;**

**LinkedList part= new LinkedList();**

**Node curr;**

**for(int i=1;i<m-1 ;i++)**

**{**

**prev=prev.next;**

**if(prev.next==null)**

**System.out.print("n value out of range");**

**}**

**curr = prev.next;**

**for(int i=m;i<=n && curr!=null;i++)**

**{**

**insertAtLast(part, curr.data);**

**if (curr.next==null)**

**System.out.print("n value out of range");**

**curr=curr.next;**

**}**

**reverse(part);**

**Node add=part.head;**

**while (add.next!=null)**

**add=add.next;**

**prev.next=part.head;**

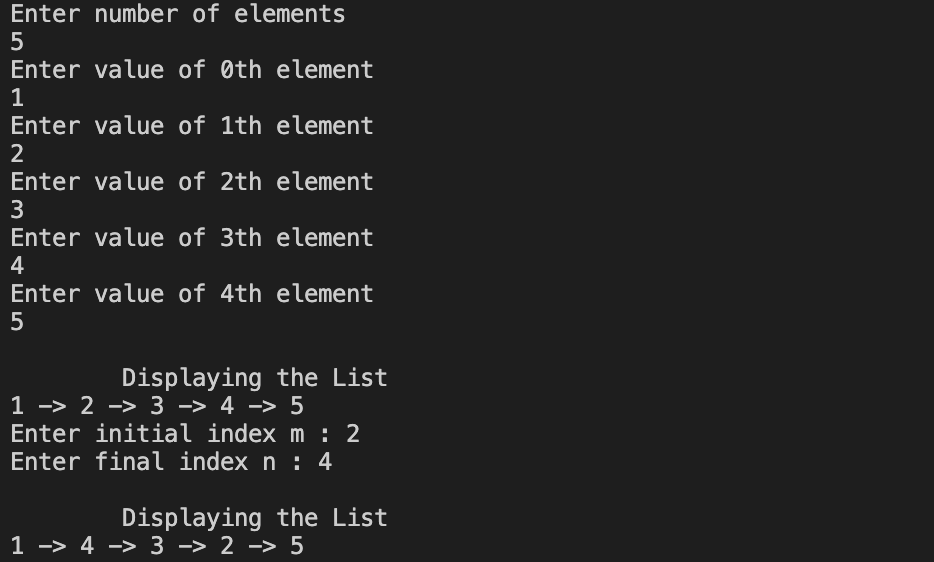
**add.next=curr;**

**print(l1);**

**}**

**}**

***OUTPUT***

******

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Question**

**Write a program to add two polynomials using a linked list.**

**SOLUTION CODE**

**//Write a program to add two polynomials using a linked list.**

**import java.io.\*;**

**import java.util.\*;**

**class Polynomial**

**{**

**Node head;**

**static class Node**

**{**

**int value;**

**int degree;**

**Node next;**

**Node(int v, int d)**

**{**

**value=v;**

**degree=d;**

**next=null;**

**}**

**}**

**static Scanner scr= new Scanner(System.in);**

**public static Polynomial insertAtLast(Polynomial p, int val,int deg)**

**{**

**Node new\_Node=new Node(val,deg);**

**new\_Node.next=null;**

**if (p.head==null)**

**{**

**p.head = new\_Node;**

**}**

**else**

**{**

**Node last = p.head;**

**while(last.next!=null)**

**{**

**last = last.next;**

**}**

**last.next = new\_Node;**

**}**

**return p;**

**}**

**static void input(Polynomial p)**

**{**

**System.out.println("Enter number of terms ");**

**int n = scr.nextInt();**

**for (int i = 0; i < n; i++)**

**{**

**System.out.println("Enter coefficient of " + i + "th term ");**

**int v= scr.nextInt();**

**System.out.println("Enter degree of " + i + "th term with "+v+" as value");**

**int d= scr.nextInt();**

**insertAtLast(p,v,d);**

**}**

**}**

**static void print(Polynomial p)**

**{**

**Node current = p.head;**

**System.out.println("\nThe polynomial expression is ");**

**while (current!=null)**

**{**

**if (current.next!=null)**

**System.out.print(current.value+"x^"+current.degree+" + ");**

**else**

**System.out.println(current.value+"x^"+current.degree);**

**current= current.next;**

**}**

**}**

**public static void main(String[] args)**

**{**

**Polynomial p1= new Polynomial();**

**Polynomial p2= new Polynomial();**

**input(p1);**

**input(p2);**

**print(p1);**

**print(p2);**

**// to add both poylnomial**

**Polynomial add= new Polynomial();**

**Node curr1=p1.head;**

**boolean res;**

**while(curr1!=null)**

**{**

**res=false;**

**Node curr2=p2.head;**

**while(curr2!=null)**

**{**

**if(curr1.degree==curr2.degree)**

**{**

**res=true;**

**insertAtLast(add, curr1.value + curr2.value , curr1.degree);**

**}**

**curr2=curr2.next;**

**}**

**if(!res)**

**insertAtLast(add, curr1.value, curr1.degree);**

**curr1=curr1.next;**

**}**

**Node curr2=p2.head;**

**while(curr2!=null)**

**{**

**res=false;**

**Node fin=add.head;**

**while(fin!=null)**

**{**

**if(fin.degree==curr2.degree)**

**{**

**res=true;**

**}**

**fin=fin.next;**

**}**

**if(!res)**

**insertAtLast(add, curr2.value, curr2.degree);**

**curr2=curr2.next;**

**}**

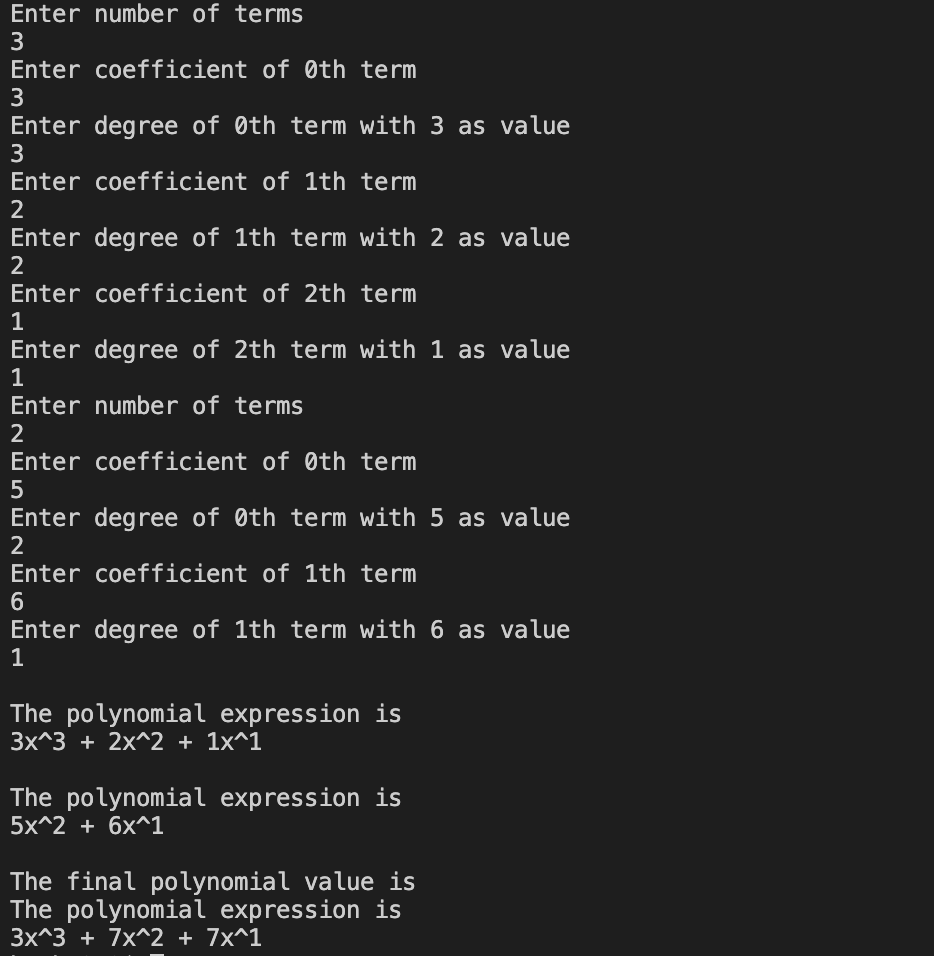
**System.out.print("\nThe final polynomial value is ");**

**print(add);**

**}**

**}**

***OUTPUT***

******

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Question**

**Write a program to get the intersection point of two Linked Lists.**

**SOLUTION CODE**

**//Write a function to get the intersection point of two Linked Lists**

**import java.io.\*;**

**import java.util.\*;**

**class a2ques extends LinkedList**

**{**

**static Scanner scr=new Scanner (System.in);**

**//insertion**

**static void input(LinkedList list)**

**{**

**System.out.println("Enter number of elements ");**

**int n = scr.nextInt();**

**for (int i = 0; i < n; i++)**

**{**

**System.out.println("Enter value of " + i + "th element ");**

**int inp= scr.nextInt();**

**insertAtLast(list,inp);**

**}**

**}**

**//reverse**

**public static void main(final String[] args)**

**{**

**LinkedList l1= new LinkedList();**

**input(l1);**

**print(l1);**

**LinkedList l2= new LinkedList();**

**input(l2);**

**print(l2);**

**Node curr1 =l1.head;**

**boolean ans = false;**

**int c=1;**

**out: while (curr1!=null)**

**{**

**Node curr2=l2.head;**

**while (curr2!=null)**

**{**

**if (curr1.data==curr2.data)**

**{**

**ans=true;**

**break out;**

**}**

**curr2=curr2.next;**

**}**

**curr1=curr1.next;**

**c++;**

**}**

**if (ans)**

**System.out.println("Intersection point found at "+c+"th position and value - "+curr1.data);**

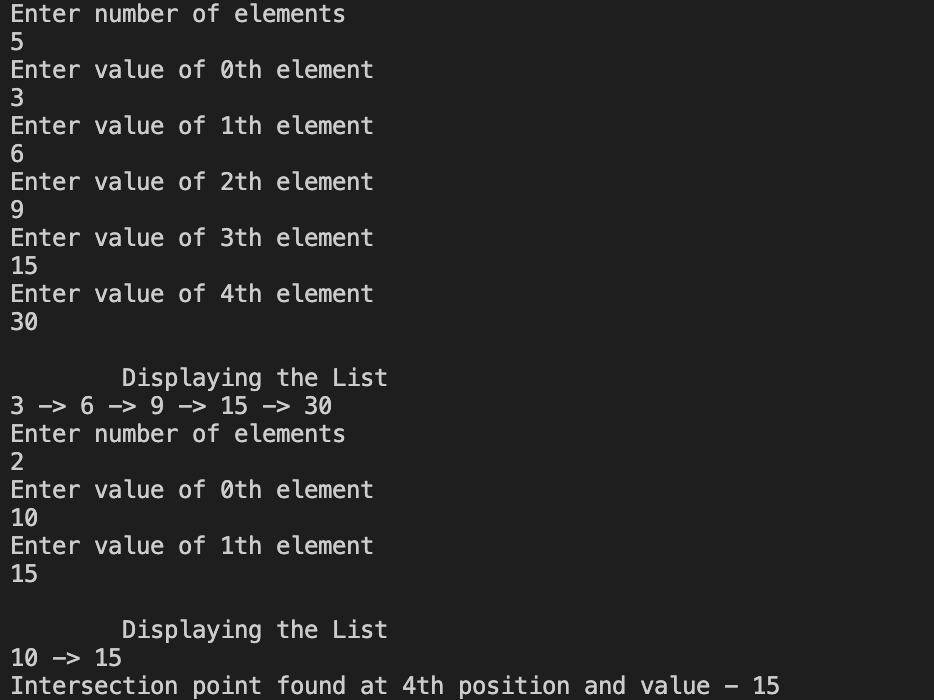
**else**

**System.out.println("Intersection point not found. ");**

**}**

**}**

***OUTPUT***

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**