**Q.1) Implementation of singly linked list (Insert , Display , Delete, Search, Count, Reverse).**

import java.util.Scanner;

class Node

    public int data;

    public Node next;

    public static Node head = null;

    public void add(int x) {

        Node temp = head;

        Node newNode = new Node();

        if (head == null) {

            newNode.data = x;

            newNode.next = null;

            head = newNode;

        } else {

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.next = newNode;

            newNode.data = x;

            newNode.next = null;

        }

    }

    public int count() {

        Node temp = head;

        int cnt = 0;

        if (head == null) {

            return -1;

        } else {

            while (temp != null) {

                cnt++;

                temp = temp.next;

            }

        }

        return cnt;

    }

    public boolean search(int target) {

        if (head == null) {

            return false;

        }

        Node temp = head;

        while (temp != null) {

            if (temp.data == target) {

                return true;

            }

            temp = temp.next;

        }

        return false;

    }

    public void delete(int pos) {

        if (head == null) {

            System.out.println("Linked list is empty.");

            return;

        }

        if (pos < 0 || pos > count()) {

            System.out.println("Invalid position");

            return;

        }

        if (pos == 1) {

            head = head.next;

            return;

        }

        Node temp = head;

        if (pos == count()) {

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

                temp = temp.next;

            }

            temp.next = null;

        } else {

            int cnt = 0;

            Node prev = null;

            while (temp != null) {

                cnt++;

                if (cnt == pos) {

                    prev.next = temp.next;

                    break;

                }

                prev = temp;

                temp = temp.next;

            }

        }

    }

    public void reverse() {

        if (head == null) {

            System.out.println("No elements");

            return;

        }

        Node temp = head;

        Node prev = null;

        while (temp != null) {

            Node nextNode = temp.next;

            temp.next = prev;

            prev = temp;

            temp = nextNode;

        }

        head = prev;

        display();

    }

    public void insert(int x, int pos) {

        Node temp = head;

        if (pos == 1) {

            Node newHead = new Node();

            newHead.data = x;

            newHead.next = head;

            head = newHead;

        }

        if (pos == count() - 1) {

            add(x);

        } else {

            int cnt = 0;

            while (temp != null) {

                cnt++;

                if (cnt == pos - 1) {

                    Node nextNode = temp.next;

                    temp.next = new Node();

                    temp.next.data = x;

                    temp.next.next = nextNode;

                    break;

                }

                temp = temp.next;

            }

        }

        display();

    }

    public void display() {

        Node temp = head;

        while (temp != null) {

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

            temp = temp.next;

        }

        System.out.println();

    }

}

public class LL {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Node n = new Node();

        int option, num;

        char ch = 'Y';

        while (ch == 'Y' || ch == 'y') {

            System.out.println("1.To Add node\n2.To Display\n3.To delete position\n4.For count\n5.To Search\n6.Reverse\n7.To insert at a position");

            option = sc.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter the element : ");

                    num = sc.nextInt();

                    n.add(num);

                    break;

                case 2:

                    n.display();

                    break;

                case 3:

                    System.out.println("Enter the position you want to delete");

                    num = sc.nextInt();

                    n.delete(num);

                    n.display();

                    break;

                case 4:

                    num = n.count();

                    System.out.println("Number of elements in LL are : " + num);

                    break;

                case 5:

                    System.out.println("Enter the target");

                    num = sc.nextInt();

                    boolean ans = n.search(num);

                    if (ans == true) {

                        System.out.println("Target is present");

                    } else {

                        System.out.println("Target is not present");

                    }

                    break;

                case 6:

                    n.reverse();

                    break;

                case 7:

                    System.out.println("Enter pos and ele");

                    int pos = sc.nextInt();

                    int ele = sc.nextInt();

                    n.insert(ele, pos);

                    break;

                default:

                    System.out.println("Invalid input");

            }

            System.out.println("Do you want to continue ?");

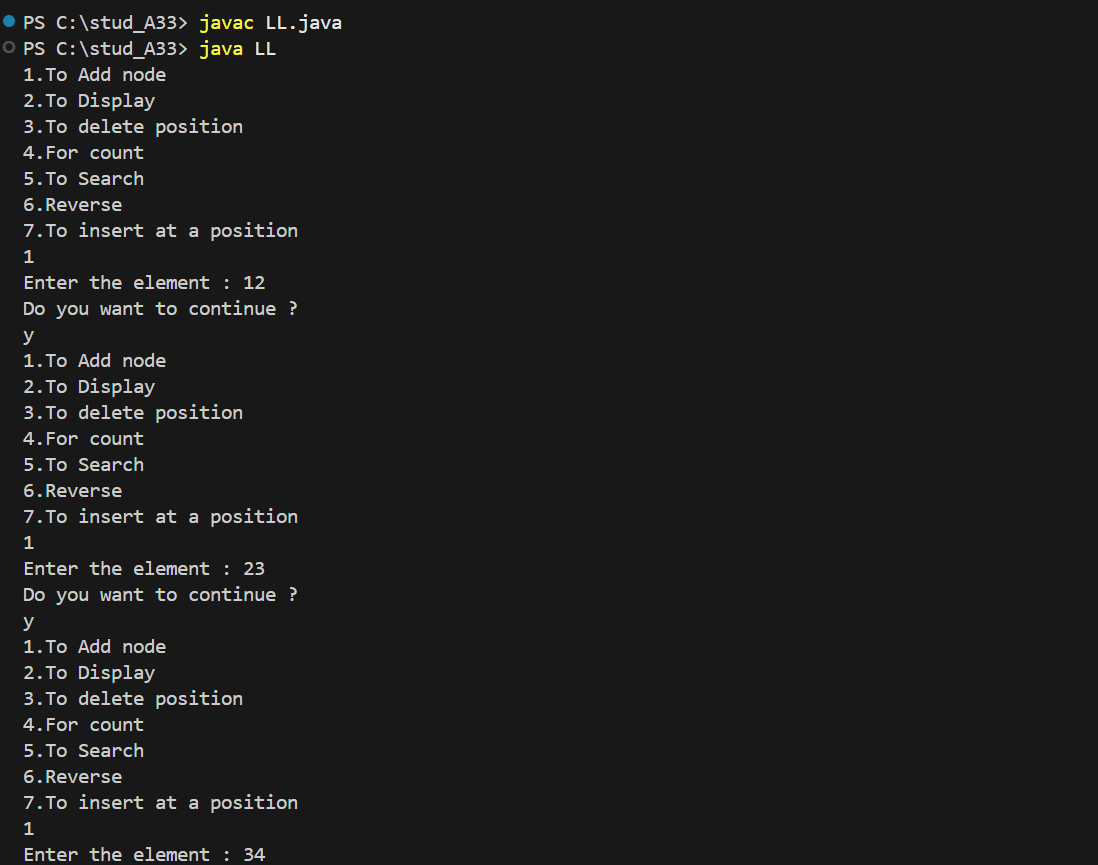
            ch = sc.next().charAt(0);

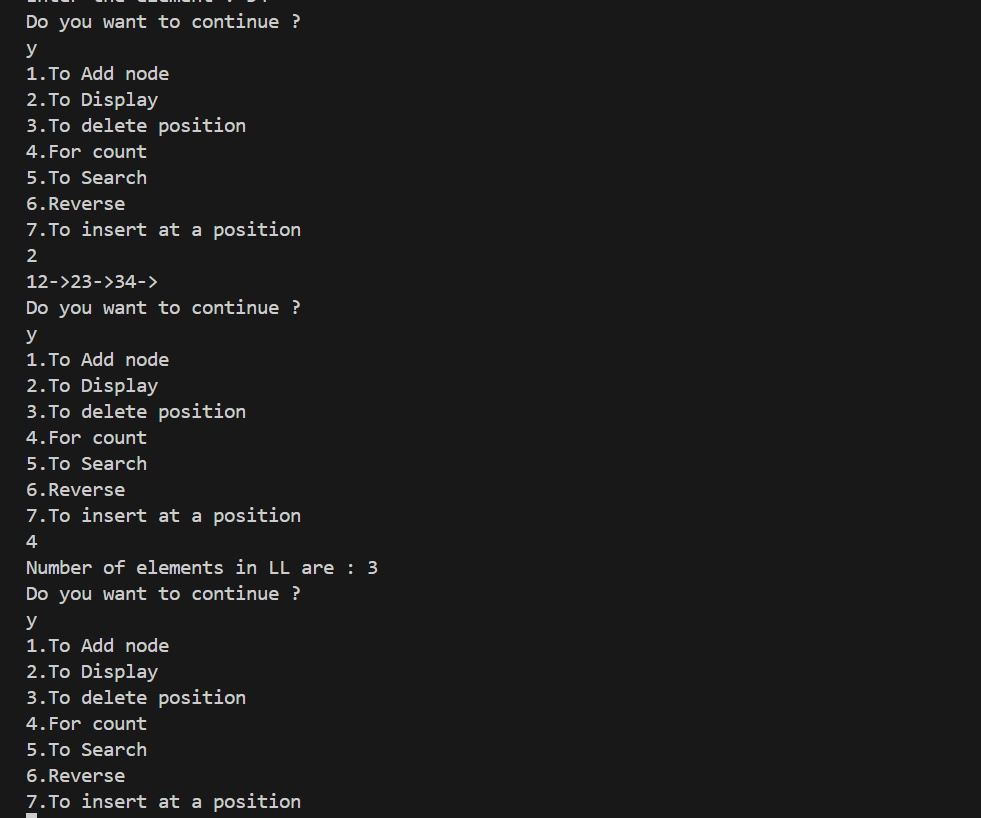
        }

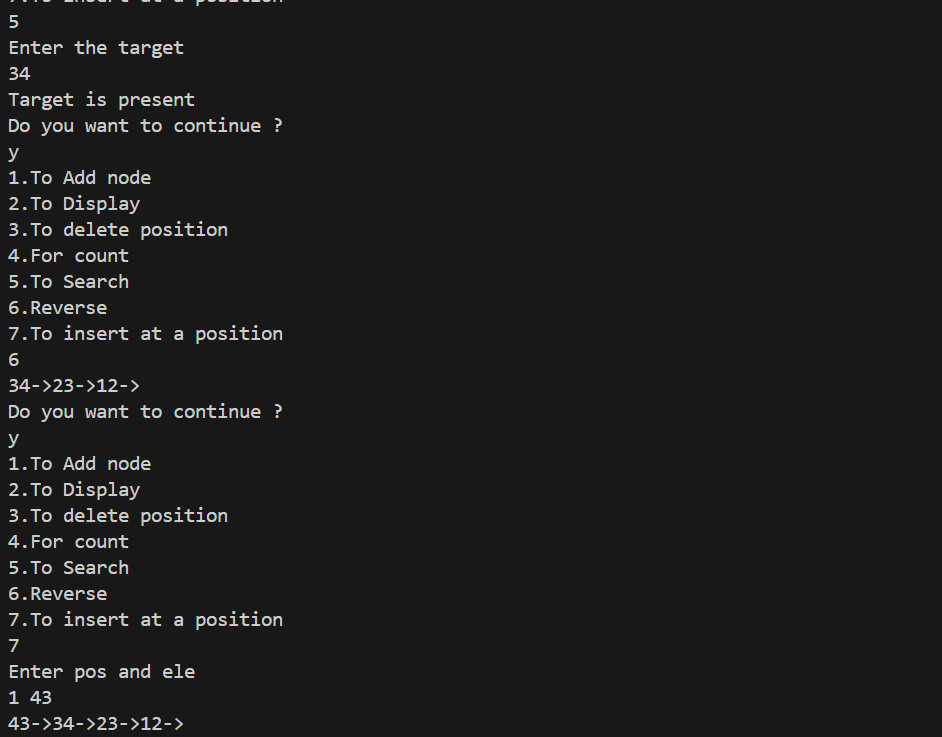
    }

}

Output :







**Q.2) Implementation of doubly Linked list (Insert , Display, Delete,Search,Count).**

import java.util.Scanner;

class Node {

    public int data;

    public Node next;

    public Node prev;

    public static Node head = null;

    public void add(int x) {

        Node newNode = new Node();

        newNode.data = x;

        newNode.next = null;

        newNode.prev = null;

        if (head == null) {

            head = newNode;

        } else {

            Node temp = head;

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.next = newNode;

            newNode.prev = temp;

        }

    }

    public void delete(int pos) {

        if (pos < 1 || pos > count()) {

            System.out.println("Invalid position.");

            return;

        }

        if (head == null) {

            System.out.println("No elements to delete");

            return;

        }

        if (pos == 1) {

            if (head.next != null) {

                head = head.next;

                head.prev = null;

            } else {

                head = null;

            }

            return;

        }

        Node temp = head;

        if (pos == count()) {

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.prev.next = null;

        } else {

            int cnt = 0;

            while (temp != null) {

                cnt++;

                if (cnt == pos) {

                    temp.prev.next = temp.next;

                    temp.next.prev = temp.prev;

                    break;

                }

                temp = temp.next;

            }

        }

    }

    public int count() {

        Node temp = head;

        int cnt = 0;

        if (head == null) {

            return -1;

        } else {

            while (temp != null) {

                cnt++;

                temp = temp.next;

            }

        }

        return cnt;

    }

    public void insert(int ele, int pos) {

        Node temp = head;

        if (head == null && pos == 1) {

            Node newNode = new Node();

            newNode.data = ele;

            newNode.next = null;

            newNode.prev = null;

            head = newNode;

            return;

        }

        if (pos == 1) {

            Node newHead = new Node();

            newHead.data = ele;

            newHead.next = head;

            head.prev = newHead;

            newHead.prev = null;

            head = newHead;

            return;

        }

        if (pos == count() + 1) {

            add(ele);

            return;

        }

        int cnt = 0;

        while (temp != null) {

            cnt++;

            if (cnt == pos - 1) {

                Node newNode = new Node();

                newNode.data = ele;

                newNode.next = temp.next;

                if (temp.next != null) {

                    temp.next.prev = newNode;

                }

                newNode.prev = temp;

                temp.next = newNode;

                break;

            }

            temp = temp.next;

        }

    }

    public boolean search(int target) {

        if (head == null) {

            return false;

        }

        Node temp = head;

        while (temp != null) {

            if (temp.data == target) {

                return true;

            }

            temp = temp.next;

        }

        return false;

    }

    public void display() {

        Node temp = head;

        while (temp != null) {

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

            temp = temp.next;

        }

        System.out.println();

    }

}

public class DLL {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Node n = new Node();

        int option, num;

        char ch = 'Y';

        while (ch == 'Y' || ch == 'y') {

            System.out.println("1.To Add node\n2.To Display\n3.For count\n4.To Search\n5.To delete a position\n6.To insert at position");

            option = sc.nextInt();

            switch (option) {

                case 1:

                    System.out.print("Enter the element : ");

                    num = sc.nextInt();

                    n.add(num);

                    break;

                case 2:

                    n.display();

                    break;

                case 3:

                    num = n.count();

                    System.out.println("Number of elements in DLL are : " + num);

                    break;

                case 4:

                    System.out.println("Enter the target");

                    num = sc.nextInt();

                    boolean ans = n.search(num);

                    if (ans == true) {

                        System.out.println("Target is present");

                    } else {

                        System.out.println("Target is not present");

                    }

                    break;

                case 5:

                    System.out.println("Enter the position you want to delete");

                    num = sc.nextInt();

                    n.delete(num);

                    n.display();

                    break;

                case 6:

                    System.out.println("Enter pos and ele");

                    int pos = sc.nextInt();

                    int ele = sc.nextInt();

                    n.insert(ele, pos);

                    break;

                default:

                    System.out.println("Invalid input");

            }

            System.out.println("Do you want to continue ?");

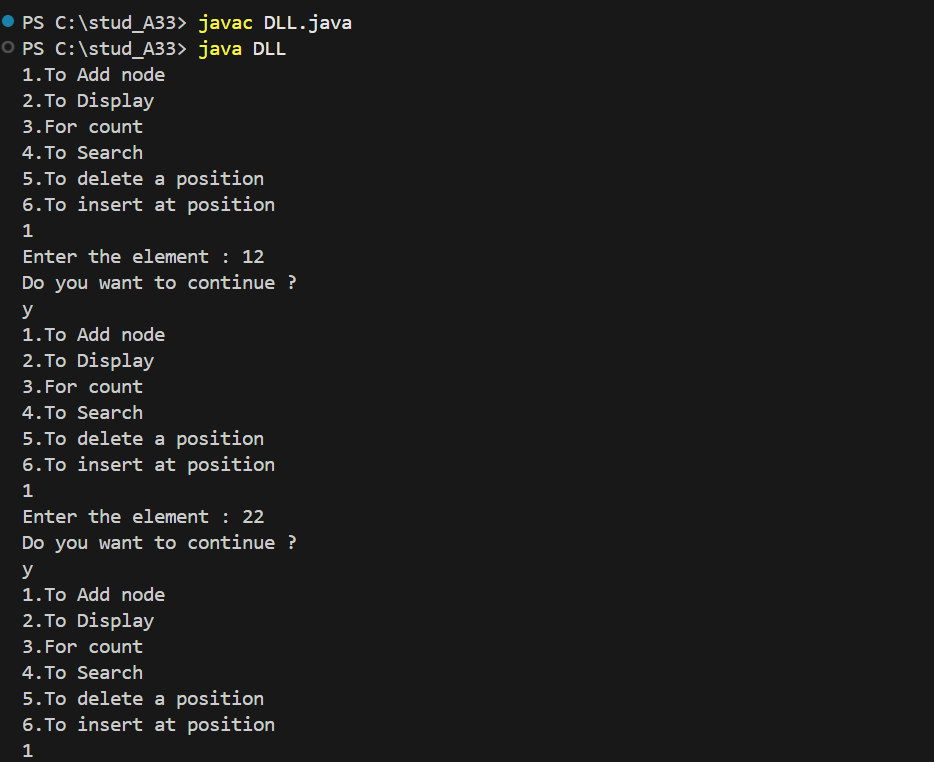
            ch = sc.next().charAt(0);

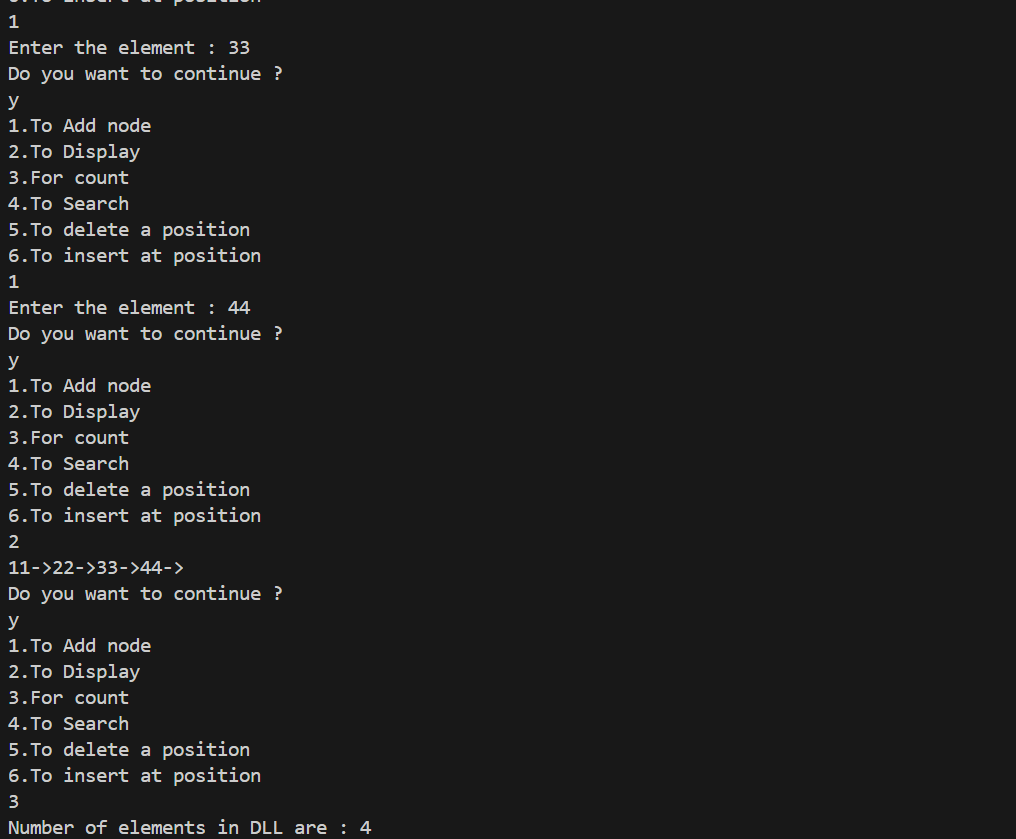
        }

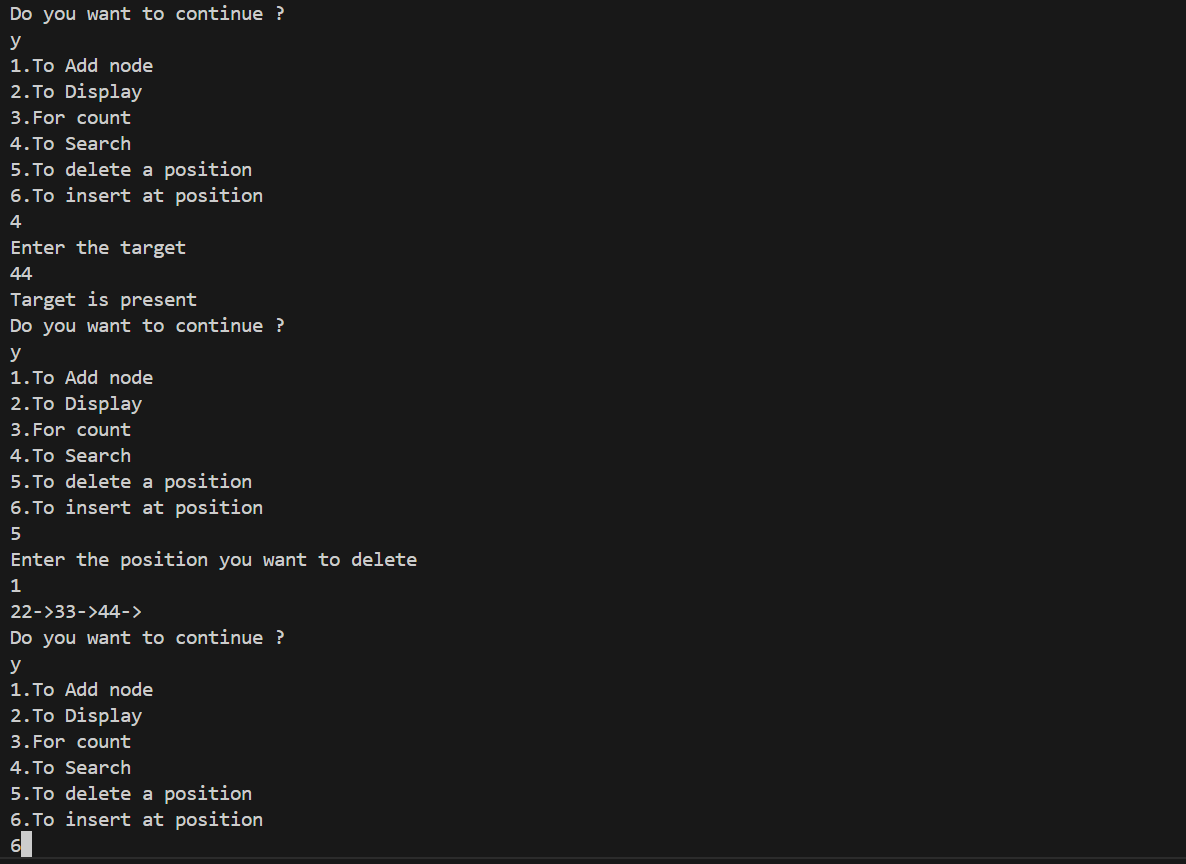
    }

}

Output :









**Q.3) Implementation of circular linked list(Insert, Display, Delete, Search, Count).**

import java.util.Scanner;

class Node {

    public int data;

    public Node next;

    public Node prev;

    public static Node head = null;

    public void makeCircular() {

        if (head == null) {

            return;

        }

        Node last = head;

        while (last.next != null && last.next != head) {

            last = last.next;

        }

        last.next = head;

        head.prev = last;

    }

    public void add(int x) {

        Node newNode = new Node();

        newNode.data = x;

        if (head == null) {

            head = newNode;

            head.next = head;

            head.prev = head;

        } else {

            Node last = head.prev;

            last.next = newNode;

            newNode.prev = last;

            newNode.next = head;

            head.prev = newNode;

        }

    }

    public void print() {

        if (head == null) {

            System.out.println("No LinkedList elements");

            return;

        }

        Node temp = head;

        do {

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

            temp = temp.next;

        } while (temp != head);

        System.out.println("(back to head)");

    }

    public int count() {

        int cnt = 0;

        if (head == null) {

            return cnt;

        }

        Node temp = head;

        do {

            cnt++;

            temp = temp.next;

        } while (temp != head);

        return cnt;

    }

    public void insert(int pos, int x) {

        if (pos < 1 || pos > count() + 1) {

            System.out.println("Invalid position");

            return;

        }

        Node newNode = new Node();

        newNode.data = x;

        if (pos == 1) {

            if (head == null) {

                head = newNode;

                head.next = head;

                head.prev = head;

            } else {

                Node last = head.prev;

                newNode.next = head;

                newNode.prev = last;

                head.prev = newNode;

                last.next = newNode;

                head = newNode;

            }

        } else {

            Node temp = head;

            for (int i = 1; i < pos - 1; i++) {

                temp = temp.next;

            }

            newNode.next = temp.next;

            newNode.prev = temp;

            temp.next.prev = newNode;

            temp.next = newNode;

        }

        print();

    }

    public void remove(int pos) {

        if (head == null) {

            System.out.println("List is empty, nothing to remove.");

            return;

        }

        if (pos < 1 || pos > count()) {

            System.out.println("Invalid position: " + pos);

            return;

        }

        if (pos == 1) {

            if (head.next == head) {

                head = null;

            } else {

                Node last = head.prev;

                head = head.next;

                head.prev = last;

                last.next = head;

            }

        } else {

            Node temp = head;

            for (int i = 1; i < pos; i++) {

                temp = temp.next;

            }

            temp.prev.next = temp.next;

            temp.next.prev = temp.prev;

        }

        print();

    }

    public void search(int x) {

        if (head == null) {

            System.out.println("No elements in list.");

            return;

        }

        Node temp = head;

        boolean found = false;

        do {

            if (temp.data == x) {

                found = true;

                break;

            }

            temp = temp.next;

        } while (temp != head);

        System.out.println(found ? "Element found" : "Element not found");

    }

}

public class CircularDLL {

    public static void main(String[] args) {

        Node n = new Node();

        int option, num;

        char ch = 'Y';

        Scanner sc = new Scanner(System.in);

        while (ch == 'Y' || ch == 'y') {

            System.out.println("1.Add node\n2.Print list\n3.Count elements\n4.Insert \n5.Search \n6.Remove \n7.Make Circular");

            option = sc.nextInt();

            switch (option) {

                case 1:

                    System.out.println("Enter the value of the node:");

                    num = sc.nextInt();

                    n.add(num);

                    break;

                case 2:

                    n.print();

                    break;

                case 3:

                    int cnt = n.count();

                    System.out.println("There are " + cnt + " elements in the linked list.");

                    break;

                case 4:

                    System.out.println("Enter position:");

                    int pos = sc.nextInt();

                    System.out.println("Enter data:");

                    int x = sc.nextInt();

                    n.insert(pos, x);

                    break;

                case 5:

                    System.out.println("Enter target:");

                    int target = sc.nextInt();

                    n.search(target);

                    break;

                case 6:

                    System.out.println("Enter the position:");

                    int posi = sc.nextInt();

                    n.remove(posi);

                    break;

                case 7:

                    n.makeCircular();

                    System.out.println("List made circular.");

                    break;

                default:

                    System.out.println("Invalid option. Please try again.");

                    break;

            }

            System.out.println("Do you want to continue? (Y/N)");

            ch = sc.next().charAt(0);

            if (ch == 'N' || ch == 'n') {

                break;

            }

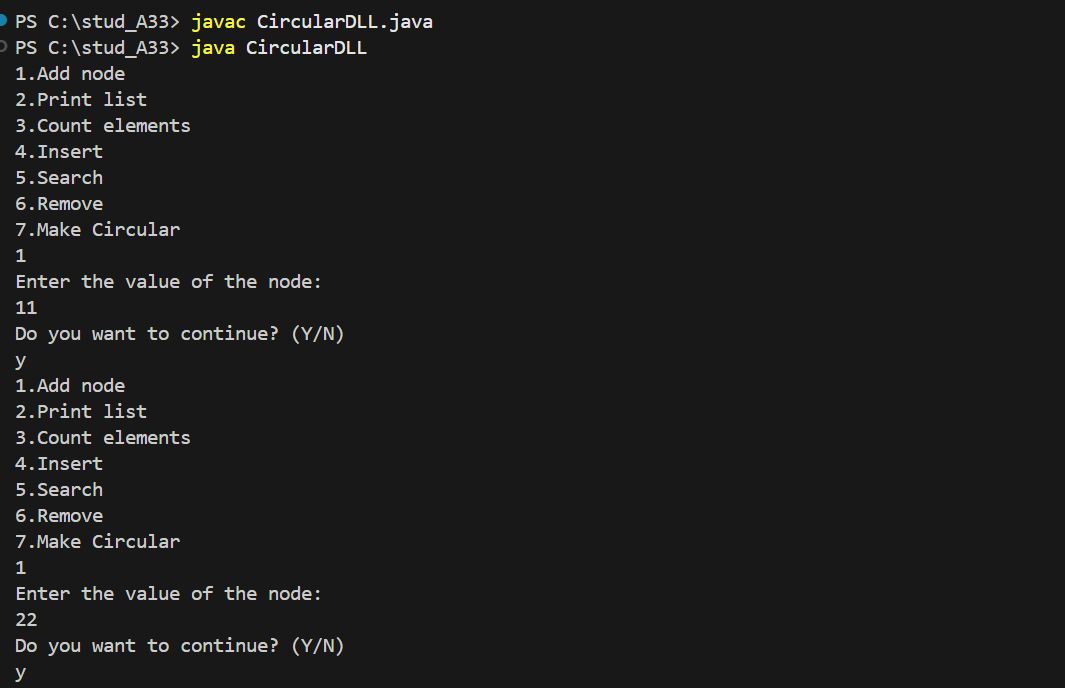
        }

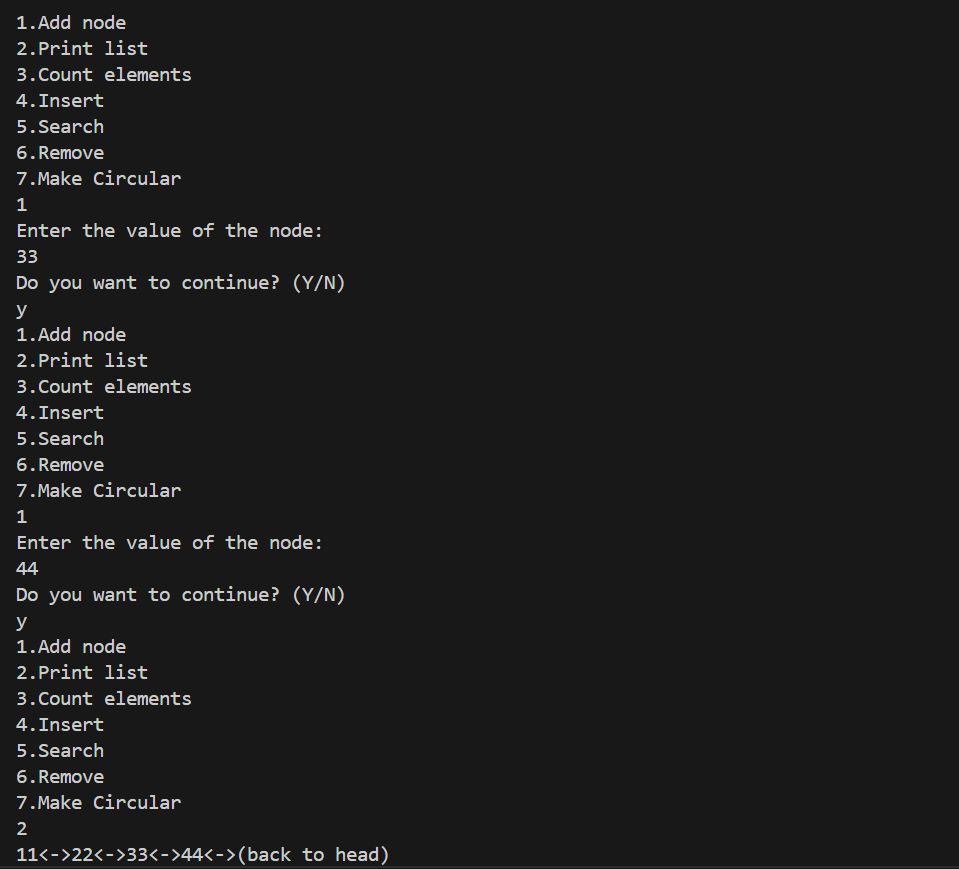
        sc.close();

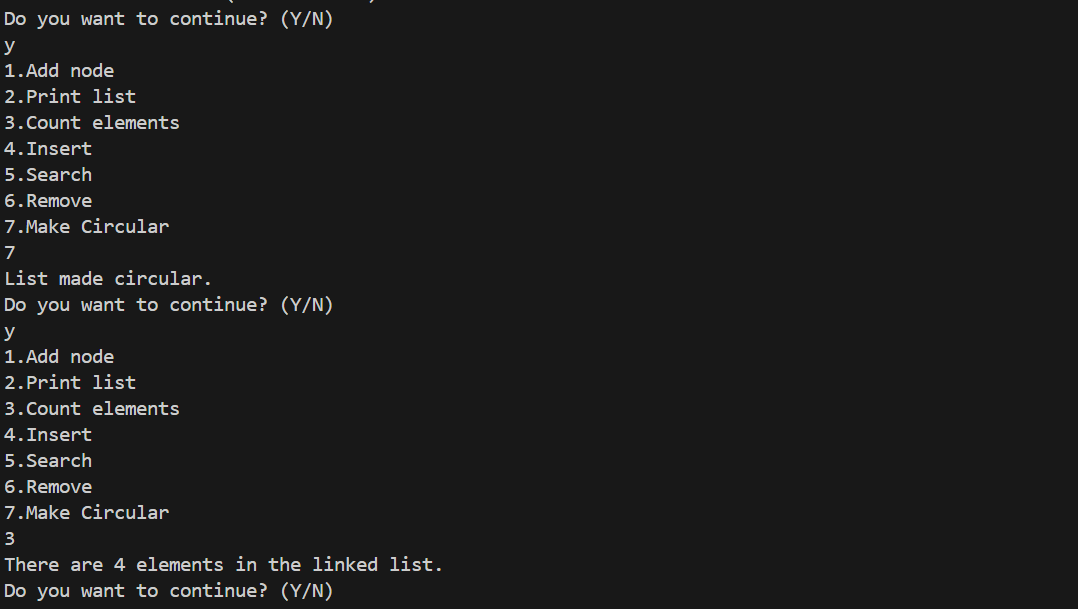
    }

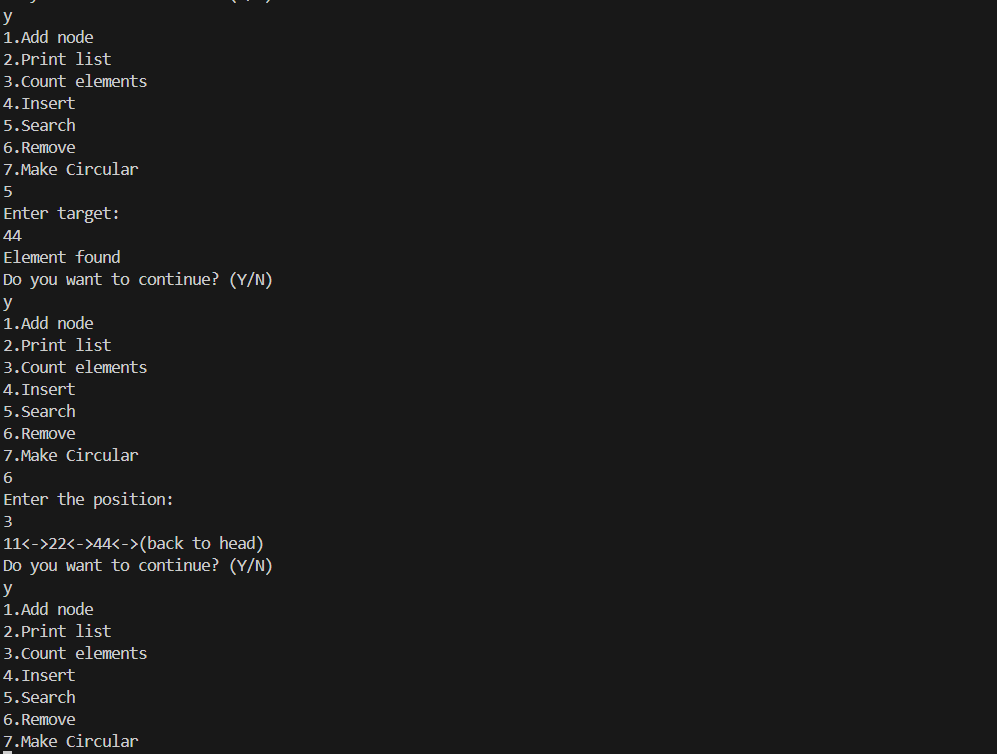
}

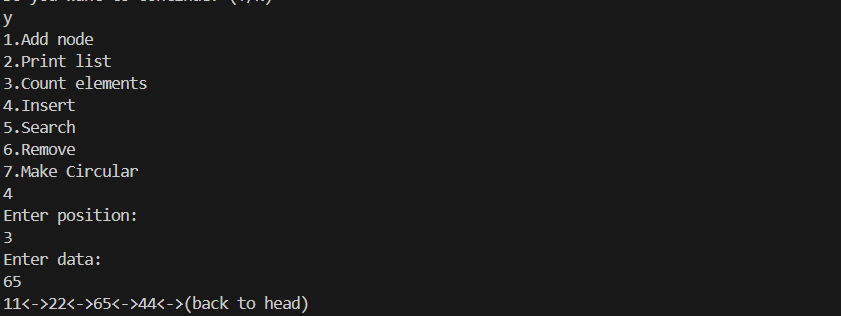
Output :











**Q.4) Polynomial addition using linked list.**

class PolynomialAddition

    static class Node {

        int coefficient;

        int exponent;

        Node next;

        Node(int coeff, int exp) {

            coefficient = coeff;

            exponent = exp;

            next = null;

        }

    }

    public static Node insertTerm(Node head, int coefficient, int exponent) {

        Node newNode = new Node(coefficient, exponent);

        if (head == null || head.exponent < exponent) {

            newNode.next = head;

            return newNode;

        }

        Node temp = head;

        while (temp.next != null && temp.next.exponent > exponent) {

            temp = temp.next;

        }

        if (temp.exponent == exponent) {

            temp.coefficient += coefficient;

        } else {

            newNode.next = temp.next;

            temp.next = newNode;

        }

        return head;

    }

    public static void printPolynomial(Node head) {

        if (head == null) {

            System.out.println("0");

            return;

        }

        Node temp = head;

        while (temp != null) {

            System.out.print(temp.coefficient + "x^" + temp.exponent);

            temp = temp.next;

            if (temp != null) {

                System.out.print(" + ");

            }

        }

        System.out.println();

    }

    public static Node addPolynomials(Node poly1, Node poly2) {

        Node result = null;

        while (poly1 != null && poly2 != null) {

            if (poly1.exponent == poly2.exponent) {

                result = insertTerm(result, poly1.coefficient + poly2.coefficient, poly1.exponent);

                poly1 = poly1.next;

                poly2 = poly2.next;

            } else if (poly1.exponent > poly2.exponent) {

                result = insertTerm(result, poly1.coefficient, poly1.exponent);

                poly1 = poly1.next;

            } else {

                result = insertTerm(result, poly2.coefficient, poly2.exponent);

                poly2 = poly2.next;

            }

        }

        while (poly1 != null) {

            result = insertTerm(result, poly1.coefficient, poly1.exponent);

            poly1 = poly1.next;

        }

        while (poly2 != null) {

            result = insertTerm(result, poly2.coefficient, poly2.exponent);

            poly2 = poly2.next;

        }

        return result;

    }

    public static void main(String[] args) {

        Node poly1 = null;

        Node poly2 = null;

        poly1 = insertTerm(poly1, 3, 4);

        poly1 = insertTerm(poly1, 2, 3);

        poly1 = insertTerm(poly1, 1, 0);

        poly2 = insertTerm(poly2, 5, 3);

        poly2 = insertTerm(poly2, 2, 2);

        poly2 = insertTerm(poly2, 1, 0);

        System.out.print("Polynomial 1: ");

        printPolynomial(poly1);

        System.out.print("Polynomial 2: ");

        printPolynomial(poly2);

        Node result = addPolynomials(poly1, poly2);

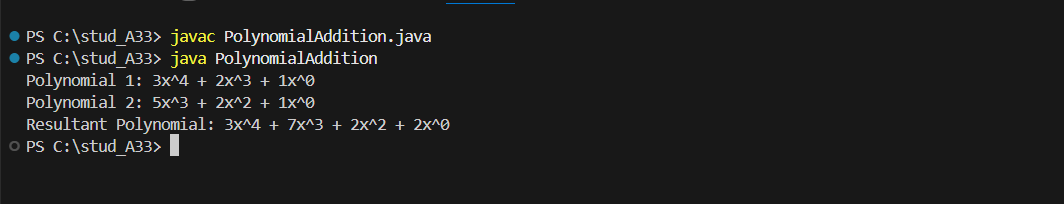
        System.out.print("Resultant Polynomial: ");

        printPolynomial(result);

    }

}

Output :



**Q.5) Linkedlist implementation of stack and ordinary queue.**

1. **Stack**

import java.util.Scanner;

class stack {

private stack bottom;

private stack top;

private int data;

private stack next;

private stack prev;

public void push(int x) {

if (top == null && bottom == null) {

bottom = new stack();

bottom.data = x;

bottom.next = null;

bottom.prev = null;

top = bottom;

} else {

stack newtop = new stack();

newtop.data = x;

top.next = newtop;

newtop.prev = top;

newtop.next = null;

top = newtop;

}

}

public void displayBottomUp() {

stack temp = bottom;

if (top == null && bottom == null) {

System.out.println("stack is empty");

} else {

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

}

public void displayTopToBottom() {

stack temp = top;

if (top == null && bottom == null) {

System.out.println("stack is empty");

} else {

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.prev;

}

System.out.println();

}

}

public int pop() {

int popped;

if (top == null && bottom == null) {

return -1;

} else {

popped = top.data;

top = top.prev;

if (top != null) {

top.next = null;

} else {

bottom = null;

}

return popped;

}

}

}

public class Stack1 {

public static void main(String[] args) {

stack st = new stack();

int option, num;

char ch = 'Y';

Scanner sc = new Scanner(System.in);

System.out.println("choose option");

while (ch == 'Y' || ch == 'y') {

System.out.println("1.To push element into stack. \n2.To display bottomUp. \n3.To display topTobottom. \n4.To pop element");

option = sc.nextInt();

switch (option) {

case 1:

System.out.println("Enter the element");

num = sc.nextInt();

st.push(num);

break;

case 2:

st.displayBottomUp();

break;

case 3:

st.displayTopToBottom();

break;

case 4:

int n = st.pop();

if (n == -1) {

System.out.println("stack is empty");

} else {

System.out.println("popped element is : " + n);

}

break;

default:

System.out.println("Invalid input");

break;

}

System.out.println("Do you want to continue? (Y/N)");

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

if (ch == 'N' || ch == 'n') {

break;

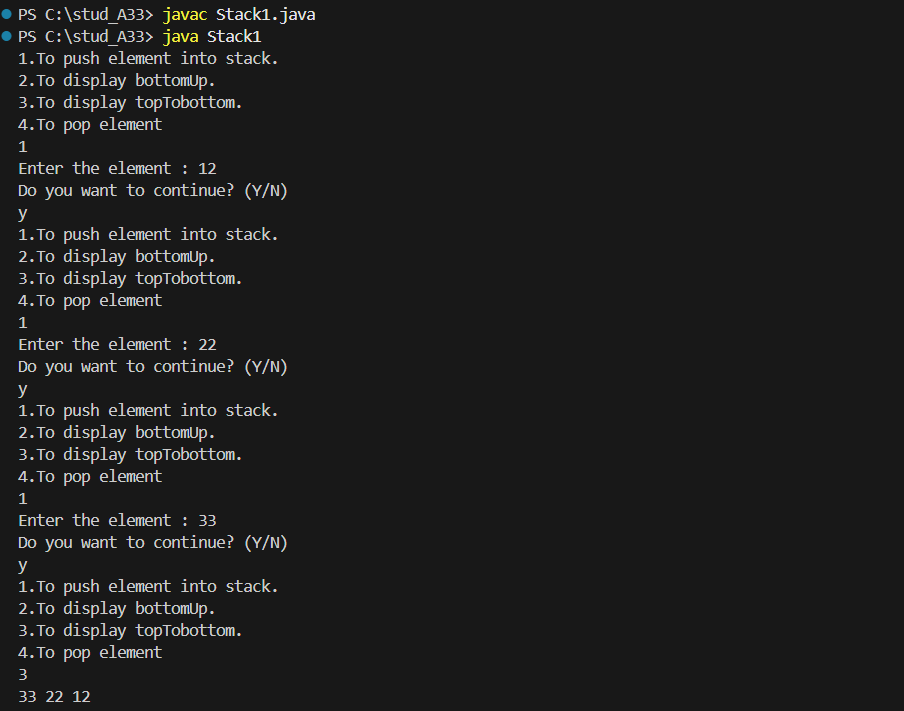
}

}

}

}

Output :





1. **Ordinary queue**

import java.util.Scanner;

class Queue {

int data;

Queue next;

Queue front;

Queue rear;

public boolean isEmpty() {

return (front == null && rear == null);

}

public void insert(int x) {

if (isEmpty()) {

front = rear = new Queue();

rear.data = x;

rear.next = null;

} else {

rear.next = new Queue();

rear.next.data = x;

rear.next.next = null;

rear = rear.next;

}

}

public void display() {

if (isEmpty()) {

System.out.println("No elements");

} else {

Queue temp = front;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

}

public int remove() {

int ans;

if (isEmpty()) {

return -1;

} else {

ans = front.data;

front = front.next;

if (front == null) {

front = rear = null;

}

return ans;

}

}

}

public class Q {

public static void main(String[] args) {

Queue q = new Queue();

int option, num;

char ch = 'Y';

Scanner sc = new Scanner(System.in);

while (ch == 'Y' || ch == 'y') {

System.out.println("1.To insert.\n2.To Display.\n3.To remove");

option = sc.nextInt();

switch (option) {

case 1:

System.out.print("Enter the element : ");

num = sc.nextInt();

q.insert(num);

break;

case 2:

q.display();

break;

case 3:

int n = q.remove();

if (n == -1) {

System.out.println("no elemnts");

} else {

System.out.println("removed element is : " + n);

}

break;

default:

System.out.println("Invalid option. Please try again.");

break;

}

System.out.println("Do you want to continue? (Y/N)");

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

if (ch == 'N' || ch == 'n') {

break;

}

}

}

}

Output:





**Q.4) Implemntation of priority and doubly ended queue using linked list.**

1. **Doubly endend queue.**

import java.util.Scanner;

class dqueue {

int data;

dqueue next;

dqueue prev;

dqueue front;

dqueue rear;

public boolean isEmpty() {

return (front == null && rear == null);

}

public void display() {

dqueue temp = front;

if (isEmpty()) {

System.out.println("No elements");

} else {

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

}

public void insertfront(int x) {

if (isEmpty()) {

front = rear = new dqueue();

front.data = x;

front.next = null;

front.prev = null;

} else {

dqueue newFront = new dqueue();

newFront.data = x;

newFront.next = front;

front.prev = newFront;

newFront.prev = null;

front = newFront;

}

}

public void insertRear(int x) {

if (isEmpty()) {

front = rear = new dqueue();

front.data = x;

front.next = null;

front.prev = null;

} else {

dqueue newRear = new dqueue();

newRear.data = x;

rear.next = newRear;

newRear.prev = rear;

newRear.next = null;

rear = newRear;

}

}

public int removeFront() {

int ans;

if (isEmpty()) {

return -1;

} else {

ans = front.data;

front = front.next;

if (front != null) {

front.prev = null;

} else {

front = rear = null;

}

return ans;

}

}

public int removeRear() {

int ans;

if (isEmpty()) {

return -1;

} else {

ans = rear.data;

rear = rear.prev;

if (rear != null) {

rear.next = null;

} else {

front = rear = null;

}

return ans;

}

}

}

public class DQ1 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int option, num;

char ch = 'Y';

dqueue q = new dqueue();

while (ch == 'Y' || ch == 'y') {

System.out.println("choose option : ");

System.out.println("1.To insert from front.\n2.To Display.\n3.To remove from rear.\n4.To Remove from front.\n5.To insert from rear");

option = sc.nextInt();

switch (option) {

case 1:

System.out.println("Enter the value : ");

num = sc.nextInt();

q.insertfront(num);

break;

case 2:

q.display();

break;

case 3:

int n = q.removeRear();

if (n == -1) {

System.out.println("Queue is empty");

} else {

System.out.println("popped element is : " + n);

}

break;

case 4:

n = q.removeFront();

if (n == -1) {

System.out.println("Queue is empty");

} else {

System.out.println("popped element is : " + n);

}

break;

case 5:

System.out.println("Enter the value : ");

num = sc.nextInt();

q.insertRear(num);

break;

default:

System.out.println("Invalid input");

break;

}

System.out.println("Do you want to continue ?");

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

if (ch == 'N' || ch == 'n') {

break;

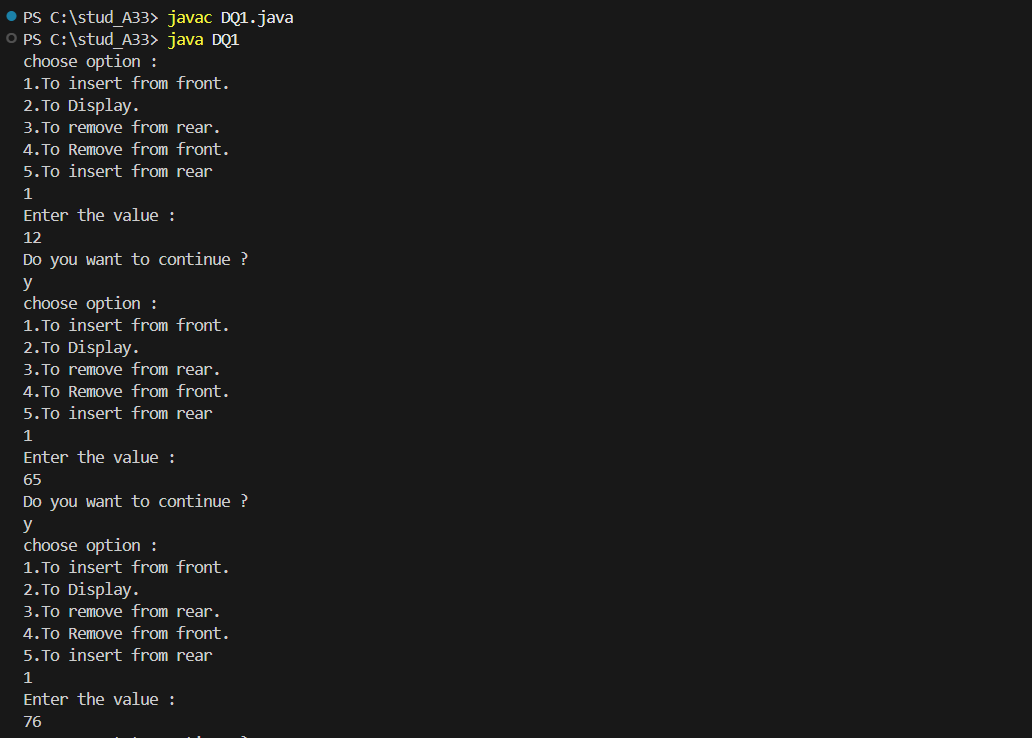
}

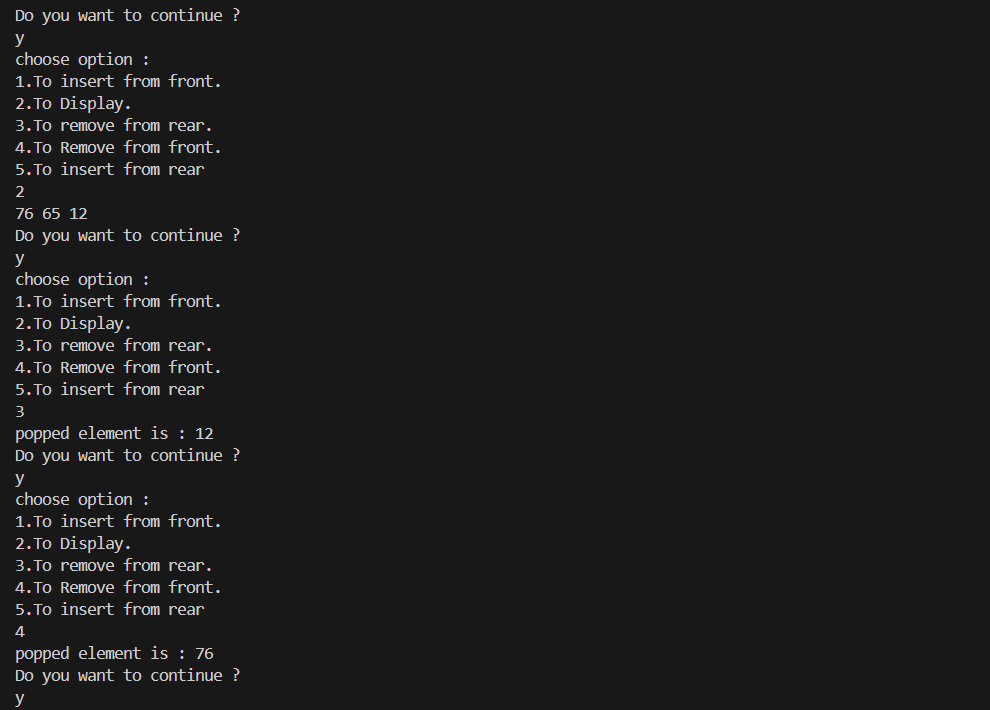
}

}

}

Output :







1. **Priority queue.**

import java.util.Scanner;

class PriorityQueueNode {

int data;

int priority;

PriorityQueueNode next;

public PriorityQueueNode(int data, int priority) {

this.data = data;

this.priority = priority;

this.next = null;

}

}

class PriorityQueue {

PriorityQueueNode front;

public boolean isEmpty() {

return front == null;

}

public void insert(int data, int priority) {

PriorityQueueNode newNode = new PriorityQueueNode(data, priority);

if (isEmpty() || front.priority < priority) {

newNode.next = front;

front = newNode;

} else {

PriorityQueueNode temp = front;

while (temp.next != null && temp.next.priority >= priority) {

temp = temp.next;

}

newNode.next = temp.next;

temp.next = newNode;

}

}

public void display() {

if (isEmpty()) {

System.out.println("No elements");

} else {

PriorityQueueNode temp = front;

while (temp != null) {

System.out.print("[" + temp.data + ", priority: " + temp.priority + "] ");

temp = temp.next;

}

System.out.println();

}

}

public int remove() {

if (isEmpty()) {

return -1;

} else {

int ans = front.data;

front = front.next;

return ans;

}

}

}

public class PQ {

public static void main(String[] args) {

PriorityQueue pq = new PriorityQueue();

int option, num, priority;

char ch = 'Y';

Scanner sc = new Scanner(System.in);

while (ch == 'Y' || ch == 'y') {

System.out.println("1.To Insert\n2.To Display\n3.To Remove");

option = sc.nextInt();

switch (option) {

case 1:

System.out.println("Enter the value:");

num = sc.nextInt();

System.out.println("Enter the priority:");

priority = sc.nextInt();

pq.insert(num, priority);

break;

case 2:

pq.display();

break;

case 3:

int n = pq.remove();

if (n == -1) {

System.out.println("No elements");

} else {

System.out.println("Removed element is: " + n);

}

break;

default:

System.out.println("Invalid option");

break;

}

System.out.println("Do you want to continue? (Y/N)");

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

if (ch == 'N' || ch == 'n') {

break;

}

}

}

}

Output :

