K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

LAB FILE

DATA STRUCTURES

(ENCS 253)

SUBMITTED BY:

B. TECH CSE

SUBMITTED TO:

Mrs. SUMAN PUNIA

ASSISTANT PROFESSOR

SOET

School of Engineering & Technology

K. R. MANGALAM UNIVERSITY

Sohna, Haryana 122103, India

INDEX

AIM: Design, Develop and Implement a menu driven Program for the following Array operations

a. Creating an Array of N Integer Elements

b. Display of Array Elements

c. Inserting an Element at a given valid Position (POS)

d. Deleting an Element at a given valid Position(POS)

e. Exit.

CODE:

```
import java.util.Scanner;
public class ArrayOperations {
  public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] array = new int[n];
        int size = 0; // Tracks the actual number of elements

        while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Create Array");
        System.out.println("2. Display Array");
        System.out.println("3. Insert Element");
        System.out.println("4. Delete Element");
        System.out.println("5. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
                case 1: // Create Array
             System.out.println("Enter elements:");
                for (int i = 0; i < n; i++) {
                array[i] = scanner.nextInt();
                }
                size = n;
             System.out.println("Array created.");
                break;

                case 2: // Display Array
             System.out.println("Array elements:");
                for (int i = 0; i < size; i++) {
                System.out.print(array[i] + " ");
                }
                System.out.println();
                break;

                case 3: // Insert Element
                if (size >= n) {
                System.out.println("Array is full. Cannot insert element.");
                } else {
                System.out.print("Enter element to insert: ");
                int element = scanner.nextInt();
                System.out.print("Enter position to insert (1 to " + (size + 1) + "): ");
                int pos = scanner.nextInt() - 1;
                if (pos >= 0 && pos <= size) {
                        for (int i = size; i > pos; i--) {
                        array[i] = array[i - 1];
                        }
                        array[pos] = element;
                        size++;
                   System.out.println("Element inserted.");
                } else {
                   System.out.println("Invalid position.");
                }
                }
                break;

                case 4: // Delete Element
            System.out.print("Enter position to delete (1 to " + size + "): ");
                int delPos = scanner.nextInt() - 1;
                if (delPos >= 0 && delPos < size) {
                for (int i = delPos; i < size - 1; i++) {
                        array[i] = array[i + 1];
                }
                size--;
                System.out.println("Element deleted.");
                } else {
                System.out.println("Invalid position.");
                }
                break;

                case 5: // Exit
            System.out.println("Exiting...");
                scanner.close();
                return;

                default:
                System.out.println("Invalid choice. Please try again.");
        }
        }
    }
}
```
OUTPUT:

```
Enter the number of elements in the array: 4
Menu:
1. Create Array
2. Display Array
3. Insert Element
4. Delete Element
5. Exit
Enter your choice: 1
Enter elements:
3
4
5
4
Array created.
```

AIM: Design, Develop and Implement a menu driven Program for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)
a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate how Stack can be used to check Palindrome
d. Demonstrate Overflow and Underflow situations on Stack
e. Display the status of Stack
f. Exit

CODE:

```
import java.util.Scanner;
public class StackOperations {
   private static final int MAX = 10;
   private int[] stack = new int[MAX];
   private int top = -1;

   public boolean isFull() {
         return top == MAX - 1;
   }

   public boolean isEmpty() {
         return top == -1;
   }

   public void push(int element) {
         if (isFull()) {
         System.out.println("Stack Overflow. Cannot push element.");
         } else {
         stack[++top] = element;
         System.out.println("Pushed " + element + " onto the stack.");
         }
   }

   public void pop() {
         if (isEmpty()) {
         System.out.println("Stack Underflow. Cannot pop element.");
         } else {
         int element = stack[top--];
         System.out.println("Popped " + element + " from the stack.");
         }
   }

   public boolean isPalindrome() {
         int start = 0;
         int end = top;
         while (start < end) {
         if (stack[start] != stack[end]) {
                 return false;
         }
         start++;
         end--;
         }
         return true;
   }

   public void display() {
         if (isEmpty()) {
```

```
        System.out.println("Stack is empty.");
        } else {
        System.out.print("Stack elements: ");
        for (int i = 0; i <= top; i++) {
                System.out.print(stack[i] + " ");
        }
        System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        StackOperations stackOps = new StackOperations();

        while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Push an element");
        System.out.println("2. Pop an element");
        System.out.println("3. Check if stack is a palindrome");
        System.out.println("4. Display stack");
        System.out.println("5. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
                case 1: // Push an Element
            System.out.print("Enter element to push: ");
                int element = scanner.nextInt();
                stackOps.push(element);
                break;

                case 2: // Pop an Element
                stackOps.pop();
                break;

                case 3: // Check if Stack is a Palindrome
                if (stackOps.isPalindrome()) {
                System.out.println("Stack is a palindrome.");
                } else {
                System.out.println("Stack is not a palindrome.");
                }
                break;

                case 4: // Display Stack
                stackOps.display();
                break;

                case 5: // Exit
            System.out.println("Exiting...");
                scanner.close();
                return;

                default:
            System.out.println("Invalid choice. Please try again.");
        }
        }
    }
}
```

OUTPUT:

```
Menu:
1. Push an element
2. Pop an element
3. Check if stack is a palindrome
4. Display stack
5. Exit
Enter your choice: 1
Enter element to push: 3
Pushed 3 onto the stack

+
```

AIM: Design, Develop and Implement a menu driven Program for the following operations on QUEUE of Characters
a. Insert an Element on to Linear QUEUE
b. Delete an Element from Linear QUEUE

CODE:

```
import java.util.Scanner;
public class LinearQueue {
   private static final int MAX = 10;
   private char[] queue = new char[MAX];
   private int front = -1;
   private int rear = -1;

   // Method to check if the queue is full
   public boolean isFull() {
         return rear == MAX - 1;
   }

   // Method to check if the queue is empty
   public boolean isEmpty() {
         return front == -1 || front > rear;
   }

   // Method to insert an element into the queue
   public void enqueue(char element) {
         if (isFull()) {
         System.out.println("Queue Overflow. Cannot insert element.");
         } else {
         if (front == -1) {
                 front = 0;
         }
         queue[++rear] = element;
         System.out.println("Inserted " + element + " into the queue.");
         }
   }

   // Method to delete an element from the queue
   public void dequeue() {
         if (isEmpty()) {
         System.out.println("Queue Underflow. Cannot delete element.");
         } else {
         char element = queue[front++];
         System.out.println("Deleted " + element + " from the queue.");
         if (front > rear) {
                 front = rear = -1; // Reset the queue if it becomes empty
         }
         }
   }

   // Method to display the elements of the queue
   public void display() {
         if (isEmpty()) {
         System.out.println("Queue is empty.");
         } else {
         System.out.print("Queue elements: ");
         for (int i = front; i <= rear; i++) {
                 System.out.print(queue[i] + " ");
```

```
        }
        System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LinearQueue queue = new LinearQueue();

        while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Insert an element into the queue");
        System.out.println("2. Delete an element from the queue");
        System.out.println("3. Display the queue");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
                case 1: // Enqueue operation
              System.out.print("Enter character to insert: ");
                char element = scanner.next().charAt(0);
                queue.enqueue(element);
                break;

                case 2: // Dequeue operation
                queue.dequeue();
                break;

                case 3: // Display the queue
                queue.display();
                break;

                case 4: // Exit
              System.out.println("Exiting...");
                scanner.close();
                return;

                default:
              System.out.println("Invalid choice. Please try again.");
        }
        }
    }
}
```
OUTPUT:

```
Menu:
1. Insert an element into the queue
2. Delete an element from the queue
3. Display the queue
4. Exit
Enter your choice: 1
Enter character to insert: 3
Inserted 3 into the queue.
```

AIM: Design, Develop and Implement a menu driven Program in java for the following operations on Circular QUEUE of Characters
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display

CODE:
```
import java.util.Scanner;
public class CircularQueue {
   private static final int MAX = 5;
   private char[] queue = new char[MAX];
   private int front = -1;
   private int rear = -1;

   // Method to check if the queue is full
   public boolean isFull() {
        return (rear + 1) % MAX == front;
   }

   // Method to check if the queue is empty
   public boolean isEmpty() {
        return front == -1;
   }

   // Method to insert an element into the circular queue
   public void enqueue(char element) {
        if (isFull()) {
        System.out.println("Queue Overflow. Cannot insert element.");
        } else {
        if (isEmpty()) {
                front = 0;
        }
        rear = (rear + 1) % MAX;
        queue[rear] = element;
        System.out.println("Inserted " + element + " into the queue.");
        }
   }

   // Method to delete an element from the circular queue
   public void dequeue() {
        if (isEmpty()) {
        System.out.println("Queue Underflow. Cannot delete element.");
        } else {
        char element = queue[front];
        if (front == rear) {
                front = rear = -1; // Reset queue if it becomes empty
        } else {
                front = (front + 1) % MAX;
        }
        System.out.println("Deleted " + element + " from the queue.");
        }
   }

   // Method to display the elements of the circular queue
   public void display() {
        if (isEmpty()) {
        System.out.println("Queue is empty.");
        } else {
```

```
        System.out.print("Queue elements: ");
        int i = front;
        while (true) {
            System.out.print(queue[i] + " ");
            if (i == rear) break;
            i = (i + 1) % MAX;
        }
        System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        CircularQueue queue = new CircularQueue();

        while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Insert an element into the queue");
        System.out.println("2. Delete an element from the queue");
        System.out.println("3. Display the queue");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
            case 1: // Enqueue operation
          System.out.print("Enter character to insert: ");
            char element = scanner.next().charAt(0);
            queue.enqueue(element);
            break;

            case 2: // Dequeue operation
            queue.dequeue();
            break;

            case 3: // Display the queue
            queue.display();
            break;

            case 4: // Exit
          System.out.println("Exiting...");
            scanner.close();
            return;

            default:
          System.out.println("Invalid choice. Please try again.");
        }
        }
    }
}
```
OUTPUT:

```
Menu:
1. Insert an element into the queue
2. Delete an element from the queue
3. Display the queue
4. Exit
Enter your choice: 1
Enter character to insert: 3
Inserted 3 into the queue.
```

AIM: Design, Develop and Implement a menu driven Program for the following operations onSingly Linked List (SLL)
a. Create a SLL.
b. Insert at Beginning
c. Insert at Last
d. Insert at any random location
e. Delete from Beginning
f. Delete from Last
g. Delete node after specified location
h. Search for an element
i. Show
j. Exit

CODE:
```
 import java.util.Scanner;
class Node {
   int data;
   Node next;

   Node(int data) {
         this.data = data;
         this.next = null;
   }
}

public class SinglyLinkedList {
   private Node head;

   // Method to insert at the beginning of the list
   public void insertAtBeginning(int data) {
         Node newNode = new Node(data);
         newNode.next = head;
         head = newNode;
         System.out.println(data + " inserted at the beginning.");
   }

   // Method to insert at the end of the list
   public void insertAtEnd(int data) {
         Node newNode = new Node(data);
         if (head == null) {
         head = newNode;
         } else {
         Node temp = head;
         while (temp.next != null) {
                 temp = temp.next;
         }
         temp.next = newNode;
         }
         System.out.println(data + " inserted at the end.");
   }

   // Method to insert at a specific position
   public void insertAtPosition(int data, int position) {
         Node newNode = new Node(data);
         if (position == 1) {
         newNode.next = head;
         head = newNode;
         } else {
         Node temp = head;
```

```
        for (int i = 1; i < position - 1 && temp != null; i++) {
                temp = temp.next;
        }
        if (temp != null) {
                newNode.next = temp.next;
                temp.next = newNode;
                System.out.println(data + " inserted at position " + position);
        } else {
           System.out.println("Invalid position.");
        }
        }
}

// Method to delete from the beginning
public void deleteFromBeginning() {
        if (head == null) {
        System.out.println("List is empty.");
        } else {
        System.out.println("Deleted " + head.data + " from the beginning.");
        head = head.next;
        }
}

// Method to delete from the end
public void deleteFromEnd() {
        if (head == null) {
        System.out.println("List is empty.");
        } else if (head.next == null) {
        System.out.println("Deleted " + head.data + " from the end.");
        head = null;
        } else {
        Node temp = head;
        while (temp.next.next != null) {
                temp = temp.next;
        }
        System.out.println("Deleted " + temp.next.data + " from the end.");
        temp.next = null;
        }
}

// Method to delete a node after a specific position
public void deleteAfterPosition(int position) {
        if (head == null) {
        System.out.println("List is empty.");
        } else {
        Node temp = head;
        for (int i = 1; i < position && temp != null; i++) {
                temp = temp.next;
        }
        if (temp != null && temp.next != null) {
           System.out.println("Deleted " + temp.next.data + " after position " + position);
                temp.next = temp.next.next;
        } else {
           System.out.println("Invalid position.");
        }
        }
}

// Method to search for an element
```

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

// Method to display the list
public void display() {
        if (head == null) {
        System.out.println("List is empty.");
        } else {
        Node temp = head;
        System.out.print("Singly Linked List: ");
        while (temp != null) {
                System.out.print(temp.data + " ");
                temp = temp.next;
        }
        System.out.println();
        }
}

public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();
        Scanner scanner = new Scanner(System.in);

        while (true) {
        System.out.println("\nMenu:");
        System.out.println("1. Insert at Beginning");
        System.out.println("2. Insert at End");
        System.out.println("3. Insert at Position");
        System.out.println("4. Delete from Beginning");
        System.out.println("5. Delete from End");
        System.out.println("6. Delete after Position");
        System.out.println("7. Search for Element");
        System.out.println("8. Display List");
        System.out.println("9. Exit");
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();

        switch (choice) {
                case 1:
            System.out.print("Enter data to insert at the beginning: ");
                int dataBegin = scanner.nextInt();
            list.insertAtBeginning(dataBegin);
                break;

                case 2:
            System.out.print("Enter data to insert at the end: ");
                int dataEnd = scanner.nextInt();
                list.insertAtEnd(dataEnd);
                break;

                case 3:
            System.out.print("Enter data to insert: ");
```

```
                        int dataPos = scanner.nextInt();
                    System.out.print("Enter position to insert at: ");
                        int position = scanner.nextInt();
                    list.insertAtPosition(dataPos, position);
                        break;

                        case 4:
                        list.deleteFromBeginning();
                        break;

                        case 5:
                        list.deleteFromEnd();
                        break;

                        case 6:
                    System.out.print("Enter position after which to delete: ");
                        int delPosition = scanner.nextInt();
                    list.deleteAfterPosition(delPosition);
                        break;

                        case 7:
                    System.out.print("Enter element to search for: ");
                        int searchData = scanner.nextInt();
                        if (list.search(searchData)) {
                        System.out.println(searchData + " found in the list.");
                        } else {
                        System.out.println(searchData + " not found in the list.");
                        }
                        break;
                        case 8:
                        list.display();
                        break;

                        case 9:
                    System.out.println("Exiting...");
                        scanner.close();
                        return;
                        default:
                    System.out.println("Invalid choice. Please try again.");
            }
            }
    }
}
```

OUTPUT:

```
Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete after Position
7. Search for Element
8. Display List
9. Exit
Enter your choice: 1
Enter data to insert at the beginning: 3
3 inserted at the beginning.
```

AIM: Design, Develop and Implement the following menu driven Programs in java using Array operations a. Write a program for Bubble Sort algorithm b. Write a program for Merge Sort algorithm c. Write a program for Insertion Sort algorithm

CODE:

```
import java.util.Scanner;
public class SortPrograms {
    // Bubble Sort
    public static void bubbleSort(int[] arr) {
        for (int i = 0; i < arr.length - 1; i++) {
            for (int j = 0; j < arr.length - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    // Merge Sort
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    public static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1, n2 = right - mid;
        int[] L = new int[n1], R = new int[n2];
        System.arraycopy(arr, left, L, 0, n1);
        System.arraycopy(arr, mid + 1, R, 0, n2);

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            arr[k++] = L[i] <= R[j] ? L[i++] : R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    // Insertion Sort
    public static void insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i], j = i - 1;
            while (j >= 0 && arr[j] > key) arr[j + 1] = arr[j--];
            arr[j + 1] = key;
        }
    }
    // Display the array
    public static void display(int[] arr) {
        for (int i : arr) System.out.print(i + " ");
        System.out.println();
    }

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

        // Menu-driven interface
        do {
            System.out.println("Choose Sorting Algorithm:");
            System.out.println("1. Bubble Sort");
            System.out.println("2. Merge Sort");
            System.out.println("3. Insertion Sort");
            System.out.println("4. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();

            if (choice == 4) break;

            System.out.print("Enter number of elements: ");
            int n = sc.nextInt();
            int[] arr = new int[n];
            System.out.print("Enter elements: ");
            for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
            switch (choice) {
                case 1:
                    bubbleSort(arr);
                    System.out.print("Sorted array (Bubble Sort): ");
                    display(arr);
                    break;
                case 2:
                    mergeSort(arr, 0, n - 1);
                    System.out.print("Sorted array (Merge Sort): ");
                    display(arr);
                    break;
                case 3:
                    insertionSort(arr);
                    System.out.print("Sorted array (Insertion Sort): ");
                    display(arr);
                    break;
                default:
                    System.out.println("Invalid choice! Try again.");
            }
        } while (choice != 4);

        sc.close();
    }
}
```

OUTPUT:

```
Choose Sorting Algorithm:
1. Bubble Sort
2. Merge Sort
3. Insertion Sort
4. Exit
Enter choice: 1
Enter number of elements: 5
Enter elements: 64 34 25 12 22
Sorted array (Bubble Sort): 12 22 25 34 64
```

AIM: Design, Develop and Implement a Program in C/C++ for converting an Infix Expression to PostfixExpression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands

CODE:

```
import java.util.Stack;
public class InfixToPostfixConverter {

    // Method to determine operator precedence
    private static int precedence(char op) {
        return switch (op) {
            case '^' -> 3;
            case '*', '/', '%' -> 2;
            case '+', '-' -> 1;
            default -> -1;
        };
    }

    // Method to check if a character is an operator
    private static boolean isOperator(char c) {
        return "+-*/%^".indexOf(c) != -1;
    }

    // Method to convert infix to postfix
    public static String infixToPostfix(String infix) {
        StringBuilder postfix = new StringBuilder();
        Stack<Character> stack = new Stack<>();

        for (char c : infix.toCharArray()) {
            if (Character.isLetterOrDigit(c)) {
                postfix.append(c); // Append operands directly
            } else if (c == '(') {
                stack.push(c); // Push '(' onto stack
            } else if (c == ')') {
                // Pop until '(' is found
                while (!stack.isEmpty() && stack.peek() != '(') {
                    postfix.append(stack.pop());
                }
                stack.pop(); // Remove '('
            } else if (isOperator(c)) {
                // Pop higher precedence operators from stack
                while (!stack.isEmpty() && precedence(stack.peek()) >= precedence(c)) {
                    postfix.append(stack.pop());
                }
                stack.push(c); // Push current operator
            }
        }

        // Pop remaining operators from stack
        while (!stack.isEmpty()) {
            postfix.append(stack.pop());
        }
        return postfix.toString();
    }

    public static void main(String[] args) {
        String infixExpression = "A*(B+C)/D-E^F";
        System.out.println("Infix Expression: " + infixExpression);
        String postfixExpression = infixToPostfix(infixExpression);
        System.out.println("Postfix Expression: " + postfixExpression);
    }
}
```

OUTPUT:

```
Input (Infix): A*(B+C)/D-E^F
Output (Postfix): ABC+*D/E-F^
```

AIM: Design, Develop and Implement the following menu driven Programs in C/C++ for implementing a. Write a program for linear search algorithm b. Write a program for displaying a sparse matrix

CODE:

```
import java.util.Scanner;

public class MenuDrivenProgram {

    // Linear Search method
    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i; // Return the index of the target
            }
        }
        return -1; // Return -1 if the target is not found
    }

    // Method to display sparse matrix in a compact form
    public static void displaySparseMatrix(int[][] matrix, int rows, int cols) {
        System.out.println("Non-zero elements in Sparse Matrix:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (matrix[i][j] != 0) {
                    System.out.println("Row: " + i + " Column: " + j + " Value: " + matrix[i][j]);
                }
            }
        }
    }

    // Main method to drive the menu
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            // Display menu
            System.out.println("\nMenu:");
            System.out.println("1. Linear Search Algorithm");
            System.out.println("2. Display Sparse Matrix");
            System.out.println("3. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1: // Linear Search Algorithm
                    System.out.print("Enter the size of the array: ");
                    int size = scanner.nextInt();
                    int[] arr = new int[size];
                    System.out.println("Enter the elements of the array:");
                    for (int i = 0; i < size; i++) {
                        arr[i] = scanner.nextInt();
                    }
                    System.out.print("Enter the value to search: ");
                    int target = scanner.nextInt();
                    int result = linearSearch(arr, target);
                    if (result != -1) {
                        System.out.println("Element found at index: " + result);
                    } else {
                        System.out.println("Element not found.");
                    }
                    break;

                case 2: // Display Sparse Matrix
                    System.out.print("Enter the number of rows: ");
                    int rows = scanner.nextInt();
                    System.out.print("Enter the number of columns: ");
                    int cols = scanner.nextInt();
                    int[][] matrix = new int[rows][cols];
                    System.out.println("Enter the matrix elements:");
                    for (int i = 0; i < rows; i++) {
                        for (int j = 0; j < cols; j++) {
                            matrix[i][j] = scanner.nextInt();
                        }
                    }
                    displaySparseMatrix(matrix, rows, cols);
                    break;

                case 3: // Exit
                    System.out.println("Exiting program...");
                    break;

                default:
                    System.out.println("Invalid choice! Please enter a valid option.");
            }
        } while (choice != 3); // Repeat until the user selects exit

        scanner.close();
    }
}
```

OUTPUT:

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the matrix elements:
0 0 0
0 5 0
0 0 3
Non-zero elements in Sparse Matrix:
Row: 1 Column: 1 Value: 5
Row: 2 Column: 2 Value: 3
```

AIM: Design, Develop and Implement a menu driven Program in C/C++ for the following operations on Binary Search Tree (BST) of Integers a. Create a BST of N Integers: 8, 10, 3, 1, 6, 14, 7 b. Traverse the BST in Inorder c. Traverse the BST in Preorder d. Traverse the BST in and Postorder

CODE:

```
import java.util.Scanner;

class BinarySearchTree {
    static class Node {
        int value;
        Node left, right;

        public Node(int value) {
            this.value = value;
            left = right = null;
        }
    }

    private Node root;

    // Method to insert a node in BST
    public void insert(int value) {
        root = insertRec(root, value);
    }

    private Node insertRec(Node root, int value) {
        // If the tree is empty, create a new node
        if (root == null) {
            root = new Node(value);
            return root;
        }

        // Otherwise, recur down the tree
        if (value < root.value) {
            root.left = insertRec(root.left, value);
        } else if (value > root.value) {
            root.right = insertRec(root.right, value);
        }

        // return the (unchanged) node pointer
        return root;
    }

    // Inorder Traversal (Left, Root, Right)
    public void inorder() {
        inorderRec(root);
        System.out.println();
    }

    private void inorderRec(Node root) {
        if (root != null) {
            inorderRec(root.left);
            System.out.print(root.value + " ");
            inorderRec(root.right);
        }
    }

    // Preorder Traversal (Root, Left, Right)
    public void preorder() {
        preorderRec(root);
        System.out.println();
    }

    private void preorderRec(Node root) {
        if (root != null) {
            System.out.print(root.value + " ");
            preorderRec(root.left);
            preorderRec(root.right);
        }
    }

    // Postorder Traversal (Left, Right, Root)
    public void postorder() {
        postorderRec(root);
        System.out.println();
    }

    private void postorderRec(Node root) {
        if (root != null) {
            postorderRec(root.left);
            postorderRec(root.right);
            System.out.print(root.value + " ");
        }
    }

    // Main driver program
    public static void main(String[] args) {
        BinarySearchTree bst = new BinarySearchTree();
        Scanner scanner = new Scanner(System.in);
        int choice;

        // Predefined values for the BST
        int[] values = {8, 10, 3, 1, 6, 14, 7};

        // Insert predefined values into the BST
        for (int value : values) {
            bst.insert(value);
        }

        do {
            // Display menu
            System.out.println("\nMenu:");
            System.out.println("1. Create BST with predefined values");
            System.out.println("2. Inorder Traversal");
            System.out.println("3. Preorder Traversal");
            System.out.println("4. Postorder Traversal");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.println("BST created with predefined values: 8, 10, 3, 1, 6, 14, 7");
                    break;
                case 2:
                    System.out.print("Inorder Traversal: ");
                    bst.inorder();
                    break;
                case 3:
                    System.out.print("Preorder Traversal: ");
                    bst.preorder();
                    break;
                case 4:
                    System.out.print("Postorder Traversal: ");
                    bst.postorder();
                    break;
                case 5:
                    System.out.println("Exiting program...");
                    break;
                default:
                    System.out.println("Invalid choice! Please enter a valid option.");
            }
        } while (choice != 5);

        scanner.close();
    }
}
```

OUTPUT:

```
Menu:
1. Create BST with predefined values
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Exit
Enter your choice: 1
BST created with predefined values: 8, 10, 3, 1, 6, 14, 7
```

AIM: Design, Develop and Implement a Program in C/C++ for the following operations on Graph(G) a. Create a Graph N using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a graph using BFS method c. Print all the nodes reachable from a given starting node in a graph using DFS method

CODE:

```
import java.util.*;
class Graph {
  private int[][] adjMatrix;
  private int vertices;

  // Constructor to initialize the graph with N vertices
  public Graph(int vertices) {
    this.vertices = vertices;
    adjMatrix = new int[vertices][vertices];
  }

  // Method to add an edge from u to v
  public void addEdge(int u, int v) {
    adjMatrix[u][v] = 1;
  }

  // BFS Method to print all reachable nodes from the starting node
  public void BFS(int start) {
    boolean[] visited = new boolean[vertices];
    Queue<Integer> queue = new LinkedList<>();

    visited[start] = true;
    queue.add(start);

    System.out.println("BFS starting from node " + start + ":");

    while (!queue.isEmpty()) {
      int currentNode = queue.poll();
      System.out.print(currentNode + " ");

      // Visit all the neighbors of the current node
      for (int i = 0; i < vertices; i++) {
        if (adjMatrix[currentNode][i] == 1 && !visited[i]) {
          visited[i] = true;
          queue.add(i);
        }
      }
    }
    System.out.println();
  }

  // DFS Method to print all reachable nodes from the starting node
  public void DFS(int start) {
    boolean[] visited = new boolean[vertices];
    System.out.println("DFS starting from node " + start + ":");
    DFSUtil(start, visited);
    System.out.println();
    }

    // Helper function for DFS
    private void DFSUtil(int node, boolean[] visited) {
        visited[node] = true;
        System.out.print(node + " ");

        // Visit all the neighbors of the current node
        for (int i = 0; i < vertices; i++) {
            if (adjMatrix[node][i] == 1 && !visited[i]) {
                DFSUtil(i, visited);
            }
        }
    }

    // Method to print the adjacency matrix
    public void printGraph() {
        System.out.println("Adjacency Matrix of the Graph:");
        for (int i = 0; i < vertices; i++) {
            for (int j = 0; j < vertices; j++) {
                System.out.print(adjMatrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}

public class GraphOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Create a graph with 5 vertices
        Graph graph = new Graph(5);

        // Add edges to the graph (directed)
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(2, 4);
        graph.addEdge(3, 2);
        graph.addEdge(4, 0);

        // Print the adjacency matrix representation of the graph
        graph.printGraph();

        // BFS traversal starting from node 0
        graph.BFS(0);

        // DFS traversal starting from node 0
        graph.DFS(0);

        scanner.close();
    }
```