**PRACTICAL NO: 1**

**Aim : A) Implementation of different Searching Techniques.**

**1) Write a program to implement Linear Search.**

**Code:**

import java.util.Scanner;

 public class LinearSearch {

    public static void main(String[] args) {

        int i, x, n;

        int flag = 0;

        Scanner scanner = new Scanner(System.in);

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        System.out.println("How many numbers you want to enter in the array?");

        n = scanner.nextInt();

        // Declare array

        int[] a = new int[n];

        // Input array elements

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

        for (i = 0; i < n; i++) {

            a[i] = scanner.nextInt();

        }

  // Input the number to search for

        System.out.println("Enter number which you want to search:");

        x = scanner.nextInt();

        // Linear search

        for (i = 0; i < n; i++) {

            if (a[i] == x) {

                flag = 1;

                break;

            }

        }

        // Output result

        if (flag == 1) {

            System.out.println("Element Found!");

        } else {

            System.out.println("Element not found!");

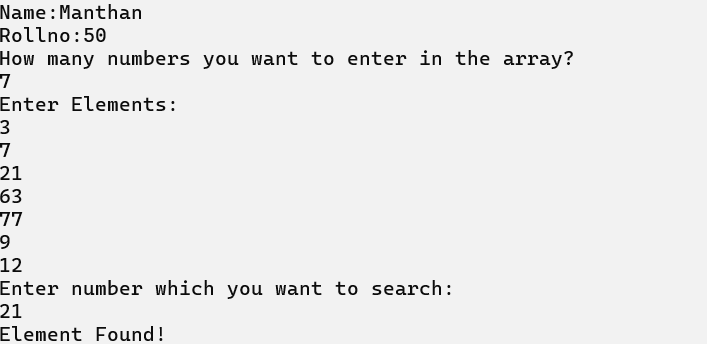
        }

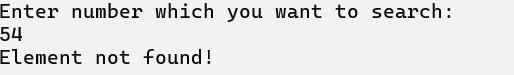
        scanner.close();

    }

 }

**OUTPUT :**





**2) Write a program to implement Binary Search.**

**Code :**

import java.util.Scanner;

 public class BinarySearch {

    // Binary search method

    public static int binarySearch(int[] arr, int l, int r, int x) {

        if (r >= l) {

            int mid = l + (r - l) / 2;

            // Check if x is present at mid

            if (arr[mid] == x) {

                return mid;

            }

            // If x is smaller than mid, search in the left half

            if (arr[mid] > x) {

                return binarySearch(arr, l, mid - 1, x);

            }

            // If x is larger than mid, search in the right half

            return binarySearch(arr, mid + 1, r, x);

        }

        // Return -1 if the element is not present in the array

        return -1;

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        // Initialize the array

        int[] arr = new int[5];

        // Taking user input for array elements

        for (int i = 0; i < 5; i++) {

            System.out.print("Enter element " + (i + 1) + ": ");

            arr[i] = scanner.nextInt();

        }

        // Displaying the elements

        System.out.println("Elements are:");

        for (int i = 0; i < 5; i++) {

            System.out.println(arr[i]);

        }

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

        int x = scanner.nextInt();

        // Calling binary search and displaying the result

        int n = arr.length;

        int result = binarySearch(arr, 0, n - 1, x);

        if (result == -1) {

            System.out.println("Element is not present in the array");

        } else {

            System.out.println("Element is present at index " + result);

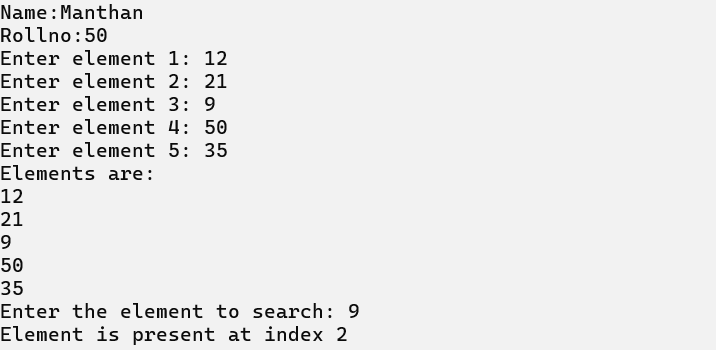
        }

        scanner.close();

    }

 }

**OUTPUT :**



**Aim : B) Implementation of different Sorting Techniques.**

**1) Write a program to implement Bubble Sort.**

**Code:**

import java.util.Scanner;

 public class BubbleSort {

    public static void main(String[] args) {

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        Scanner scanner = new Scanner(System.in);

        System.out.println("How many numbers you want to enter?");

        int n = scanner.nextInt();

        int[] a = new int[n];

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

        for (int i = 0; i < n; i++) {

            a[i] = scanner.nextInt();

        }

        // Bubble sort algorithm

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - i - 1; j++) {

                if (a[j] > a[j + 1]) {

                    // Swap elements

                    int temp = a[j];

                    a[j] = a[j + 1];

                    a[j + 1] = temp;

                }

            }

        }

        // Print sorted array with spaces

        System.out.println("Sorted array is:");

        for (int i = 0; i < n; i++) {

            System.out.print(a[i]);

            if (i != n - 1) {

          System.out.print(" ");

// Print space after every number except the last one

            }

        }

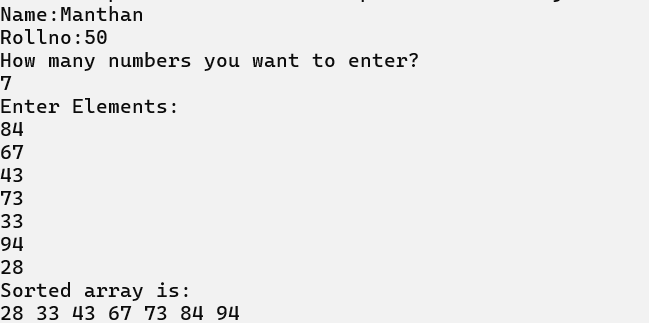
        System.out.println(); // Add a newline after the sorted array

        scanner.close();

    }

 }

**OUTPUT :**



**2) Write a program to implement Insertion Sort.**

**Code :**

 import java.util.Scanner;

 public class InsertionSort {

    // Method to display the array

    public static void display(int[] array, int size) {

        for (int i = 0; i < size; i++) {

            System.out.print(array[i] + " ");

        }

        System.out.println();

    }

    // Insertion sort algorithm

    public static void insertionSort(int[] array, int size) {

        int key, j;

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

            key = array[i];

            j = i;

            // Move elements of array[0..i-1] that are greater than key to one position ahead

            while (j > 0 && array[j - 1] > key) {

                array[j] = array[j - 1];

                j--;

            }

            array[j] = key;

        }

    }

    public static void main(String[] args) {

        // Print roll number at the start of the output

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        Scanner scanner = new Scanner(System.in);

        // Read the number of elements

        System.out.print("Enter the number of elements: ");

        int n = scanner.nextInt();

        // Create the array to hold the elements

        int[] arr = new int[n];

        // Read the elements into the array

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

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextInt();

        }

        // Display the array before sorting

        System.out.print("Array before Sorting: ");

        display(arr, n);

        // Perform the insertion sort

        insertionSort(arr, n);

        // Display the array after sorting

        System.out.print("Array after Sorting: ");

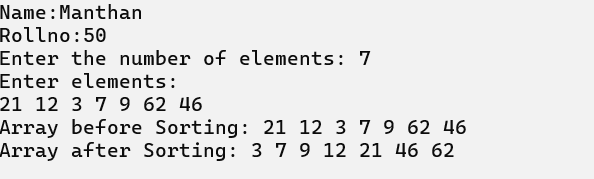
        display(arr, n);

        scanner.close();

    }

 }

**OUTPUT :**

****

**3) Write a program to implement Selection Sort.**

**Code :**

import java.util.Scanner;

 public class SelectionSort {

    // Method to display the array

    public static void display(int[] array, int size) {

        for (int i = 0; i < size; i++) {

            System.out.print(array[i] + " ");

        }

        System.out.println();

    }

    // Method to perform selection sort

    public static void selectionSort(int[] array, int size) {

        int imin, temp;

        for (int i = 0; i < size - 1; i++) {

            imin = i;

            for (int j = i + 1; j < size; j++) {

                if (array[j] < array[imin]) {

                    imin = j;

                }

            }

            // Swap the elements

            temp = array[i];

            array[i] = array[imin];

            array[imin] = temp;

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // This line ensures your roll number is printed at the start

        System.out.println("Name:Manthan");

        System.out.println("Rolno:50");

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

        int n = sc.nextInt();

        int[] arr = new int[n];

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

        // Accepting array elements from the user

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        // Display the array before sorting

        System.out.print("Array before Sorting: ");

        display(arr, n);

        // Performing selection sort

        selectionSort(arr, n);

        // Display the array after sorting

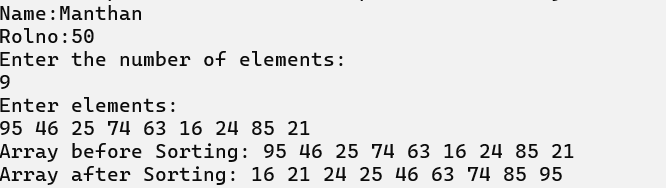
        System.out.print("Array after Sorting: ");

        display(arr, n);

    }

 }

**OUTPUT :**



**4) Write a program to implement Shell Sort.**

**Code:**

 import java.util.Scanner;

 public class ShellSort {

    public static void main(String[] args) {

      System.out.println("Name:Manthan");

      System.out.println("Rollno:50");

      Scanner scanner = new Scanner(System.in);

        // Read the size of the array

        System.out.print("Enter the size of the array: ");

        int n = scanner.nextInt();

        // Create the array and read its elements

        int[] a = new int[n];

        System.out.print("Enter the elements of the array: ");

        for (int i = 0; i < n; i++) {

            a[i] = scanner.nextInt();

        }

        // Print array before sorting

        System.out.print("Array elements before sorting: ");

        for (int i = 0; i < n; i++) {

            System.out.print(a[i] + " ");

        }

        System.out.println();

        // Shell Sort Logic

        for (int gap = n / 2; gap > 0; gap /= 2) {

            for (int i = gap; i < n; i++) {

                int temp = a[i];

                int j = i;

                // Move elements of the array that are greater than temp to the gap distance ahead

                while (j >= gap && a[j - gap] > temp) {

                    a[j] = a[j - gap];

                    j -= gap;

                }

                a[j] = temp;

            }

        }

        // Print array after sorting

        System.out.print("Array after sorting: ");

        for (int i = 0; i < n; i++) {

            System.out.print(a[i] + " ");

        }

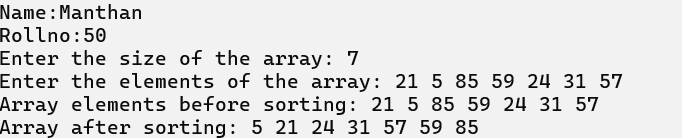
        // Close the scanner

        scanner.close();

    }

 }

**OUTPUT :**



**PRACTICAL NO: 2**

**Aim : Perform various Hashing techniques with Linear Probe as collision.**

**1)Write a program to implement Hashing using Linear Probing.**

**Code :**

 import java.util.Scanner;

 public class HashingLinearProbing {

    // Function to perform linear probing search in the hash table

    public static int hashSearch(int[] hashTable, int x, int n) {

        int index = x % n;

        int start = index; // To prevent infinite loop if the entire table is searched

        if (hashTable[index] == x) {

            return index;

        } else if (hashTable[index] == -1) {

            return -1;

        } else {

            do {

                index = (index + 1) % n; // Linear probing

                if (hashTable[index] == x) {

                    return index;

                } else if (hashTable[index] == -1) {

                    break;

                }

            } while (index != start);

            return -1; // Element not found

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int[] hashTable = new int[10]; // Hash table with size 10

        int i, x, index, k;

        // Initialize the hash table with -1 (indicating empty slots)

        for (i = 0; i < 10; i++) {

            hashTable[i] = -1;

        }

        // Input 7 elements and insert them into the hash table using linear probing

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        System.out.println("Hashing using linear probing\n");

        System.out.println("Hash table creation\n");

        for (i = 1; i <= 7; i++) {

            System.out.print("\nEnter a number: ");

            x = scanner.nextInt();

            index = x % 10; // Hash function (modulo)

            // Linear probing to find an empty slot

            while (hashTable[index] != -1) {

                index = (index + 1) % 10;

            }

            hashTable[index] = x; // Insert the element at the found index

        }

        // Display the hash table

        System.out.println("\nHASH TABLE");

        for (i = 0; i < 10; i++) {

            System.out.print(i + " ");

        }

System.out.println();

        for (i = 0; i < 10; i++) {

            System.out.print(hashTable[i] + " ");

        }

        // Search loop

        do {

            System.out.print("\nElement to be searched (enter -1 to stop): ");

            x = scanner.nextInt();

            if (x >= 0) {

                k = hashSearch(hashTable, x, 10);

                if (k >= 0) {

                    System.out.println(x + " is present at hash[" + k + "]");

                } else {

                    System.out.println(x + " is not present");

                }

            }

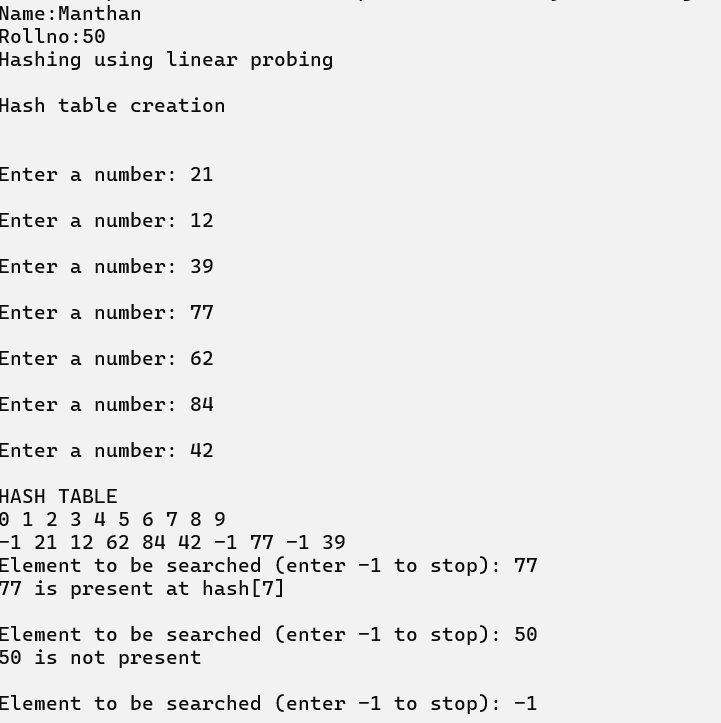
        } while (x >= 0);

         scanner.close();

    }

 }

**OUTPUT :**

****

**PRACTICAL NO: 3**

**Aim : Implementation Using Array :**

**1) Write a program to implement Stack Using Array.**

**Code :**

import java.util.Scanner;

 class Stack {

    private int[] stk = new int[5]; // Stack array of size 5

    private int top = -1;           // Keeps track of the top index of the stack

    // Push method to add an element to the stack

    public void push(int x) {

        if (top >= 4) {

            System.out.println("Stack is full.");

            return;

        }

        stk[++top] = x;  // Increment top and add element

        System.out.println("Element Inserted");

    }

    // Pop method to remove an element from the stack

    public void pop() {

        if (top < 0) {

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

            return;

        }

        System.out.println("Deleted " + stk[top--]);

    }

    // Display method to print the elements in the stack

    public void display() {

        if (top < 0) {

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

            return;

        }

        System.out.println("Inserted Numbers are:");

        for (int i = top; i >= 0; i--) {

            System.out.println(stk[i]);

        }

    }

 }

 public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        Stack st = new Stack(); // Create Stack object

        int ch;

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        System.out.println("Select the option:");

        while (true) {

            System.out.println("\n1. Push");

            System.out.println("2. Pop");

            System.out.println("3. Display");

            System.out.println("4. Exit");

            System.out.print("Enter Your choice: ");

            ch = sc.nextInt();

            switch (ch) {

                case 1:

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

                    int x = sc.nextInt();

                    st.push(x);

                    break;

                case 2:

                    st.pop();

                    break;

                case 3:

                    st.display();

                    break;

                case 4:

                    System.exit(0);

                    break;

                default:

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

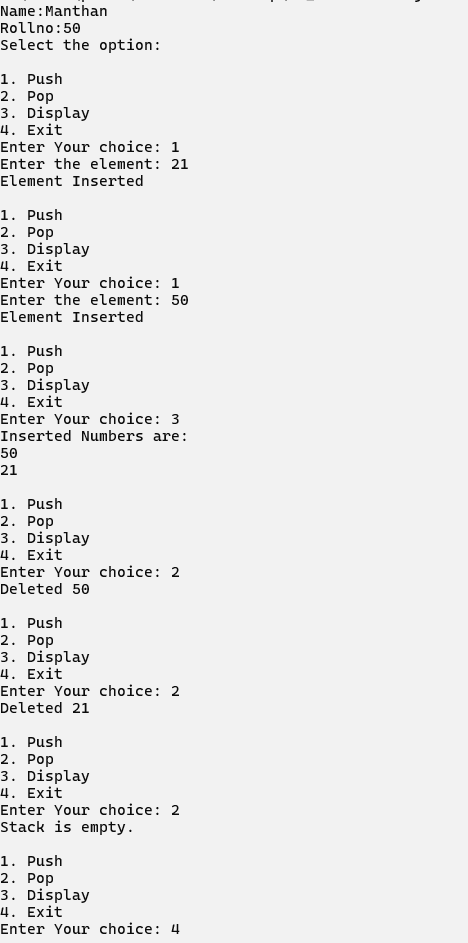
            }

        }

    }

 }

**OUTPUT :**

****

**2) Write a program to implement Ordinary Queue Using Array.**

**Code :**

import java.util.Scanner;

class Queue {

    private int front, rear;

    private int[] q;

    private static final int max = 10;

    public Queue() {

        front = rear = -1;

        q = new int[max];

    }

    public void enqueue(int x) {

        if (rear == max - 1) {

            System.out.println("Queue is Full!");

        } else {

            if (front == -1 && rear == -1) {

                front = rear = 0;

            } else {

                rear++;

            }

            q[rear] = x;

            System.out.println(x + " enqueued to queue");

        }

    }

    public int dequeue() {

        int x = -1;

        if (front == -1) {

            System.out.println("Queue is Empty!");

        } else {

            if (front == rear) {

                x = q[front];

                front = rear = -1;

            } else {

                x = q[front];

                front++;

            }

        }

        return x;

    }

    public void peek() {

        if (front == -1) {

            System.out.println("Queue is Empty!");

        } else {

            System.out.print("Queue Elements are: ");

            for (int i = front; i <= rear; i++) {

                System.out.print(q[i] + " ");

            }

            System.out.println();

        }

    }

}

public class Main1 {

    public static void main(String[] args) {

        // Display roll number at the top of the output

        System.out.println("Name:Manthan");

        System.out.println("R0llno:50");

        Queue q = new Queue();

        Scanner scanner = new Scanner(System.in);

        int ch, x;

        do {

            System.out.print("1. Enqueue 2. Dequeue 3. Display Queue 4. Exit: ");

            ch = scanner.nextInt();

            switch (ch) {

                case 1:

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

                    x = scanner.nextInt();

                    q.enqueue(x);

                    break;

                case 2:

                    x = q.dequeue();

                    if (x != -1) {

                        System.out.println(x + " dequeued!");

                    }

                    break;

                case 3:

                    q.peek();

                    break;

                case 4:

                    return;

                default:

                    System.out.println("Invalid choice. Exiting.");

                    return;

            }

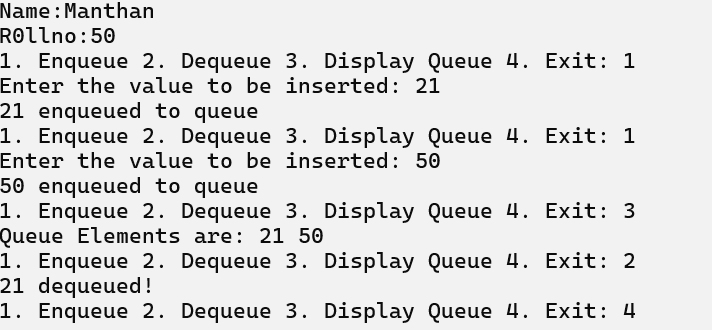
        } while (ch != 4);

        scanner.close();

    }

}

**OUTPUT :**

****

**3) Write a program to implement Circular Queue Using Array.**

**Code :**

import java.util.Scanner;

 public class CircularQueue {

    static int[] cqueue = new int[5];

    static int front = -1, rear = -1, n = 5;

    // Method to insert an element in the queue

    public static void insertCQ(int val) {

        if ((front == 0 && rear == n - 1) || (front == rear + 1)) {

            System.out.println("Queue Overflow");

            return;

        }

        if (front == -1) {

            front = 0;

            rear = 0;

        } else {

            if (rear == n - 1)

                rear = 0;

            else

                rear = rear + 1;

        }

        cqueue[rear] = val;

    }

    // Method to delete an element from the queue

    public static void deleteCQ() {

        if (front == -1) {

            System.out.println("Queue Underflow");

            return;

        }

        System.out.println("Element deleted from queue is: " + cqueue[front]);

        if (front == rear) {

            front = -1;

            rear = -1;

        } else {

            if (front == n - 1)

                front = 0;

            else

                front = front + 1;

        }

    }

    // Method to display elements of the queue

    public static void displayCQ() {

        int f = front, r = rear;

        if (front == -1) {

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

            return;

        }

        System.out.println("Queue elements are:");

        if (f <= r) {

            while (f <= r) {

                System.out.print(cqueue[f] + " ");

                f++;

            }

        } else {

            while (f <= n - 1) {

                System.out.print(cqueue[f] + " ");

                f++;

            }

            f = 0;

            while (f <= r) {

                System.out.print(cqueue[f] + " ");

                f++;

            }

        }

        System.out.println();

    }

    public static void main(String[] args) {

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

 Scanner scanner = new Scanner(System.in);

        int ch, val;

        do {

            System.out.println("1) Insert");

            System.out.println("2) Delete");

            System.out.println("3) Display");

            System.out.println("4) Exit");

            System.out.print("Enter choice: ");

            ch = scanner.nextInt();

            switch (ch) {

                case 1:

                    System.out.print("Input for insertion: ");

                    val = scanner.nextInt();

                    insertCQ(val);

                    break;

                case 2:

                    deleteCQ();

                    break;

                case 3:

                    displayCQ();

                    break;

                case 4:

                    System.out.println("Exit");

                    break;

                default:

                    System.out.println("Incorrect choice!");

            }

        } while (ch != 4);

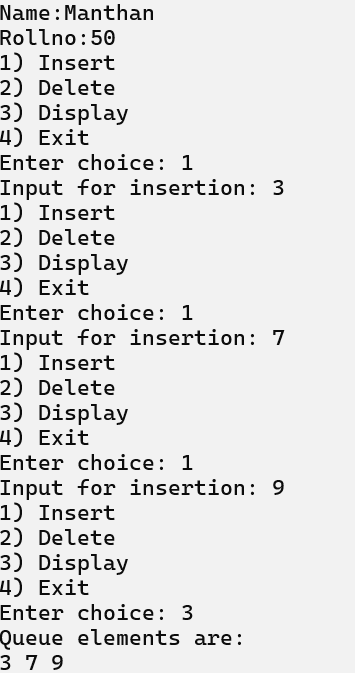
        scanner.close();

    }

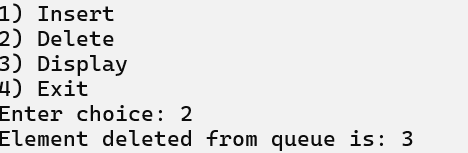
 }

**OUTPUT :**

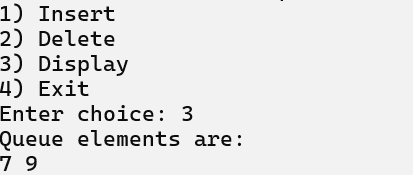
**1) Insert**

****

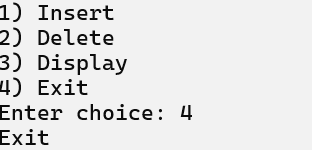
**2) Delete**

****

**3) Display**

****

**4) Exit**

****

**PRACTICAL NO : 4**

**Aim : Implementation of Stack Applications.**

**1) Write a program to implement Infix to Postfix.**

**Code :**

import java.util.Scanner;

import java.util.Stack;

public class InfixToPostfixConverter {

    // Method to check if a character is an operator

    private static boolean isOperator(char c) {

        return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

    }

    // Method to check if a character is an operand

    private static boolean isOperand(char c) {

        return Character.isLetterOrDigit(c);

    }

    // Method to get the precedence of an operator

    private static int precedence(char c) {

        switch (c) {

            case '+':

            case '-':

                return 1;

            case '\*':

            case '/':

                return 2;

            case '^':

                return 3;

            default:

                return -1;

        }

    }

    // Method to convert infix expression to postfix

    public static String infixToPostfix(String infix) {

        Stack<Character> stack = new Stack<>();

        StringBuilder postfix = new StringBuilder();

        for (int i = 0; i < infix.length(); i++) {

            char current = infix.charAt(i);

            // If the character is an operand, add it to the postfix expression

            if (isOperand(current)) {

                postfix.append(current);

            }

            // If the character is '(', push it onto the stack

            else if (current == '(') {

                stack.push(current);

            }

            // If the character is ')', pop from the stack until '(' is encountered

            else if (current == ')') {

                while (!stack.isEmpty() && stack.peek() != '(') {

                    postfix.append(stack.pop());

                }

                if (!stack.isEmpty() && stack.peek() == '(') {

                    stack.pop(); // Remove '('

                }

            }

            // If the character is an operator

            else if (isOperator(current)) {

                while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(current)) {

                    postfix.append(stack.pop());

                }

                stack.push(current);

            }

        }

        // Pop all the remaining operators from the stack

        while (!stack.isEmpty()) {

            postfix.append(stack.pop());

        }

        return postfix.toString();

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Input from user

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        System.out.println("Enter an infix expression:");

        String infixExpression = scanner.nextLine();

        // Display infix expression

        System.out.println("Infix Expression: " + infixExpression);

        // Convert to postfix

        String postfixExpression = infixToPostfix(infixExpression);

        // Display postfix expression

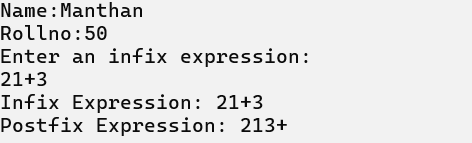
        System.out.println("Postfix Expression: " + postfixExpression);

        scanner.close();

    }

}

**OUTPUT :**

****

**2) Write a program to implement Postfix Evaluation.**

**Code :**

import java.util.Stack;

 public class PostEval {

    // Method to evaluate postfix expression

    public static int evaluatePostfix(String expr) {

        Stack<Integer> stack = new Stack<>();

        // Loop through each character in the postfix expression

        for (int i = 0; i < expr.length(); i++) {

            char ch = expr.charAt(i);

            // If the character is a digit, push it onto the stack

            if (Character.isDigit(ch)) {

                stack.push(ch - '0'); // Convert char to int

            } else {

                // Pop two elements from the stack and apply the operator

                int b = stack.pop();

                int a = stack.pop();

                // Perform the operation based on the operator

                switch (ch) {

                    case '+':

                        stack.push(a + b);

                        break;

                    case '-':

                        stack.push(a - b);

                        break;

                    case '\*':

                        stack.push(a \* b);

                        break;

                    case '/':

                        if (b != 0) {

                            stack.push(a / b); // Avoid division by zero

                        } else {

                            System.out.println("Division by zero error");

                            return -1; // Exit on division by zero error

                        }

                        break;

                    default:

                        System.out.println("Invalid operator: " + ch);

                        return -1; // Exit on invalid operator

                }

            }

        }

        // The final result will be at the top of the stack

        return stack.pop();

    }

    public static void main(String[] args) {

System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        java.util.Scanner scanner = new java.util.Scanner(System.in);

        System.out.print("Enter Postfix Expression: ");

        String expr = scanner.next();

        int result = evaluatePostfix(expr);

        if (result != -1) {

            System.out.println("Result: " + result);

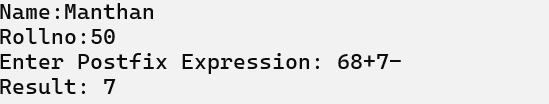
        }

        scanner.close();

    }

 }

**OUTPUT :**

****

**3) Write a program to implement Balancing of Parenthesis.**

**Code :**

import java.util.Scanner;

import java.util.Stack;

public class BalancedExpression {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Name:Manthan");

System.out.println("\nRollno:50\nEnter expression :");

String expression = scanner.nextLine();

if (isBalanced(expression)) {

System.out.println("Balanced");

} else {

System.out.println("Unbalanced");

}

scanner.close();

}

private static boolean isBalanced(String expression) {

Stack<Character> stack = new Stack<>();

for (char ch : expression.toCharArray()) {

if (ch == '{' || ch == '[' || ch == '(') {

stack.push(ch);

} else {

switch (ch) {

case ')':

if (stack.isEmpty() || stack.pop() != '(') {

return false;

}

break;

case ']':

if (stack.isEmpty() || stack.pop() != '[') {

return false;

}

break;

case '}':

if (stack.isEmpty() || stack.pop() != '{') {

return false;

}

break;

default:

System.out.println("Enter the correct choice");

return false;

}

}

}

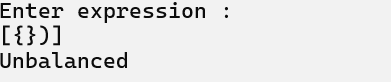
return stack.isEmpty();

}

}

**OUTPUT :**

****

****

**PRACTICAL NO : 5**

**Aim : Implementation of all types of Linked Lists.**

**1) Write a program to implement Singly Linked List.**

**Code :**

import java.util.Scanner;

class SinglyLinkedList {

// Node class for the linked list

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

private Node start = null;

// Insert at the beginning

public void insertAtBeg(int x) {

Node newNode = new Node(x);

if (start == null) {

start = newNode;

} else {

newNode.next = start;

start = newNode;

}

}

// Insert at the end

public void insertAtEnd(int x) {

Node newNode = new Node(x);

if (start == null) {

start = newNode;

} else {

Node p = start;

while (p.next != null) {

p = p.next;

}

p.next = newNode;

}

}

// Insert at specific position

public void insertAtPos(int x, int pos) {

Node newNode = new Node(x);

if (start == null) {

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

return;

}

Node p = start;

int count = 1;

// Traverse to position

while (p != null && count < pos - 1) {

p = p.next;

count++;

}

if (p == null || count < pos - 1) {

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

} else {

newNode.next = p.next;

p.next = newNode;

}

}

// Search for a value

public void searchPos(int value) {

if (start == null) {

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

return;

}

Node p = start;

int count = 1;

while (p != null) {

if (p.data == value) {

System.out.println("Value found at position " + count + ".");

return;

}

p = p.next;

count++;

}

System.out.println("Value not found.");

}

// Delete a node at a specific position

public void del(int pos) {

if (start == null) {

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

return;

}

if (pos == 1) {

start = start.next;

return;

}

Node p = start;

int count = 1;

while (p != null && count < pos - 1) {

p = p.next;

count++;

}

if (p == null || p.next == null) {

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

} else {

p.next = p.next.next;

}

}

// Sort the list

public void sort() {

if (start == null) {

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

return;

}

Node ptr = start;

while (ptr != null) {

Node p = ptr.next;

while (p != null) {

if (ptr.data > p.data) {

int temp = ptr.data;

ptr.data = p.data;

p.data = temp;

}

p = p.next;

}

ptr = ptr.next;

}

}

// Reverse the list

public void reverse() {

if (start == null) {

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

return;

}

if (start.next == null) {

System.out.println("Only one element in the list.");

return;

}

Node prev = null;

Node current = start;

Node next = null;

while (current != null) {

next = current.next;

current.next = prev;

prev = current;

current = next;

}

start = prev;

System.out.println("List reversed.");

}

// Display the list

public void display() {

if (start == null) {

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

return;

}

Node p = start;

System.out.println("\nSingly Linked List:");

while (p != null) {

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

p = p.next;

}

System.out.println();

}

}

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

SinglyLinkedList list = new SinglyLinkedList();

System.out.println("Name:Manthan");

System.out.println("Rollno:50");

int ch, x, pos;

while (true) {

System.out.println("\n1. Insert at beginning");

System.out.println("2. Insert at end");

System.out.println("3. Insert at position");

System.out.println("4. Delete");

System.out.println("5. Search");

System.out.println("6. Display");

System.out.println("7. Sort");

System.out.println("8. Reverse");

System.out.println("9. Exit");

System.out.print("Enter your choice: ");

ch = sc.nextInt();

switch (ch) {

case 1:

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

x = sc.nextInt();

list.insertAtBeg(x);

list.display();

break;

case 2:

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

x = sc.nextInt();

list.insertAtEnd(x);

list.display();

break;

case 3:

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

x = sc.nextInt();

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

pos = sc.nextInt();

list.insertAtPos(x, pos);

list.display();

break;

case 4:

System.out.print("Enter the position to delete: ");

pos = sc.nextInt();

list.del(pos);

list.display();

break;

case 5:

System.out.print("Enter the value to search: ");

x = sc.nextInt();

list.searchPos(x);

break;

case 6:

list.display();

break;

case 7:

list.sort();

list.display();

break;

case 8:

list.reverse();

list.display();

break;

case 9:

sc.close();

System.out.println("Exiting program.");

return;

default:

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

}

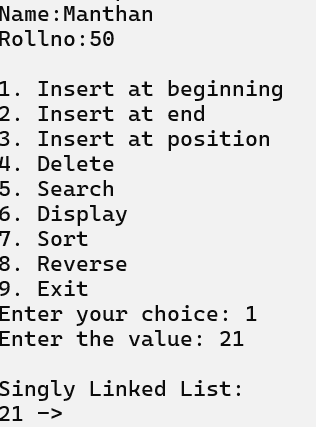
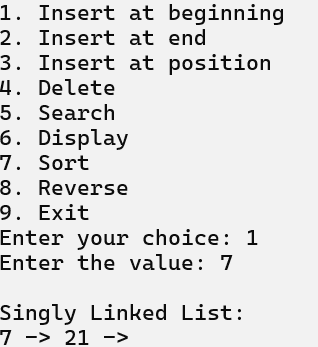
}

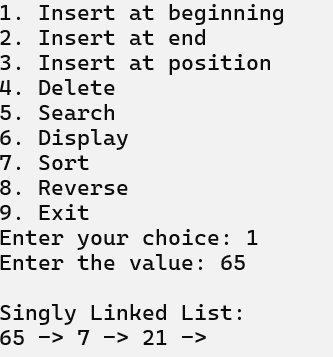
}

}

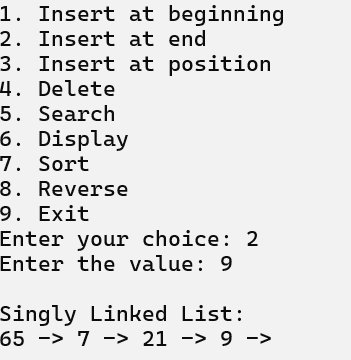
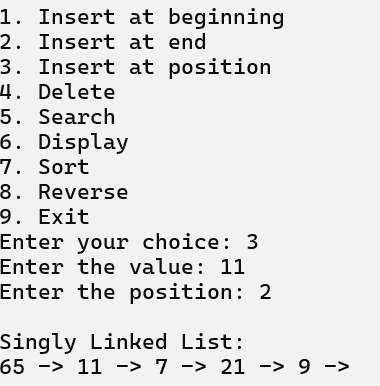
**OUTPUT :**

**1. Insert at beginning:**

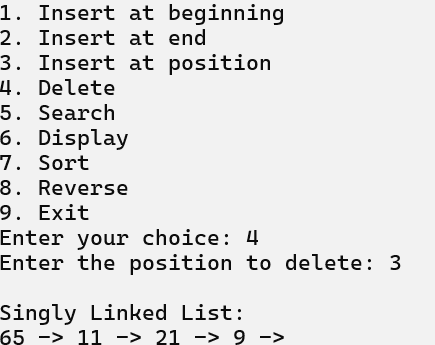
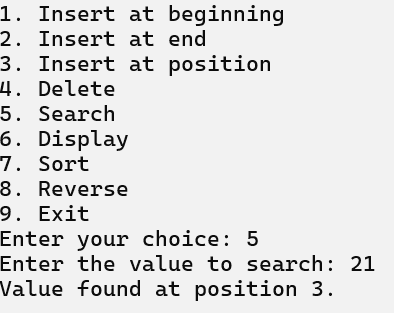
** **

****

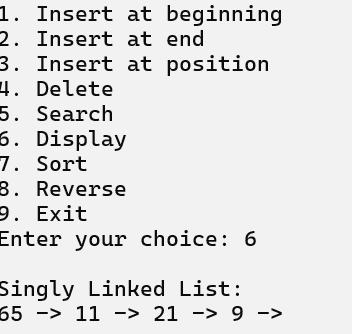
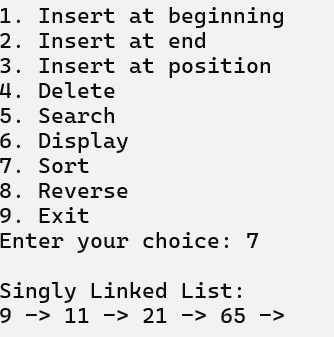
**2. Insert at end : 3. Insert at position :**

** **

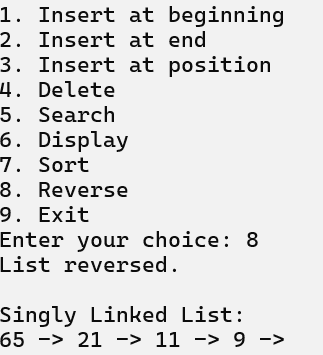
**4. Delete : 5. Search :**

** **

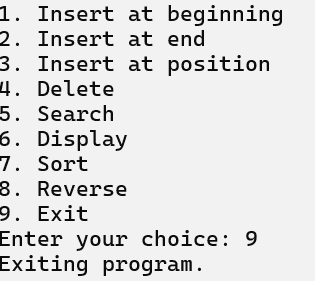
**6. Display : 7. Sort :**

** **

**8. Reverse :**

****

**9.** **Exit :**

****

**2) Write a program to implement Circular Linked List.**

**Code :**

import java.util.Scanner;

 class SinglyCircularLinkedList {

    private Node last = null;

    private int count = 0;

    // Node class for the circular linked list

    class Node {

        int data;

        Node next;

        Node(int data) {

            this.data = data;

            this.next = null;

        }

    }

    // Create the list with a single node

    public void create(int x) {

        Node tmp = new Node(x);

        if (last == null) {

            last = tmp;

            tmp.next = last; // Points to itself, forming a circle

        } else {

            tmp.next = last.next;

            last.next = tmp;

            last = tmp; // Update the last node to the new node

        }

    }

    // Add element at the beginning

    public void addAtBegin(int x) {

        if (last == null) {

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

            return;

        }

        Node tmp = new Node(x);

        tmp.next = last.next;

        last.next = tmp;

    }

    // Add element after a given position

    public void addAfter(int x, int pos) {

        if (last == null) {

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

            return;

        }

        Node p = last.next;

        for (int i = 0; i < pos - 1; i++) {

            p = p.next;

            if (p == last.next) {

                System.out.println("Position does not exist.");

                return;

            }

        }

        Node tmp = new Node(x);

        tmp.next = p.next;

        p.next = tmp;

        if (p == last) {

            last = tmp; // Update last if added at the end

        }

    }

    // Delete a node with the given value

    public void del(int x) {

        if (last == null) {

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

            return;

        }

        Node p = last.next;

        if (last.next == last && last.data == x) {

            last = null; // Single node case

            return;

        }

        if (p.data == x) { // If first node needs to be deleted

            last.next = p.next;

            return;

        }

        while (p.next != last) {

            if (p.next.data == x) {

                p.next = p.next.next;

                if (p.next == last) {

                    last = p; // Update last if last node is deleted

                }

                return;

            }

            p = p.next;

        }

        if (p.next.data == x) { // Deleting the last node

            p.next = last.next;

            last = p;

        }

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

    }

    // Search for an element and print its position

    public void search(int x) {

        int pos = 1;

        if (last == null) {

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

            return;

        }

        Node p = last.next;

        while (p != last) {

            if (p.data == x) {

                System.out.println("Element found at position " + pos + ".");

                return;

            }

            p = p.next;

            pos++;

        }

        if (last.data == x) {

            System.out.println("Element found at position " + pos + ".");

        } else {

            System.out.println("Item not found.");

        }

    }

    // Sort the list using bubble sort

    public void sort() {

        if (last == null) {

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

            return;

        }

        Node p = last.next;

        Node ptr = null;

        int temp;

        while (p != last) {

            ptr = p.next;

            while (ptr != last.next) {

                if (p.data > ptr.data) {

                    temp = p.data;

                    p.data = ptr.data;

                    ptr.data = temp;

                }

                ptr = ptr.next;

            }

            p = p.next;

        }

    }

    // Count the number of elements in the list

    public void count() {

        if (last == null) {

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

            return;

        }

        Node p = last.next;

        int count = 0;

        do {

            count++;

            p = p.next;

        } while (p != last.next);

        System.out.println("Number of elements: " + count);

    }

    // Display the list

    public void display() {

        if (last == null) {

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

            return;

        }

        Node p = last.next;

        System.out.println("\nSingly Circular Linked List:");

        do {

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

            p = p.next;

        } while (p != last.next);

        System.out.println("(head)");

    }

 }

 public class Main3 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        SinglyCircularLinkedList list = new SinglyCircularLinkedList();

        int ch, x, pos;

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        while (true) {

            System.out.println("1. Create a list");

            System.out.println("2. Add at begin");

            System.out.println("3. Add after");

            System.out.println("4. Search");

            System.out.println("5. Sort");

            System.out.println("6. Count");

            System.out.println("7. Display");

            System.out.println("8. Delete");

            System.out.println("9. Exit");

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

            ch = sc.nextInt();

            switch (ch) {

                case 1:

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

                    x = sc.nextInt();

                    list.create(x);

                    list.display();

                    break;

                case 2:

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

                    x = sc.nextInt();

                    list.addAtBegin(x);

                    list.display();

                    break;

                case 3:

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

                    pos = sc.nextInt();

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

                    x = sc.nextInt();

                    list.addAfter(x, pos);

                    list.display();

                    break;

                case 4:

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

                    x = sc.nextInt();

                    list.search(x);

                    break;

                case 5:

                    System.out.println("Before sorting:");

                    list.display();

                    list.sort();

                    System.out.println("After sorting:");

                    list.display();

                    break;

                case 6:

                    list.count();

                    break;

                case 7:

                    list.display();

                    break;

                case 8:

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

                    x = sc.nextInt();

                    list.del(x);

                    list.display();

                    break;

                case 9:

                    sc.close();

                    return;

                default:

                    System.out.println("Wrong choice.");

            }

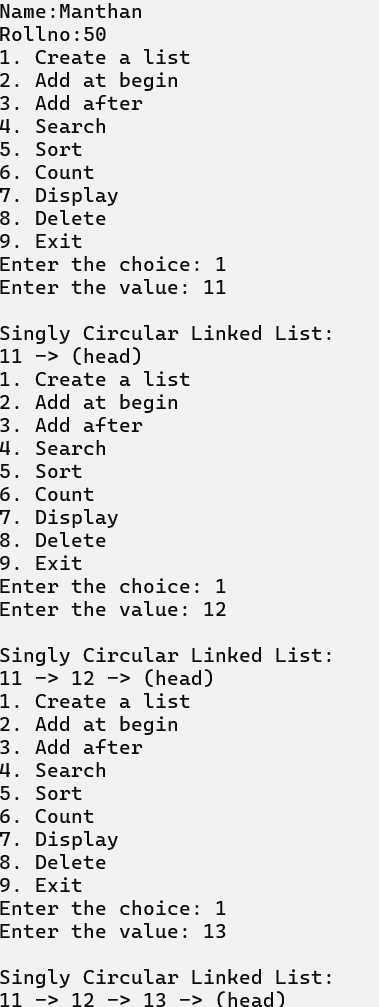
        }

    }

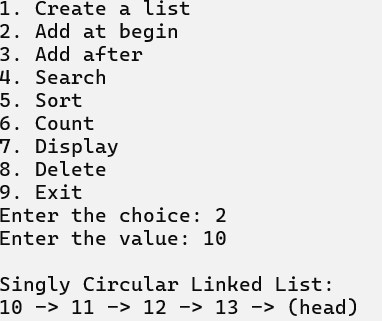
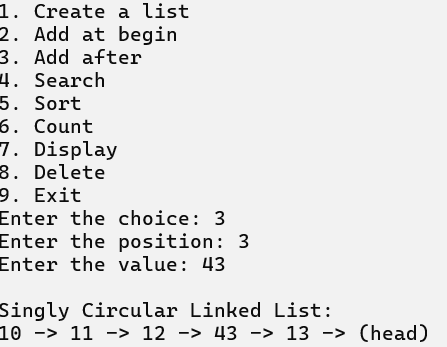
 }

**OUTPUT :**

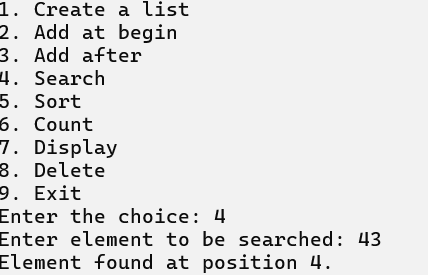
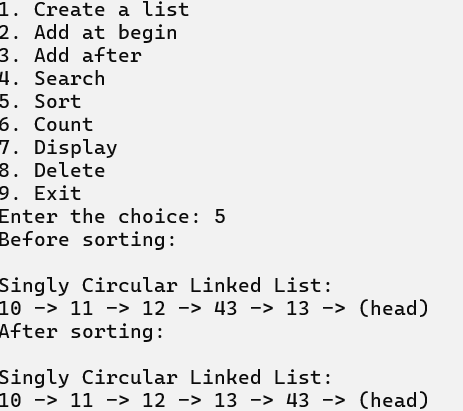
**1. Create a List :**

****

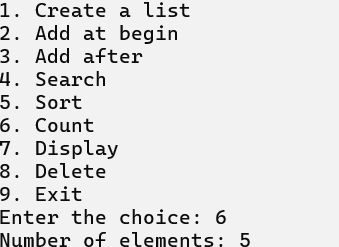
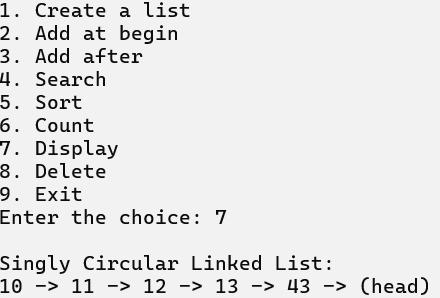
**2. Add at Begin : 3. Add After :**

** **

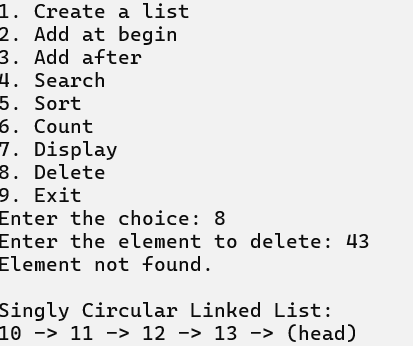
**4. Search : 5. Sort :**

** **

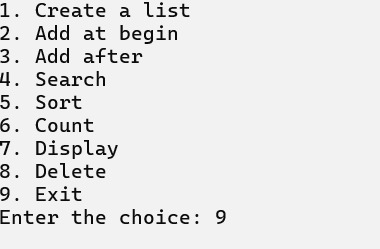
**6. Count : 7. Display :**

** **

**8. Delete :**

****

**9. Exit :**

****

**3) Write a program to implement Doubly Linked List.**

**Code :**

import java.util.Scanner;

 public class DoublyLinkedList {

    static class Node {

        int data;

        Node next;

        Node prev;

        Node(int data) {

            this.data = data;

            this.next = null;

            this.prev = null;

        }

    }

    private Node start = null;

    // Create a list

    public void create(int x) {

        Node tmp = new Node(x);

        if (start == null) {

            start = tmp;

        } else {

            Node p = start;

            while (p.next != null) {

                p = p.next;

            }

            p.next = tmp;

            tmp.prev = p;

        }

    }

    // Add at the beginning

    public void addAtBegin(int x) {

        if (start == null) {

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

        } else {

            Node tmp = new Node(x);

            tmp.next = start;

            start.prev = tmp;

            start = tmp;

        }

    }

    // Add after a given position

    public void addAfter(int x, int pos) {

        if (start == null) {

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

        } else {

            Node p = start;

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

                if (p == null) {

                    System.out.println("Position does not exist.");

                    return;

                }

                p = p.next;

            }

            Node tmp = new Node(x);

            tmp.next = p.next;

            if (p.next != null) {

                p.next.prev = tmp;

            }

            p.next = tmp;

            tmp.prev = p;

        }

    }

    // Delete an element

    public void delete(int x) {

        if (start == null) {

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

            return;

        }

        // Delete first element

        if (start.data == x) {

            Node tmp = start;

            start = start.next;

            if (start != null) start.prev = null;

            tmp = null;

            return;

        }

        // Delete middle or last element

        Node p = start;

        while (p != null && p.next != null) {

            if (p.next.data == x) {

                Node tmp = p.next;

                p.next = tmp.next;

                if (tmp.next != null) {

                    tmp.next.prev = p;

                }

                tmp = null;

                return;

            }

            p = p.next;

        }

        // If we reach here, the element was not found

        if (p != null && p.data == x) {

            Node tmp = p;

            if (p.prev != null) p.prev.next = null;

            tmp = null;

        }

    }

    // Reverse the list

    public void reverse() {

        Node p1 = start;

        Node p2 = (p1 != null) ? p1.next : null;

        p1.next = null;

        if (p1 != null) p1.prev = p2;

        while (p2 != null) {

            p2.prev = p2.next;

            p2.next = p1;

            p1 = p2;

            p2 = p2.prev;

        }

        start = p1;

        System.out.println("List reversed.");

    }

    // Count elements in the list

    public void count() {

        Node p = start;

        int cnt = 0;

        while (p != null) {

            cnt++;

            p = p.next;

        }

        System.out.println("Number of elements are " + cnt + ".");

    }

    // Search for an element

    public void search() {

        Scanner sc = new Scanner(System.in);

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

        int value = sc.nextInt();

        if (start == null) {

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

            return;

        }

        Node p = start;

        int count = 0;

        while (p != null) {

            count++;

            if (p.data == value) {

                System.out.println("Element found at position " + count + ".");

                return;

            }

            p = p.next;

        }

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

    }

    // Sort the list

    public void sort() {

        if (start == null) {

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

            return;

        }

        Node ptr = start;

        while (ptr != null) {

            Node p = ptr.next;

            while (p != null) {

                if (ptr.data > p.data) {

                    int temp = ptr.data;

                    ptr.data = p.data;

                    p.data = temp;

                }

                p = p.next;

            }

            ptr = ptr.next;

        }

    }

    // Display the list

    public void display() {

        if (start == null) {

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

            return;

        }

        Node p = start;

        System.out.print("\nDoubly Linked List: ");

        while (p != null) {

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

            p = p.next;

        }

        System.out.println("null\n");

    }

    public static void main(String[] args) {

        DoublyLinkedList d = new DoublyLinkedList();

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        Scanner sc = new Scanner(System.in);

        int x, pos, ch;

        while (true) {

            System.out.println("1. Create a list");

            System.out.println("2. Add at begin");

            System.out.println("3. Add after");

            System.out.println("4. Search");

            System.out.println("5. Reverse");

            System.out.println("6. Count");

            System.out.println("7. Sort");

            System.out.println("8. Display");

            System.out.println("9. Delete");

            System.out.println("10. Exit");

            System.out.print("Enter your choice: ");

            ch = sc.nextInt();

            switch (ch) {

                case 1:

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

                    x = sc.nextInt();

                    d.create(x);

                    d.display();

                    break;

                case 2:

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

                    x = sc.nextInt();

                    d.addAtBegin(x);

                    d.display();

                    break;

                case 3:

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

                    pos = sc.nextInt();

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

                    x = sc.nextInt();

                    d.addAfter(x, pos);

                    d.display();

                    break;

                case 4:

                    d.search();

                    d.display();

                    break;

                case 5:

                    d.reverse();

                    d.display();

                    break;

                case 6:

                    d.count();

                    d.display();

                    break;

                case 7:

                    System.out.println("Before sorting:");

                    d.display();

                    d.sort();

                    System.out.println("After sorting:");

                    d.display();

                    break;

                case 8:

                    d.display();

                    break;

                case 9:

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

                    x = sc.nextInt();

                    d.delete(x);

                    d.display();

                    break;

                case 10:

                    sc.close();

                    System.exit(0);

                    break;

                default:

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

            }

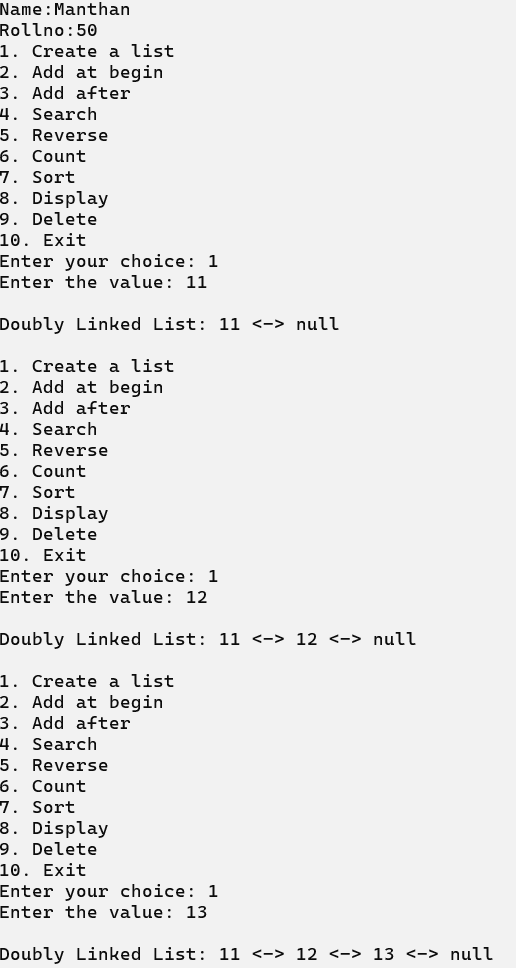
        }

    }

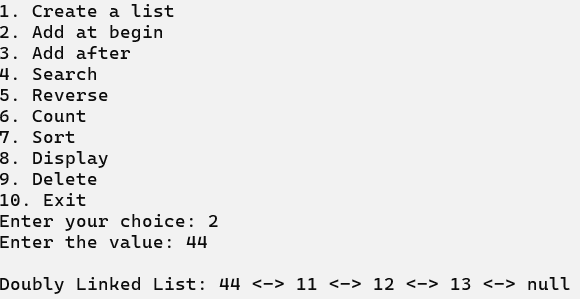
 }

**OUTPUT :**

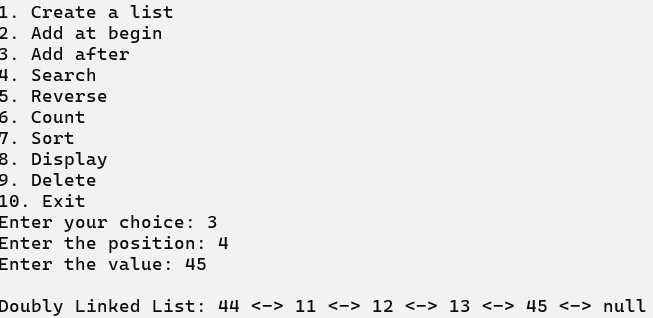
**1. Create a list:**

****

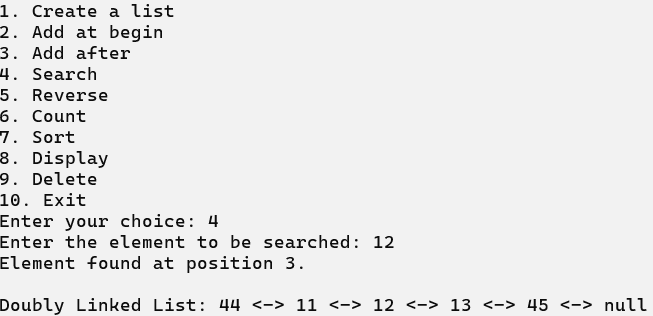
**2. Add at Begin :**

****

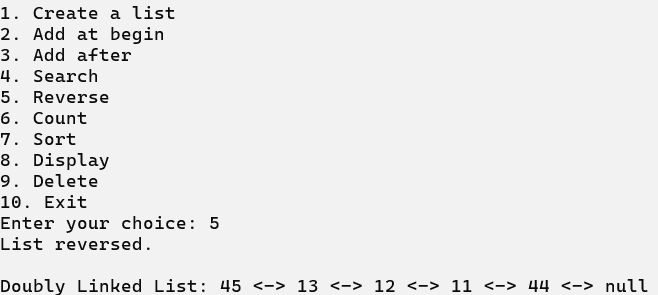
**3. Add After :**

****

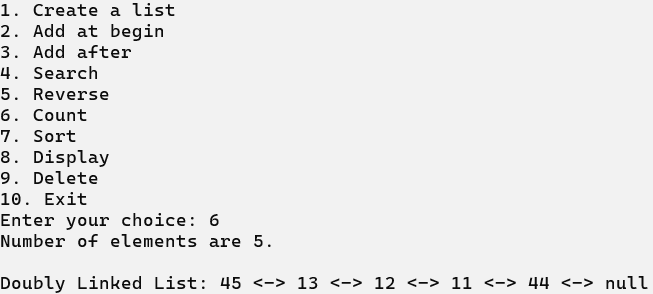
**4. Search :**

****

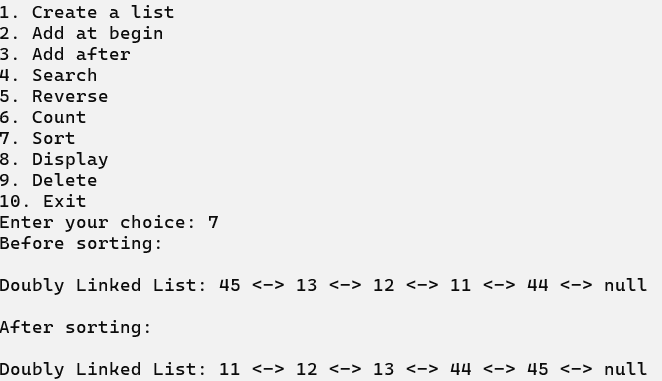
**5. Reverse :**

****

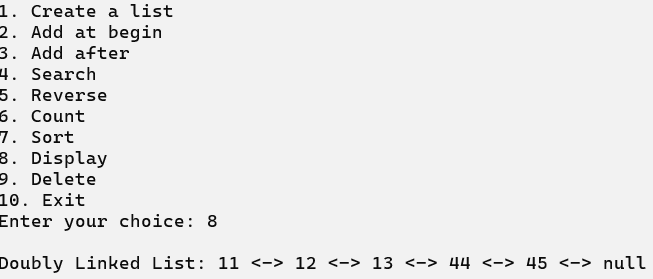
**6. Count :**

****

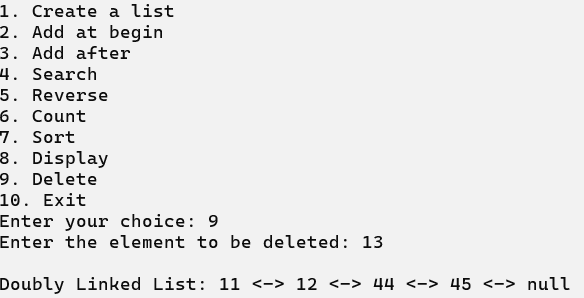
**7. Sort :**

****

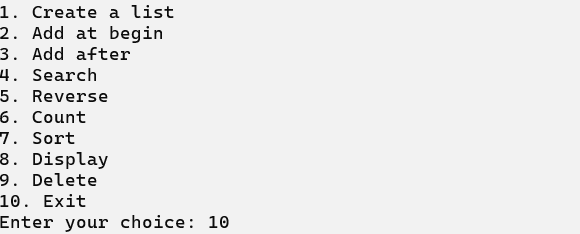
**8. Display :**

****

**9. Delete :**

****

**10. Exit :**

****

**4) Write a program to implement Polynomial Addition.**

**Code :**

public class PolynomialAddition {

    // Method to find the maximum of two integers

    public static int max(int m, int n) {

        return (m > n) ? m : n;

    }

    // Method to add two polynomials

    public static int[] add(int[] A, int[] B, int m, int n) {

        int size = max(m, n);

        int[] sum = new int[size];

        // Copying elements from A[] to sum[]

        for (int i = 0; i < m; i++) {

            sum[i] = A[i];

        }

        // Adding elements from B[] to sum[]

        for (int i = 0; i < n; i++) {

            sum[i] += B[i];

        }

        return sum;

    }

    // Method to print the polynomial

    public static void printPoly(int[] poly, int n) {

        for (int i = 0; i < n; i++) {

            System.out.print(poly[i]);

            if (i != 0) {

                System.out.print("x^" + i);

            }

            if (i != n - 1) {

                System.out.print(" + ");

            }

        }

        System.out.println();

    }

    public static void main(String[] args) {

        // Student details

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        System.out.println("\nPOLYNOMIAL ADDITION\n");

        // First polynomial A = 1 + 0x + 2x^2 + 4x^3

        int[] A = {1, 0, 2, 4};

        // Second polynomial B = 3 + 5x + 7x^2

        int[] B = {3, 5, 7};

        // Sizes of A[] and B[]

        int m = A.length;

        int n = B.length;

        // Printing the first polynomial

        System.out.println("First polynomial is ");

        printPoly(A, m);

        // Printing the second polynomial

        System.out.println("Second polynomial is ");

        printPoly(B, n);

        // Adding polynomials A[] and B[] and getting the sum polynomial

        int[] sum = add(A, B, m, n);

        int size = max(m, n);

        // Printing the sum of polynomials

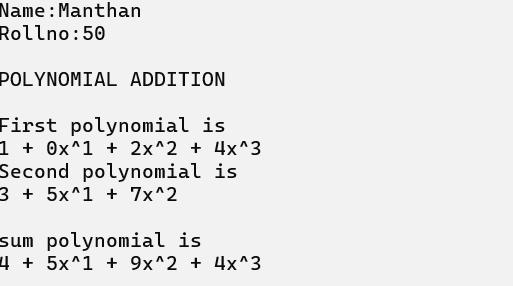
        System.out.println("\nsum polynomial is ");

        printPoly(sum, size);

    }

 }

**OUTPUT :**

****

**PRACTICAL NO : 6**

**Aim : Demonstrate application of linked list.**

**1) Write a program to implement Stack Using Linked List.**

**Code :**

class Stack {

    // Define the Node class

    static class Node {

        int data;

        Node link;

    }

    // Define the top of the stack

    static Node top;

    // Method to push data onto the stack

    public static void push(int data) {

        Node temp = new Node();

        temp.data = data;

        // If memory is not available

        if (temp == null) {

            System.out.println("\nHeap Overflow");

            System.exit(1);

        }

        temp.link = top;

        top = temp;

    }

    // Method to check if the stack is empty

    public static boolean isEmpty() {

        return top == null;

    }

    // Method to peek at the top element of the stack

    public static int peek() {

        if (!isEmpty()) {

            return top.data;

        } else {

            System.exit(1); // Exit if stack is empty

            return -1; // This line will never be reached

        }

    }

    // Method to pop data from the stack

    public static void pop() {

        if (top == null) {

            System.out.println("\nStack Underflow");

            System.exit(1);

        } else {

            Node temp = top;

            top = top.link;

            temp.link = null;

        }

    }

    // Method to display the stack

    public static void display() {

        if (top == null) {

            System.out.println("\nStack Underflow");

            System.exit(1);

        } else {

            Node temp = top;

            while (temp != null) {

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

                temp = temp.link;

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        // Push some elements to the stack

        push(12);

        push(32);

        push(52);

        push(72);

        // Display the stack

        display();

        // Display the top element of the stack

        System.out.println("\nTop element is " + peek());

        // Pop two elements from the stack

        pop();

        pop();

        // Display the stack after popping

        display();

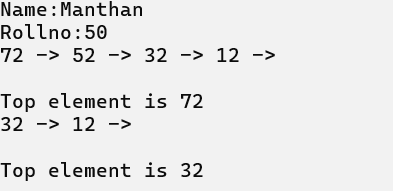
        // Display the top element of the stack after popping

        System.out.println("\nTop element is " + peek());

    }

}

**OUTPUT :**

****

**2) Write a program to implement Ordinary Queue Using Linked List.**

**Code :**

import java.util.Scanner;

 class Node {

 int data;

 Node next;

 Node(int value) {

 data = value;

 next = null;

 }

 }

 class Queue {

 private Node front;

 private Node rear;

 public Queue() {

 front = rear = null;

 }

 public void enqueue(int value) {

 Node newNode = new Node(value);

 if (rear == null) {

 front = rear = newNode;

 } else {

 rear.next = newNode;

 rear = newNode;

}

 System.out.println(value + " enqueued to queue");

 }

 public int dequeue() {

 if (front == null) {

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

 return -1;

 }

 Node temp = front;

 int value = front.data;

 front = front.next;

 if (front == null) {

 rear = null;

 }

 return value;

 }

 public int peek() {

 if (front == null) {

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

 return -1;

 }

 return front.data;

 }

 public void display() {

if (front == null) {

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

 return;

 }

 Node current = front;

 System.out.print("Queue contents: ");

 while (current != null) {

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

 current = current.next;

 }

 System.out.println();

 }

 }

 public class Main4 {

 public static void main(String[] args) {

 Queue q = new Queue();

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

 Scanner scanner = new Scanner(System.in);

 int ch, value;

 do {

 System.out.println("\nQueue Operations Menu:");

 System.out.println("1. Enqueue 2. Dequeue 3. Display 4. Exit");

 System.out.print("Enter your choice: ");

ch = scanner.nextInt();

 switch (ch) {

 case 1:

 System.out.print("Enter value to enqueue: ");

 value = scanner.nextInt();

 q.enqueue(value);

 break;

 case 2:

 value = q.dequeue();

 if (value != -1)

 System.out.println(value + " dequeued from queue");

 break;

 case 3:

 q.display();

 break;

 case 4:

 System.exit(0);

 default:

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

 }

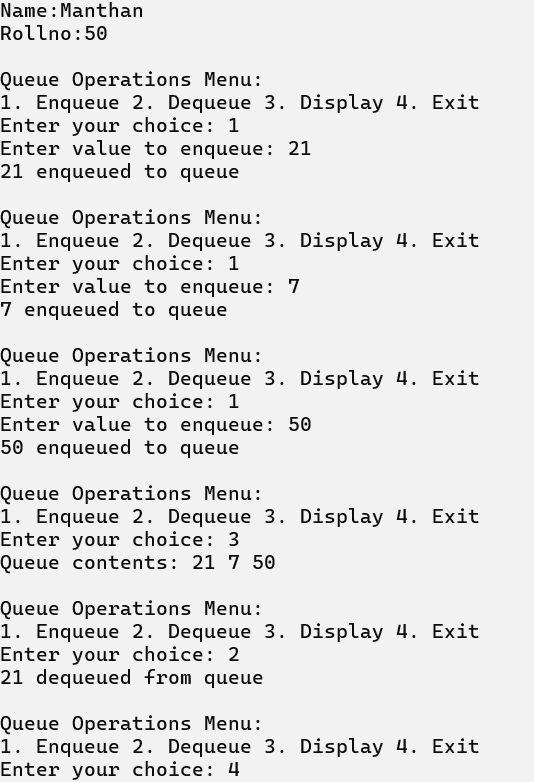
 } while (ch != 4);

 scanner.close();

 }

 }

**OUTPUT :**

****

**3) Write a program to implement Priority Queue Using Linked List.**

**Code :**

import java.util.Scanner;

class Node {

int data;

int priority;

Node next;

Node(int data, int priority) {

this.data = data;

this.priority = priority;

this.next = null;

}

}

class PriorityQueue {

private Node front;

public PriorityQueue() {

front = null;

}

public void enqueue(int value, int priority) {

Node newNode = new Node(value, priority);

if (front == null || priority < front.priority) {

newNode.next = front;

front = newNode;

} else {

Node current = front;

while (current.next != null && current.next.priority <= priority) {

current = current.next;

}

newNode.next = current.next;

current.next = newNode;

}

System.out.println(value + " enqueued to queue with priority " + priority);

}

public int dequeue() {

if (front == null) {

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

return -1;

}

Node temp = front;

int value = front.data;

front = front.next;

temp = null; // Help with garbage collection

return value;

}

public int peek() {

if (front == null) {

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

return -1;

}

return front.data;

}

public void display() {

if (front == null) {

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

return;

}

Node current = front;

System.out.print("Priority Queue contents (data: priority): ");

while (current != null) {

System.out.print("(" + current.data + ": " + current.priority + ") ");

current = current.next;

}

System.out.println();

}

}

public class Main5 {

public static void main(String[] args) {

PriorityQueue pq = new PriorityQueue();

System.out.println("Name:Manthan");

System.out.println("Rollno:50");

Scanner scanner = new Scanner(System.in);

int ch, value, priority;

do {

System.out.println("\nPriority Queue Operations Menu:");

System.out.println("1. Enqueue 2. Dequeue 3. Display 4. Exit");

System.out.print("Enter your choice: ");

ch = scanner.nextInt();

switch (ch) {

case 1:

System.out.print("Enter value to enqueue: ");

value = scanner.nextInt();

System.out.print("Enter priority (lower number = higher priority): ");

priority = scanner.nextInt();

pq.enqueue(value, priority);

break;

case 2:

value = pq.dequeue();

if (value != -1) {

System.out.println(value + " dequeued from queue");

}

break;

case 3:

pq.display();

break;

case 4:

System.exit(0);

default:

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

}

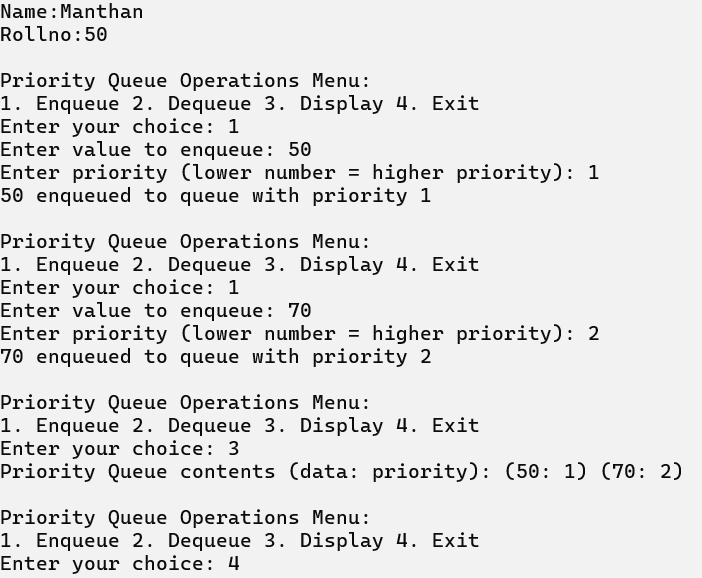
} while (ch != 4);

scanner.close();

}

}

**OUTPUT :**

****

**4) Write a program to implement Double Ended Queue Using Linked List. Code :**

import java.util.Scanner;

 class Node {

 int data;

 Node next;

 Node prev;

 Node(int value) {

 data = value;

 next = null;

 prev = null;

 }

 }

 class Deque {

 private Node front;

 private Node rear;

 public Deque() {

 front = rear = null;

 }

 public void insertFront(int value) {

 Node newNode = new Node(value);

 newNode.next = front;

 if (front != null) {

 front.prev = newNode;

}

 front = newNode;

 if (rear == null) {

 rear = newNode;

 }

 System.out.println(value + " inserted at front");

 }

 public void insertRear(int value) {

 Node newNode = new Node(value);

 newNode.next = null;

 newNode.prev = rear;

 if (rear != null) {

 rear.next = newNode;

 }

 rear = newNode;

 if (front == null) {

 front = newNode;

 }

 System.out.println(value + " inserted at rear");

 }

 public int deleteFront() {

 if (front == null) {

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

 return -1;

}

 int value = front.data;

 front = front.next;

 if (front != null) {

 front.prev = null;

 } else {

 rear = null;

 }

 return value;

 }

 public int deleteRear() {

 if (rear == null) {

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

 return -1;

 }

 int value = rear.data;

 rear = rear.prev;

 if (rear != null) {

 rear.next = null;

 } else {

 front = null;

 }

 return value;

 }

 public int getFront() {

if (front == null) {

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

 return -1;

 }

 return front.data;

 }

 public int getRear() {

 if (rear == null) {

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

 return -1;

 }

 return rear.data;

 }

 public void display() {

 if (front == null) {

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

 return;

 }

 Node current = front;

 System.out.print("Deque contents: ");

 while (current != null) {

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

 current = current.next;

 }

System.out.println();

 }

 }

 public class Main6 {

 public static void main(String[] args) {

 Deque dq = new Deque();

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

 Scanner scanner = new Scanner(System.in);

 int ch, value;

 System.out.println("\nDeque Operations Menu:");

 System.out.println("1. Insert Front 2. Insert Rear 3. Delete Front");

 System.out.println("4. Delete Rear 5. Display 6. Exit");

 do {

 System.out.print("Enter your choice: ");

 ch = scanner.nextInt();

 switch (ch) {

 case 1:

 System.out.print("Enter value to insert at front: ");

 value = scanner.nextInt();

 dq.insertFront(value);

 break;

 case 2:

 System.out.print("Enter value to insert at rear: ");

 value = scanner.nextInt();

dq.insertRear(value);

 break;

 case 3:

 value = dq.deleteFront();

 if (value != -1) {

 System.out.println(value + " deleted from front");

 }

 break;

 case 4:

 value = dq.deleteRear();

 if (value != -1) {

 System.out.println(value + " deleted from rear");

 }

 break;

 case 5:

 dq.display();

 break;

 case 6:

 System.exit(0);

 default:

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

 }

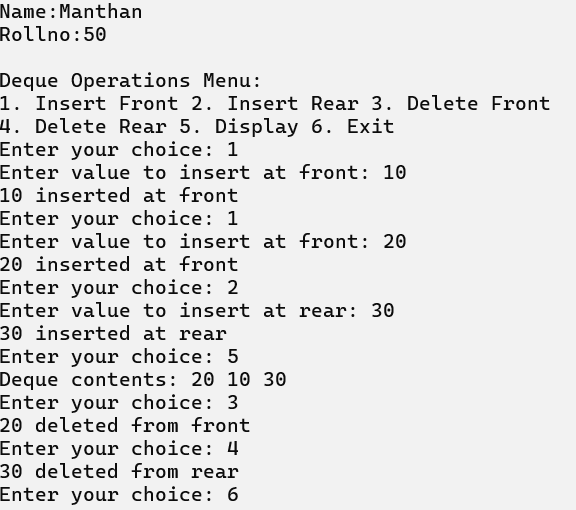
 } while (ch != 6);

 scanner.close();

 }

}

**OUTPUT :**

****

**PRACTICAL NO : 7**

**Aim : Create and perform various operations on BST.**

**1) Write a program to implement Binary Search Tree (BST).**

**Code :**

import java.util.Scanner;

 class BinarySearchTree {

    // Node structure for the tree

    class Node {

        int data;

        Node left, right;

        public Node(int item) {

            data = item;

            left = right = null;

        }

    }

    private Node tree;

    public BinarySearchTree() {

        tree = null;

    }

    // Method to create a tree (insert node)

    public Node createTree(Node node, int item) {

        if (node == null) {

            node = new Node(item);

        } else {

            if (node.data > item) {

                node.left = createTree(node.left, item);

            } else {

                node.right = createTree(node.right, item);

            }

        }

        return node;

    }

    // Preorder traversal

    public void preorder(Node node) {

        if (node != null) {

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

            preorder(node.left);

            preorder(node.right);

        }

    }

    // Inorder traversal

    public void inorder(Node node) {

        if (node != null) {

            inorder(node.left);

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

            inorder(node.right);

        }

    }

    // Postorder traversal

    public void postorder(Node node) {

        if (node != null) {

            postorder(node.left);

            postorder(node.right);

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

        }

    }

    // Count total nodes in the tree

    public int totalNodes(Node node) {

        if (node == null)

            return 0;

        return totalNodes(node.left) + totalNodes(node.right) + 1;

    }

    // Find the smallest node

    public void findSmallestNode(Node node) {

        if (node == null || node.left == null)

            System.out.println(node.data);

        else

            findSmallestNode(node.left);

    }

    // Find the largest node

    public void findLargestNode(Node node) {

        if (node == null || node.right == null)

            System.out.println(node.data);

        else

            findLargestNode(node.right);

    }

    // Main method to handle user input

    public static void main(String[] args) {

        BinarySearchTree obj = new BinarySearchTree();

        Scanner scanner = new Scanner(System.in);

        int choice, n, item;

 System.out.println("Name:Manhan");

 System.out.println("Rollno:50");

        while (true) {

            System.out.println("\nBinary search tree common operation");

            System.out.println("1) Create Tree");

            System.out.println("2) Traversal");

            System.out.println("3) Total Nodes");

            System.out.println("4) Insert Nodes");

            System.out.println("5) Find Smallest Node");

            System.out.println("6) Find Largest Node");

            System.out.println("7) Exit");

            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            switch (choice) {

                case 1:

                    System.out.print("\nCreating Tree----");

                    System.out.print("\nHow many nodes do you want to enter: ");

                    n = scanner.nextInt();

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

                        System.out.print("Enter value: ");

                        item = scanner.nextInt();

                        obj.tree = obj.createTree(obj.tree, item);

                    }

                    break;

                case 2:

                    System.out.println("\nInorder Traversal:");

                    obj.inorder(obj.tree);

                    System.out.println("\nPreorder Traversal:");

                    obj.preorder(obj.tree);

                    System.out.println("\nPostorder Traversal:");

                    obj.postorder(obj.tree);

                    break;

                case 3:

                    int total = obj.totalNodes(obj.tree);

                    System.out.println("Total nodes: " + total);

                    break;

                case 4:

                    System.out.print("\nInsert node in a tree \nEnter value: ");

                    item = scanner.nextInt();

                    obj.tree = obj.createTree(obj.tree, item);

                    System.out.println("\nItem is inserted.");

                    break;

                case 5:

                    System.out.println("\nSmallest node is:");

                    obj.findSmallestNode(obj.tree);

                    break;

                case 6:

                    System.out.println("\nLargest node is:");

                    obj.findLargestNode(obj.tree);

                    break;

                case 7:

                    System.out.println("Exiting program.");

                    scanner.close();

                    System.exit(0);

                default:

                    System.out.println("Invalid choice, try again.");

            }

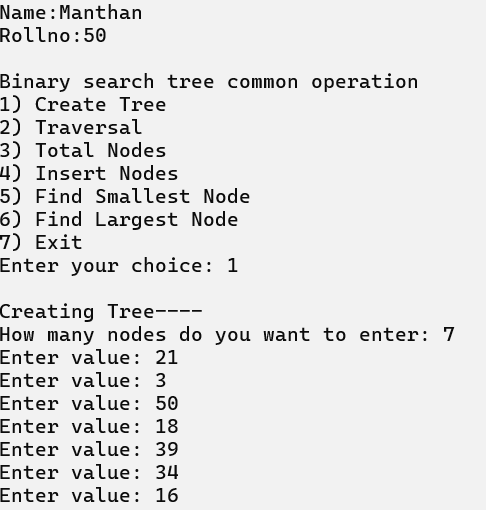
        }

    }

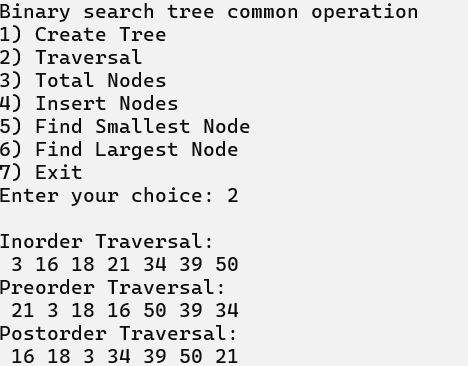
 }

**OUTPUT :**

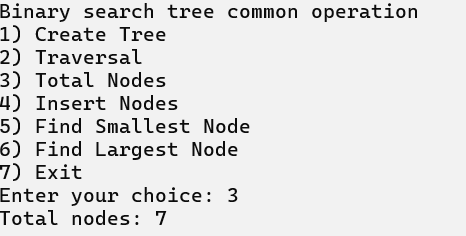
**1. Create Tree :**

****

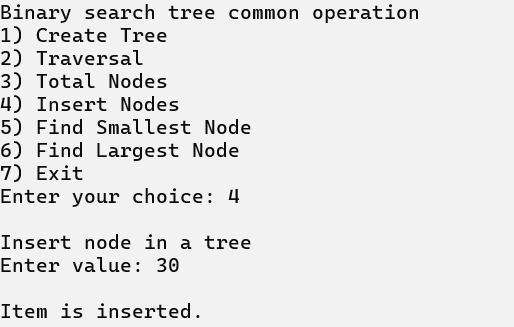
**2. Traversal :**

****

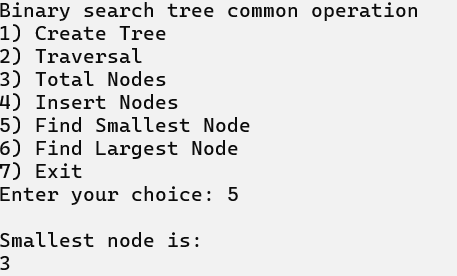
**3. Total Nodes :**

****

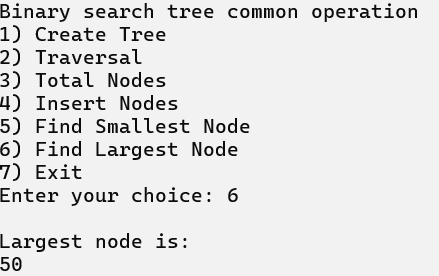
**4. Insert Nodes :**

****

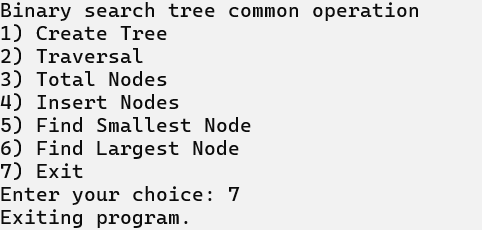
**5. Find Smallest Node :**

****

**6. Find Largest Node :**

****

**7. Exit :**

****

**PRACTICAL NO : 8**

**Aim : Implementing Heap with different operations.**

**1) Write a program to implement Max Heap.**

**Code :**

import java.util.Scanner;

 public class MaxHeap {

    // Method to perform max heapify

    public static void maxHeapify(int[] a, int i, int n) {

        int j, temp;

        temp = a[i];

        j = 2 \* i;

        // Perform max-heapify by comparing the node with its children

        while (j <= n) {

            if (j < n && a[j + 1] > a[j]) {

                j = j + 1; // If right child is larger, select right child

            }

            if (temp > a[j]) {

                break; // If the node is larger than or equal to the largest child, stop

            } else if (temp <= a[j]) {

                a[j / 2] = a[j]; // Swap the node with the largest child

                j = 2 \* j; // Move down the tree

            }

        }

        a[j / 2] = temp; // Place the original value at the correct position

    }

    // Method to build the max heap from an unsorted array

    public static void buildMaxHeap(int[] a, int n) {

        for (int i = n / 2; i >= 1; i--) {

            maxHeapify(a, i, n); // Call maxHeapify on all non-leaf nodes

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        // Input the number of elements

        System.out.println("Enter number of elements in the array:");

        int n = scanner.nextInt();

        // Initialize the array

        int[] a = new int[21]; // Size 21 to handle 1-based indexing

        // Input the elements into the array

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

            System.out.println("Enter element " + i + ":");

            a[i] = scanner.nextInt();

        }

        // Build the max heap

        buildMaxHeap(a, n);

        // Output the max heap

        System.out.println("Max Heap:");

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

            System.out.println(a[i]);

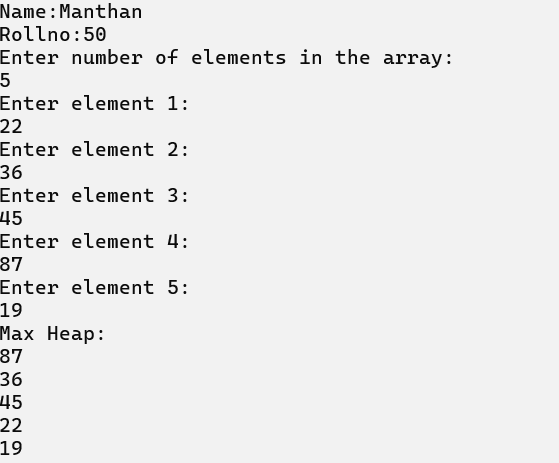
        }

        scanner.close();

    }

 }

**OUTPUT :**



**2) Write a program to implement Min Heap.**

**Code :**

import java.util.Scanner;

 public class MinHeap {

    // Method to perform min heapify

    public static void minHeapify(int[] a, int i, int n) {

        int j, temp;

        temp = a[i];

        j = 2 \* i; // Left child index

        // Perform min-heapify by comparing the node with its children

        while (j <= n) {

            if (j < n && a[j + 1] < a[j]) {

                j = j + 1; // If right child is smaller, select right child

            }

            if (temp < a[j]) {

                break; // If the node is smaller than or equal to the smallest child, stop

            } else if (temp >= a[j]) {

                a[j / 2] = a[j]; // Swap the node with the smallest child

                j = 2 \* j; // Move down the tree

            }

        }

        a[j / 2] = temp; // Place the original value at the correct position

    }

    // Method to build the min heap from an unsorted array

    public static void buildMinHeap(int[] a, int n) {

        for (int i = n / 2; i >= 1; i--) {

            minHeapify(a, i, n); // Call minHeapify on all non-leaf nodes

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        // Input the number of elements

        System.out.println("Enter number of elements in the array:");

        int n = scanner.nextInt();

        // Initialize the array

        int[] a = new int[21]; // Size 21 to handle 1-based indexing

        // Input the elements into the array

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

            System.out.println("Enter element " + i + ":");

            a[i] = scanner.nextInt();

        }

        // Build the min heap

        buildMinHeap(a, n);

        // Output the min heap

        System.out.println("Min Heap:");

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

            System.out.println(a[i]);

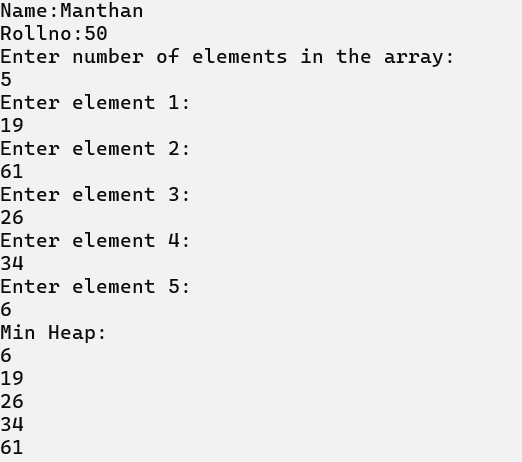
        }

        scanner.close();

    }

 }

**OUTPUT :**

****

**3) Write a program to implement select kth element from heap.**

**Code :**

import java.util.Scanner;

 public class HeapOperations {

    static int[] array1 = new int[100]; // Heap array

    static int n = 0; // Number of nodes in the heap

    // Method to display elements in the heap

    public static void display() {

        if (n == 0) {

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

            return;

        }

        for (int i = 0; i < n; i++) {

            System.out.print(array1[i] + "\t");

        }

        System.out.println();

    }

    // Method to insert an element into the heap

    public static void insert1(int num, int location) {

        int parentnode;

        while (location > 0) {

            parentnode = (location - 1) / 2;

            if (num <= array1[parentnode]) {

                array1[location] = num;

                return;

            }

            array1[location] = array1[parentnode];

            location = parentnode;

        }

        array1[0] = num; // Assign number to the root node

    }

    // Method to delete an element from the heap

    public static void delete1(int num) {

        int left, right, i, temp, parentnode;

        // Find the element to delete

        for (i = 0; i < n; i++) {

            if (num == array1[i]) {

                break;

            }

        }

        if (num != array1[i]) {

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

            return;

        }

        array1[i] = array1[n - 1];

        n = n - 1;

        parentnode = (i - 1) / 2; // Find the parent node of node i

        if (array1[i] > array1[parentnode]) {

            insert1(array1[i], i); // Insert to restore heap property

            return;

        }

        left = 2 \* i + 1; // Left child of i

        right = 2 \* i + 2; // Right child of i

        while (right < n) {

            if (array1[i] >= array1[left] && array1[i] >= array1[right]) {

                return; // If the node is greater than both children, no need to move

            }

            if (array1[right] <= array1[left]) {

                temp = array1[i];

                array1[i] = array1[left];

                array1[left] = temp;

                i = left;

            } else {

                temp = array1[i];

                array1[i] = array1[right];

                array1[right] = temp;

                i = right;

            }

            left = 2 \* i + 1;

            right = 2 \* i + 2;

        }

        if (left == n - 1 && array1[i] < array1[left]) {

            temp = array1[i];

            array1[i] = array1[left];

            array1[left] = temp;

        }

    }

    public static void main(String[] args) {

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

 Scanner scanner = new Scanner(System.in);

        int choice, num;

        // Loop until the user decides to quit

        while (true) {

            System.out.println("1. Insert the element");

            System.out.println("2. Delete the element");

            System.out.println("3. Display all elements");

            System.out.println("4. Quit");

            System.out.print("Enter your choice: ");

            choice = scanner.nextInt();

            switch (choice) {

                case 1:

                    System.out.print("Enter the element to be inserted to the list: ");

                    num = scanner.nextInt();

                    insert1(num, n);

                    n = n + 1;

                    break;

                case 2:

                    System.out.print("Enter the element to be deleted from the list: ");

                    num = scanner.nextInt();

                    delete1(num);

                    break;

                case 3:

                    display();

                    break;

                case 4:

                    System.exit(0);

                    break;

                default:

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

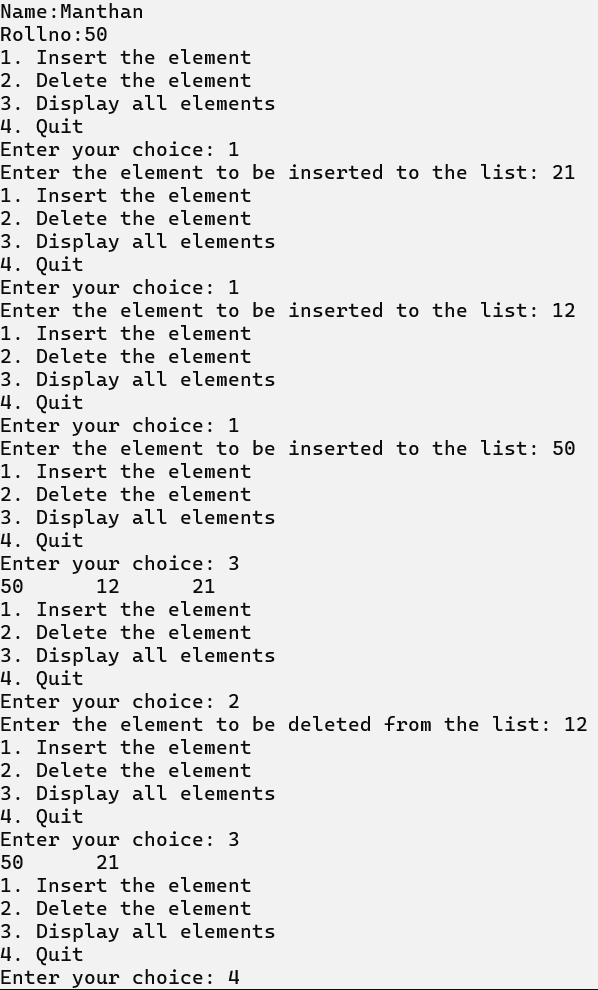
            }

        }

    }

 }

**OUTPUT :**

****

**PRACTICAL NO : 9**

**Aim : Implementation of Adjacency Matrix.**

**1) Write a program to implement Adjacency Matrix.**

**Code :**

import java.util.Scanner;

 class AdjacencyMatrix {

    private int[][] adj;

    private boolean[] visited;

    private int n;

    // Constructor to initialize the adjacency matrix

    public AdjacencyMatrix(int n) {

        this.n = n;

        visited = new boolean[n];

        adj = new int[n][n];

        // Initialize all values to 0

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                adj[i][j] = 0;

            }

        }

    }

    // Method to add an edge

    public void addEdge(int origin, int dest) {

        if (origin >= n || dest >= n || origin < 0 || dest < 0) {

            System.out.println("Wrong nodes");

        } else {

            adj[origin][dest] = 1;

        }

    }

    // Method to display the adjacency matrix

    public void display() {

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                System.out.print(adj[i][j] + "\t");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int nodes, maxEdges, origin, dest;

 System.out.println("Name:Manthan");

 System.out.println("Rollno:50");

        System.out.print("Enter Maximum number of nodes: ");

        nodes = scanner.nextInt();

        AdjacencyMatrix am = new AdjacencyMatrix(nodes);

        maxEdges = nodes \* (nodes - 1);

        System.out.println("Enter -1 -1 to exit");

        for (int i = 0; i < maxEdges; i++) {

            System.out.print("\nEnter edges: ");

            origin = scanner.nextInt();

            dest = scanner.nextInt();

            if (origin == -1 && dest == -1) {

                break;

            } else {

                am.addEdge(origin, dest);

            }

        }

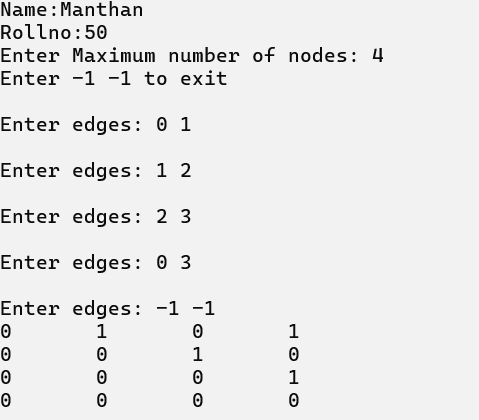
        am.display();

        scanner.close();

    }

 }

**OUTPUT :**

****

**PRACTICAL NO : 10**

**Aim : Implementation of Graph Traversal (DFS and BFS).**

**1) Write a program to implement Breadth First Search.**

**Code :**

import java.util.\*; public class BFS {

static final int MAX = 20;

static int[][] adj = new int[MAX][MAX]; static boolean[] visited = new boolean[MAX]; static int n;

// Create the graph by taking input static void createGraph() {

Scanner sc = new Scanner(System.in); int maxEdges, origin, destin;

System.out.print("Enter number of Nodes: "); n = sc.nextInt();

maxEdges = n \* (n - 1);

for (int i = 1; i <= maxEdges; i++) { System.out.println("Enter edge (0 0 to quit): " + i); origin = sc.nextInt();

destin = sc.nextInt();

if (origin == 0 || destin == 0) break;

if (origin > n || destin > n || origin <= 0 || destin <= 0) { System.out.println("Invalid Edge");

i--;

} else { adj[origin][destin] = 1;

}

}

}

// Display the adjacency matrix static void display() {

for (int i = 1; i <= n; i++) { for (int j = 1; j <= n; j++) {

System.out.print(adj[i][j] + "\t");

}

System.out.println();

}

}

// Breadth First Search (BFS) implementation static void bfs(int v) {

int frnt, rear;

int[] que = new int[MAX]; frnt = rear = -1; System.out.print(v + " "); visited[v] = true;

rear++; frnt++; que[rear] = v

while (frnt <= rear) { v = que[frnt]; frnt++;

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

if (adj[v][i] == 1 && !visited[i]) { System.out.print(i + "\t"); visited[i] = true;

rear++; que[rear] = i;

}

}

}

}

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int v, ch;

System.out.println("Kaustubh Mhatre");

System.out.println("Roll no.39"); createGraph();

while (true) {

System.out.println("\n1. Breadth First Search Using Queue"); System.out.println("2. Exit");

System.out.print("Enter your choice: "); ch = sc.nextInt();

switch (ch) { case 1:

System.out.print("Enter Starting node for Breadth First Search: "); v = sc.nextInt();

Arrays.fill(visited, false); bfs(v);

break; case 2:

System.exit(0); break;

default:

System.out.println("Wrong Choice"); break;

}

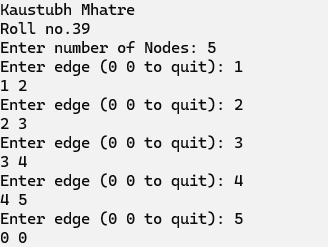
}

}

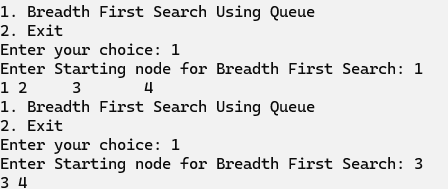
}

# OUTPUT :

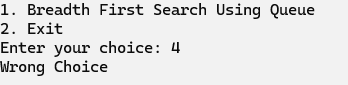
**Enter the No. of nodes and the edges :**



# Breadth First Search using Queue :



**Wrong Choice :**



# Exit Operation :



**2. Write a program to implement Depth First Search.**

**CODE :**

import java.util.\*;

class Graph {

    // Adjacency list representation of the graph

    private Map<Integer, List<Integer>> adj;

    // Constructor to initialize the graph

    public Graph() {

        adj = new HashMap<>();

    }

    // Function to add an edge to the graph

    public void addEdge(int v, int w) {

        adj.putIfAbsent(v, new ArrayList<>());

        adj.get(v).add(w);

    }

    // Recursive DFS function

    public void DFS(int v, Set<Integer> visited) {

        // Mark the current node as visited and print it

        visited.add(v);

        System.out.print(v + " ");

        // Recur for all the vertices adjacent to this vertex

        for (int neighbor : adj.getOrDefault(v, new ArrayList<>())) {

            if (!visited.contains(neighbor)) {

                DFS(neighbor, visited);

            }

        }

    }

    public static void main(String[] args) {

        // Create a new graph

        Graph g = new Graph();

        // Add edges to the graph

        g.addEdge(0, 1);

        g.addEdge(0, 2);

        g.addEdge(1, 2);

        g.addEdge(2, 0);

        g.addEdge(2, 3);

        g.addEdge(3, 3);

// Create a set to track visited nodes

Set<Integer> visited = new HashSet<>();

// Print DFS traversal starting from vertex 2

System.out.println("Name:Manthan");

System.out.println("Rollno:50");

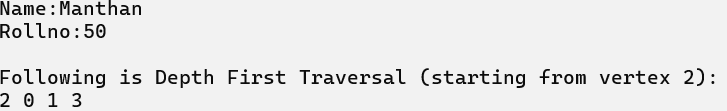
System.out.println("\nFollowing is Depth First Traversal (starting from vertex 2):");

g.DFS(2, visited);

}

}

**OUTPUT :**

****

**PRACTICAL NO : 11**

**Aim : Create a Minimum Spanning Tree using any method Kruskal’s or Prim’s Algorithm.**

**1) Write a program to implement Minimum Spanning Tree using Kruskal’s Algorithm.**

**Code:**

import java.util.\*;

class Graph1 {

    int V, E;

    List<Edge> edges;

    // Constructor to initialize graph with V vertices and E edges

    public Graph1(int V, int E) {

        this.V = V;

        this.E = E;

        edges = new ArrayList<>();

    }

    // Add an edge to the graph

    public void addEdge(int u, int v, int w) {

        edges.add(new Edge(u, v, w));

    }

    // Kruskal's MST algorithm

    public int kruskalMST() {

        int mst\_wt = 0;

        // Sort the edges in ascending order of their weights

        Collections.sort(edges, Comparator.comparingInt(edge -> edge.weight));

        // Create a DisjointSets object for union-find

        DisjointSets ds = new DisjointSets(V);

        // Iterate through sorted edges and include them in MST if no cycle is formed

        for (Edge edge : edges) {

            int u = edge.src;

            int v = edge.dest;

            int set\_u = ds.find(u);

            int set\_v = ds.find(v);

            // If u and v are not in the same set, include the edge

            if (set\_u != set\_v) {

                System.out.println(u + " - " + v);

                mst\_wt += edge.weight;

                ds.merge(set\_u, set\_v);

            }

        }

        return mst\_wt;

    }

    public static void main(String[] args) {

        System.out.println("Name:Manthan");

        System.out.println("Rollno:50");

        System.out.println("\nKruskal's Algorithm\n");

        // Number of vertices and edges

        int V = 9, E = 14;

        // Create the graph and add edges

        Graph1 g = new Graph1(V, E);

        g.addEdge(0, 1, 4);

        g.addEdge(0, 7, 8);

        g.addEdge(1, 2, 8);

        g.addEdge(1, 7, 11);

        g.addEdge(2, 3, 7);

        g.addEdge(2, 8, 2);

        g.addEdge(2, 5, 4);

        g.addEdge(3, 4, 9);

        g.addEdge(3, 5, 14);

        g.addEdge(4, 5, 10);

        g.addEdge(5, 6, 2);

        g.addEdge(6, 7, 1);

        g.addEdge(6, 8, 6);

        g.addEdge(7, 8, 7);

        System.out.println("Edges of MST are:");

        int mst\_wt = g.kruskalMST();

        System.out.println("\nWeight of MST is " + mst\_wt);

    }

}

// Edge class to store information about an edge

class Edge {

    int src, dest, weight;

    // Constructor

    public Edge(int src, int dest, int weight) {

        this.src = src;

        this.dest = dest;

        this.weight = weight;

    }

}

class DisjointSets {

    int[] parent, rnk;

    int n;

    // Constructor to initialize DisjointSets

    public DisjointSets(int n) {

        this.n = n;

        parent = new int[n + 1];

        rnk = new int[n + 1];

        // Initialize each node as its own parent (self-loop)

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

            rnk[i] = 0;

            parent[i] = i;

        }

    }

    // Find the root of the set containing 'u'

    public int find(int u) {

        if (u != parent[u]) {

            parent[u] = find(parent[u]); // Path compression

        }

        return parent[u];

    }

    // Merge two sets

    public void merge(int x, int y) {

        x = find(x);

        y = find(y);

        // Union by rank

        if (rnk[x] > rnk[y]) {

            parent[y] = x;

        } else {

            parent[x] = y;

            if (rnk[x] == rnk[y]) {

                rnk[y]++;

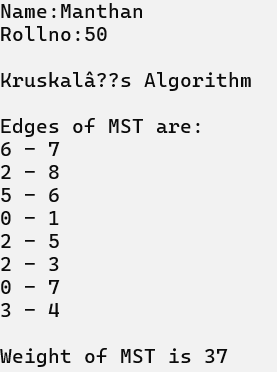
            }

        }

    }

}

**OUTPUT :**

****