K M M INSTITUTE OF POSTGRADUATE STUDIES

(Affiliated to Sri Venkateswara University, Tirupati)
RAMIREDDIPALLE, TIRUPATI-517 102

DEAPARTMENT OF COMPUTER APPLICATIONS

CERTIFICATE

REGISTERED NO: _____

This is to certify that _____ is a bonafide student of MCA (II Semester) course in our Institution. The student has done the Lab work as per this record for this practical fulfillment of the requirement of the MCA course ***208 P DATA STRUCTURES LAB***

(Subject Code & Subject Title)

during the ***II*** *Semester of the Academic Year 2024-2025.*

Head-of the Department *Lecturer-In-Charge*

Attended and submitted for the University Practical Examination.

Signature of Examiners: *1.*_____
(External Examiner-1)

*2.*_____
(External Examiner-2)

CONTENTS

/* 1. PROGRAM TO ARRAY OPERTIONS */

Aim:-To Implement a Java Program to Perform Different Operations on Arrays.

Description:-

- Arrays are defined as the collection of similar type of data items stored at contiguous memory locations.
- Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.
- Array is the simplest data structure where each data element can be randomly
- accessed by using its index number.
- For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variable for the marks in different subject. Instead of that, we can define an array which can store the marks in each subject at a the contiguous memory locations.

Algorithm to Insert an element in an Array

Insertion is the operation in which a new value is added at a particular place in an array.

Array Reg[N] with last element at M_i^{th} position value X is to be inserted at i^{th} location.

1.if(m<n) then back=m+1 else stop
2. while (back>i) repeat steps 3 to 4
3. reg[back]= reg[back-1]
4. back=back-1;
5. reg [back]='x'
6. m=m+1
7. end.

Algorithm for Deleting an element in an Array

Deletion is a process of deleting a particular element from an array.
If an element to be deleted i^{th} location then all elements from the $(i+1)^{th}$ location we have to be shifted one step towards left.
So $(i+1)^{th}$ element is copied to i^{th} location and $(i+2)^{th}$ to $(i+1)^{th}$ location and so on.

In this algorithm a value is being deleted from i^{th} location of an array Reg [N]. Let us assume that last element in the array is at M^{th} position.

1. Back=1
2. While (Back<M) repeat 3 and 4
3. Reg[Back]= Reg[Back+1]
4. Back= Back+1
5. M=M-1
6. End

Algorithm to Display the elements in an Array

step 1: start.

step 2: initialize arr[] = {1, 2, 3, 4, 5}.

step 3: length= sizeof(arr)/sizeof(arr[0])

step 4: print "elements of given array:"

step 5: i=0 repeat step 6 and step 7 until i<length.

step 6: print arr[i]

step 7: i=i+1.

step 8: return 0

// PROGRAM

```
import java.util.ArrayList;
import java.util.Scanner;
class Array {
   private ArrayList<Integer> a; // Using ArrayList for dynamic size
   private int size;

   public Array() {
      a = new ArrayList<>();
      size = 0;
   }
   public void create() {
      Scanner cin = new Scanner(System.in);
      System.out.println("Enter the size of the array: ");
      size = cin.nextInt();
      System.out.println("Enter " + size + " array elements:");
      for (int i = 0; i < size; i++) {
         a.add(cin.nextInt());
      }
   }
   public void insert(int ele, int pos) {
      if (pos < 1 || pos > size + 1) {
         System.out.println("Invalid position. Insertion failed.");
         return;
      }
      a.add(pos - 1, ele);
      size++;
      System.out.println(ele + " inserted at position " + pos);
   }

   public void delete(int ele) {
      boolean found = a.remove(Integer.valueOf(ele));
      if (found) {
         size--;
         System.out.println(ele + " deleted successfully.");
      } else {
         System.out.println("Element not found in the array.");
      }
   }
   public void display() {
```

```
    System.out.print("Array elements:");
    for (int i = 0; i < size; i++) {
      System.out.print(" " + a.get(i));
    }
    System.out.println();
  }
  public static void main(String[] args) {
    Array obj = new Array();
    Scanner cin = new Scanner(System.in);
    int ch, pos, ele;
    do {
      System.out.println("\n\t\tMENU\n");
      System.out.println("1. Create");
      System.out.println("2. Insert");
      System.out.println("3. Delete");
      System.out.println("4. Display");
      System.out.println("5. Exit");
      System.out.print("Enter your choice: ");
      ch = cin.nextInt();
      switch (ch) {
        case 1:
          obj.create();
          break;
        case 2:
          System.out.print("Enter the position to be inserted: ");
          pos = cin.nextInt();
          System.out.print("Enter the item to be inserted: ");
          ele = cin.nextInt();
          obj.insert(ele, pos);
          break;
        case 3:
          System.out.print("Enter the item to be deleted: ");
          ele = cin.nextInt();
          obj.delete(ele);
          break;
        case 4:
          obj.display();
          break;
        case 5:
          System.out.println("Exiting... Thank you!");
          break;
```

```
                default:
                  System.out.println("Invalid choice! Please enter a valid option.");
              }
            } while (ch != 5);
            cin.close();
        }
    }
```

Input/output

1. Create
2. Insert
3. Delete
4. Display
5. Exit

Enter your choice: 1
Enter the size of the array: 5
Enter 5 array elements:
10 20 30 40 50

Enter your choice: 2
Enter the position to be inserted: 3
Enter the item to be inserted: 25

Enter your choice: 4
Array elements: 10 20 25 30 40 50

Enter your choice: 3
Enter the item to be deleted: 30
30 deleted successfully.

Enter your choice: 4
Array elements: 10 20 25 40 50

Enter your choice: 5
Exiting... Thank you!

Conclusion

Array operations Menu was successfully Verified for given inputs.

/*2. STACK OPERATIONS USING ARRAYS*/

Aim:-To Implement a program for STACK Operations in Java

Description:-

- Stack is a linear data structure in which the insertion and deletion operations are performed at only one end.
- In a stack, adding and removing of elements are performed at a single position which is known as "**top**". That means, a new element is added at top of the stack and an element is removed from the top of the stack.
- In stack, the insertion and deletion operations are performed based on **LIFO** (Last In First Out) principle.

Algorithm to push(value) - Inserting value into the stack

- In a stack, push() is a function used to insert an element into the stack.
- In a stack, the new element is always inserted at **top** position.
- Push function takes one integer value as parameter and inserts that value into the stack.

We can use the following steps to push an element on to the stack...

Step 1 - Check whether **stack** is **FULL**. (**top == SIZE-1**)

Step 2 - If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

Algorithm to pop(value) - Delete a value from the Stack

- In a stack, pop() is a function used to delete an element from the stack.
- In a stack, the element is always deleted from **top** position. Pop function does not take any value as parameter.

We can use the following steps to pop an element from the stack...

Step 1 - Check whether **stack** is **EMPTY**. (**top == -1**)

Step 2 - If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**).

Algorithm to display(values) - Displays the elements of a Stack

We can use the following steps to display the elements of a stack.

Step 1 - Check whether **stack** is **EMPTY**. (**top == -1**)

Step 2 - If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then define a variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).

Step 4 - Repeat above step until **i** value becomes '0'.

// PROGRAM

```
package Datastructures;
import java.util.Scanner;

class Stack {
   private static final int MAX_SIZE = 100;
   private int[] arr;
   private int top;

   public Stack() {
      arr = new int[MAX_SIZE];
      top = -1; // Initialize top of stack
   }

   // Function to push element onto stack
   public void push(int value) {
      if (top >= MAX_SIZE - 1) {
         System.out.println("Stack Overflow! Cannot push element.");
      } else {
         arr[++top] = value;
         System.out.println(value + " pushed to stack.");
      }
   }
   // Function to pop element from stack
   public void pop() {
      if (top < 0) {
         System.out.println("Stack Underflow! Cannot pop from empty stack.");
      } else {
         int popped = arr[top--];
         System.out.println(popped + " popped from stack.");
      }
   }
   // Function to display all elements in the stack
   public void display() {
      if (top < 0) {
         System.out.println("Stack is empty.");
      } else {
         System.out.println("Stack elements are:");
         for (int i = top; i >= 0; i--) {
            System.out.println(arr[i]);
```

```
        }
      }
    }
}
public class StackOperations {
    public static void main(String[] args) {
        Stack stack = new Stack();
        Scanner scanner = new Scanner(System.in);
        int choice, value;
        do {
            System.out.println("\nStack Operations Menu:");
            System.out.println("1. Push");
            System.out.println("2. Pop");
            System.out.println("3. Display stack");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Enter value to push: ");
                    value = scanner.nextInt();
                    stack.push(value);
                    break;
                case 2:
                    stack.pop();
                    break;
                case 3:
                    stack.display();
                    break;
                case 4:
                    System.out.println("Exiting program.");
                    break;
                default:
                    System.out.println("Invalid choice. Please enter a valid choice.");
            }
        } while (choice != 4);

        scanner.close();
    }
}
```

Input/output

```
Stack Operations Menu:
1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 1
Enter value to push: 10
10 pushed to stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 1
Enter value to push: 20
20 pushed to stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 3
Stack elements are:
20
10

Stack Operations Menu:
1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 2
20 popped from stack.

Stack Operations Menu:
1. Push
2. Pop
3. Display stack
4. Exit
```

```
Enter your choice: 3
Stack elements are:
10

Stack Operations Menu:
1. Push
2. Pop
3. Display stack
4. Exit
Enter your choice: 4
Exiting program.
```

Conclusion

Stack menu driven operations was successfully tested in for the given values.

/* 3. QUEUE OPERATIONS USING ARRAYS */

Aim :- To Implement a program for QUEUE Operations in Java

Description:-

- A queue data structure can be implemented using one dimensional array.
- The queue implemented using array stores only fixed number of data values.
- The implementation of queue data structure using array is very simple.
- Just define a one dimensional array of specific size and insert or delete the values into that array by using **FIFO (First In First Out) principle** with the help of variables **'front'** and '**rear**'.
- Initially both '**front**' and '**rear**' are set to -1.
- Whenever, we want to insert a new value into the queue, increment '**REAR**' value by one and then insert at that position.
- Whenever we want to delete a value from the queue, then delete the element which is at 'FRONT' position and increment 'front' value by one.

En-Queue(value) - Inserting value into the queue

- In a queue data structure, en-Queue() is a function used to insert a new element into the queue.
- In a queue, the new element is always inserted at **rear** position.
- The en-Queue() function takes one integer value as a parameter and inserts that value into the queue.

We can use the following steps to insert an element into the queue.

Step1 - Check whether **queue** is **FULL**. (**rear == SIZE-1**)

Step2- If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step3- If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

De-Queue(value) - Deleting a value from the Queue

- In a queue data structure, de-Queue() is a function used to delete an element from the queue.
- In a queue, the element is always deleted from **front** position.
- The de-Queue() function does not take any value as parameter.

We can use the following steps to delete an element from the queue

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element.

Step 4 - Then check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

Display() - Displays the elements of a Queue

We can use the following steps to display the elements of a queue...

Step 1 - Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.

Step 3 - If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front+1**'.

Step 4 - Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value reaches to **rear** (**i** <= **rear**)

// PROGRAM

```
package Datastructures;
import java.util.Scanner;
class Queue {
    private static final int MAX_SIZE = 100;
    private int[] arr;
    private int front;
    private int rear;

    public Queue() {
        arr = new int[MAX_SIZE];
        front = -1; // Initialize front of queue
        rear = -1; // Initialize rear of queue
    }

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

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

    // Function to add element to the queue
    public void enqueue(int value) {
        if (isFull()) {
            System.out.println("Queue Overflow! Cannot enqueue element.");
        } else if (isEmpty()) {
            front = rear = 0;
            arr[rear] = value;
            System.out.println(value + " enqueued to queue.");
        } else {
            arr[++rear] = value;
            System.out.println(value + " enqueued to queue.");
        }
    }
    // Function to remove element from the queue
    public void dequeue()
```

```
{
        if (isEmpty()) {
            System.out.println("Queue Underflow! Cannot dequeue from empty queue.");
        } else if (front == rear) {
            int dequeued = arr[front];
            front = rear = -1;
            System.out.println(dequeued + " dequeued from queue.");
        } else {
            int dequeued = arr[front++];
            System.out.println(dequeued + " dequeued from queue.");
        }
    }

    // Function to display all elements in the queue
    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty.");
        } else {
            System.out.println("Queue elements are:");
            for (int i = front; i <= rear; i++) {
                System.out.println(arr[i]);
            }
        }
    }
}

public class QueueOperations {
    public static void main(String[] args) {
        Queue queue = new Queue();
        Scanner scanner = new Scanner(System.in);
        int choice, value;
        do {
            System.out.println("\nQueue Operations Menu:");
            System.out.println("1. Enqueue");
            System.out.println("2. Dequeue");
            System.out.println("3. Display queue");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("Enter value to enqueue: ");
```

```
                value = scanner.nextInt();
                queue.enqueue(value);
                break;
            case 2:
                queue.dequeue();
                break;
            case 3:
                queue.display();
                break;
            case 4:
                System.out.println("Exiting program.");
                break;
            default:
                System.out.println("Invalid choice. Please enter a valid choice.");
        }
    } while (choice != 4);

    scanner.close();
  }
}
```

Input/output

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
10 enqueued to queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
20 enqueued to queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display queue
4. Exit
Enter your choice: 3
Queue elements are:
10
20

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display queue
4. Exit
Enter your choice: 2
10 dequeued from queue.

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display queue
4. Exit

```
Enter your choice: 3
Queue elements are:
20

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display queue
4. Exit
Enter your choice: 4
Exiting program.
```

Conclusion

Queue operations are successfully in this program by given input.

/* 4. Circular Queue Using Linked List */

Aim:- To implement a java program using Linked List for Circular Queue Operations

Description:- A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a *Ring Buffer*.

Algorithms Steps:-
Start

1. Initialize Circular Queue:
   - Initialize front and rear pointers as null (empty queue).

2. Check if Queue is Empty:
   - Check if both front and rear pointers are null.
   - If true, display "Queue is empty".
   - If false, continue.

3. Check if Queue is Full (Optional):
   - Determine the condition for queue full (e.g., comparing size with maximum capacity).
   - If true, display "Queue is full".
   - If false or not applicable, continue.

4. Enqueue (Insert Operation):
   - Input data to be enqueued.
   - Create a new node with the data.
   - If queue is empty (front and rear are null):
     a. Point front and rear to the new node.
     b. Link the last node's next pointer to the front (for circular nature).
   - If queue is not empty:
     a. Link the new node's next pointer to front.
     b. Link the rear's next pointer to the new node.
     c. Move rear pointer to the new node.

5. Dequeue (Remove Operation):
   - Check if queue is empty (front and rear are null):
     a. Display underflow message and exit.
   - If queue has only one element (front == rear):
     a. Store data from front node.
     b. Set front and rear to null (empty queue).
   - If queue has more than one element:

a. Store data from front node.
b. Move front to the next node.
c. Link rear's next pointer to the new front (for circular nature).

6. Display Queue:
- Initialize a pointer (current) to front of the queue.
- Loop until current reaches rear:
a. Print data of current node.
b. Move current to the next node.

7. End

// PROGRAM

```
import java.util.Scanner;
class Node {
   int data;
   Node next;
   public Node(int data) {
      this.data = data;
   }
}
class CircularQueue {
   private Node front;
   private Node rear;

   public CircularQueue() {
      front = null;
      rear = null;
   }
   // Check if the queue is empty
   public boolean isEmpty() {
      return front == null;
   }
   // Enqueue an element
   public void enqueue(int data) {
      Node newNode = new Node(data);
      if (isEmpty()) {
         front = newNode;
      } else {
         rear.next = newNode;
      }
      rear = newNode;
      rear.next = front; // Make it circular
      System.out.println(data + " enqueued to queue");
   }
   // Dequeue an element
   public int dequeue() {
      if (isEmpty()) {
         System.out.println("Queue is empty");
         return -1;
      }
      int value;
      if (front == rear) { // Only one element
```

```
            value = front.data;
            front = null;
            rear = null;
        } else {
            Node temp = front;
            value = temp.data;
            front = front.next;
            rear.next = front;
        }
        System.out.println(value + " dequeued from queue");
        return value;
    }

    // Display the queue
    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is empty");
            return;
        }
        Node temp = front;
        System.out.println("Elements in the circular queue are: ");
        do {
            System.out.print(temp.data + " ");
            temp = temp.next;
        } while (temp != front);
        System.out.println();
    }
}
public class CircularQueueUsingLinkedList {
    public static void main(String[] args) {
        CircularQueue queue = new CircularQueue();
        Scanner scanner = new Scanner(System.in);
        int choice, value;
        while (true) {
            System.out.println("\n--- Circular Queue Menu ---");
            System.out.println("1. Enqueue");
            System.out.println("2. Dequeue");
            System.out.println("3. Display");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            switch (choice) {
```

```
case 1:
                System.out.print("Enter the value to enqueue: ");
                value = scanner.nextInt();
                queue.enqueue(value);
                break;
            case 2:
                queue.dequeue();
                break;
            case 3:
                queue.display();
                break;
            case 4:
                System.out.println("Exiting...");
                scanner.close();
                System.exit(0);
            default:
                System.out.println("Invalid choice. Please enter a valid choice.");
        }
    }
  }
}
```

Input/output

```
D:\DS>java CircularQueueUsingLinkedList

--- Circular Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 10
10 enqueued to queue

--- Circular Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter the value to enqueue: 25
25 enqueued to queue

--- Circular Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Elements in the circular queue are:
10 25

--- Circular Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
10 dequeued from queue

--- Circular Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
```

4. Exit
Enter your choice: 3
Elements in the circular queue are:
25

--- Circular Queue Menu ---
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting...

Conclusion

Circular Queue Using Linked List operations was successfully implemented for given input Values.

/*5 . ADDITION OF TWO POLYNOMALS */

Aim: To Implement a Java Code for Additions of Two Polynomials using linked List.

Description: Given two polynomials represented by two arrays, write a function that adds given two polynomials.

Input: **A[ ] = {5, 0, 10, 6}**
B[ ] = {1, 2, 4}
Output: **Sum[ ] = {6, 2, 14, 6}**

The first input array represents "5 + 0x^1 + 10x^2 + 6x^3"
The second array represents "1 + 2x^1 + 4x^2"
And Output is "6 + 2x^1 + 14x^2 + 6x^3"

Algorithm:

Start

1. Initialize two empty polynomial linked lists: poly1 and poly2

2. Input Polynomial 1:
   - Prompt user to enter number of terms
   - For each term:
     a. Input coefficient
     b. Input exponent
     c. Insert term into poly1 linked list

3. Input Polynomial 2:
   - Prompt user to enter number of terms
   - For each term:
     a. Input coefficient
     b. Input exponent
     c. Insert term into poly2 linked list

4. Initialize an empty linked list for the result: result_Poly

5. Initialize pointers (current1, current2) to the heads of poly1 and poly2 respectively

6. Loop while current1 or current2 is not null:
   a. If current1 is not null and current2 is null:
      - Insert current1 term into result_Poly
      - Move current1 to the next term in poly1
   b. If current2 is not null and current1 is null:

- Insert current2 term into result_Poly
- Move current2 to the next term in poly2

c. If current1 and current2 have the same exponent:
- Calculate sum of coefficients
- If sum is not zero, insert term (sum, exponent) into result_Poly
- Move both current1 and current2 to their next terms

d. If current1 exponent is greater than current2 exponent:
- Insert current1 term into result_Poly
- Move current1 to the next term in poly1

e. If current2 exponent is greater than current1 exponent:
- Insert current2 term into result_Poly
- Move current2 to the next term in poly2

7. Display Result:
- Display polynomial representation of result_Poly

End

// PROGRAM

```
package Datastructures;
import java.io.*;
class Node {
   public int exp, coeff;
   public Node next;
   public Node(int x, int y) {
      coeff = x;
      exp = y;
      next = null;
   }
}
class LinkList {
   public Node first;
   public LinkList() {
      first = null;
   }
   public void insertFirst(int x, int y) {
      Node newNode = new Node(x, y);
      newNode.next = first;
      first = newNode;
   }
   public void insertPos(int x, int y, int p) {
      Node current = first;
      Node newNode = new Node(x, y);
      for (int i = p; i > 1 && current != null; i--)
         current = current.next;
      if (current != null) {
         newNode.next = current.next;
         current.next = newNode;
      }
   }
   public void insertLast(int x, int y) {
      Node newNode = new Node(x, y);
      if (isEmpty()) {
         first = newNode;
      } else {
         Node current = first;
         while (current.next != null) {
            current = current.next;
         }
         current.next = newNode;
```

```
        }
    }
    public boolean find(int key) {
        Node current = first;
        while (current != null) {
            if (current.exp == key)
                return true;
            current = current.next;
        }
        return false;
    }
    public boolean isEmpty() {
        return (first == null);
    }
}
class Polynomial {
    private LinkList l1;
    public Polynomial() {
        l1 = new LinkList();
    }
    public boolean insert(int x, int y) {
        Node current = l1.first;
        int pos = 0;
        while (current != null) {
            if (current.exp == y) {
                System.out.println("Not a valid term; Insert again");
                return false;
            } else if (current.exp < y)
                break;
            pos++;
            current = current.next;
        }
        if (pos == 0)
            l1.insertFirst(x, y);
        else
            l1.insertPos(x, y, pos);
        return true;
    }

    public void displayPoly() {
        int f = 0;
        Node current = l1.first;
        while (current != null) {
```

```
        if (f != 0 && current.coeff > 0)
            System.out.print("+");
        if (current.coeff != 0) {
            if (current.coeff > 1 || current.coeff < -1 || current.exp == 0)
                System.out.print(current.coeff);
            else if (current.coeff == -1)
                System.out.print("-");
            if (current.exp == 1)
                System.out.print("X");
            else if (current.exp > 1 || current.exp < 0)
                System.out.print("X^" + current.exp);
            f = 1;
        }
        current = current.next;
    }
    System.out.println(" ");
}
public void add(Polynomial poly1, Polynomial poly2) {
    int x, y;
    Node current1 = poly1.l1.first;
    Node current2 = poly2.l1.first;
    while (current1 != null && current2 != null) {
        if (current1.exp == current2.exp) {
            x = current1.coeff + current2.coeff;
            y = current1.exp;
            current1 = current1.next;
            current2 = current2.next;
        } else if (current1.exp > current2.exp) {
            x = current1.coeff;
            y = current1.exp;
            current1 = current1.next;
        } else {
            x = current2.coeff;
            y = current2.exp;
            current2 = current2.next;
        }
        l1.insertLast(x, y);
    }
    while (current1 != null) {
        x = current1.coeff;
        y = current1.exp;
        current1 = current1.next;
        l1.insertLast(x, y);
```

```
        }
        while (current2 != null) {
            x = current2.coeff;
            y = current2.exp;
            current2 = current2.next;
            l1.insertLast(x, y);
        }
    }
}
public class PolynomialAdd {
    public static void main(String args[]) throws IOException {
        String ch = "y";
        BufferedReader inp = new BufferedReader(new InputStreamReader(System.in));
        int n, co, ex;
        while (ch.equals("y")) {
            Polynomial p1 = new Polynomial();
            Polynomial p2 = new Polynomial();
            Polynomial p3 = new Polynomial();
            System.out.println("Enter the no: of terms of 1st polynomial");
            n = Integer.parseInt(inp.readLine());
            while (n != 0) {
                System.out.println("Enter the coefficent ");
                co = Integer.parseInt(inp.readLine());
                System.out.println("Enter the exponent");
                ex = Integer.parseInt(inp.readLine());
                if (p1.insert(co, ex))
                    n--;
            }
            System.out.println("Enter the no: of terms of 2nd polynomial");
            n = Integer.parseInt(inp.readLine());
            while (n != 0) {
                System.out.println("Enter the coefficent ");
                co = Integer.parseInt(inp.readLine());
                System.out.println("Enter the exponent");
                ex = Integer.parseInt(inp.readLine());
                if (p2.insert(co, ex))
                    n--;
            }
            System.out.print("1st Polynomial::  ");
            p1.displayPoly();
            System.out.print("2nd Polynomial::  ");
            p2.displayPoly();
            p3.add(p1, p2);
```

```
            System.out.print("Added Polynomial::  ");
            p3.displayPoly();
            System.out.print("Enter y to continue… ");
            ch = inp.readLine();
        }
    }
}
```

Input/output

```
Enter the no: of terms of 1st polynomial
3
Enter the coefficent
3
Enter the exponent
2
Enter the coefficent
1
Enter the exponent
4
Enter the coefficent
2
Enter the exponent
5
Enter the no: of terms of 2nd polynomial
3
Enter the coefficent
1
Enter the exponent
2
Enter the coefficent
3
Enter the exponent
4
Enter the coefficent
5
Enter the exponent
6
1st Polynomial  ::  2X^5+X^4+3X^2
2nd Polynomial ::  5X^6+3X^4+X^2
Added Polynomial ::  5X^6+2X^5+4X^4+4X^2
Enter y to continue…
```

Conclusion

Addition of two polynomials was successfully implemented for given input values.

/* 6. Single linked list Implementation */

Aim:- To Implement a menu driven Program for Single Linked List operations using JAVA.

Description:-

Start
Display Menu Options:
- Insert at the beginning
- Insert at the end
- Delete from the beginning
- Delete from the end
- Display the list
- Exit

Prompt user for choice (1-6)
Switch based on user choice:
1. Insert at the beginning:
Prompt for data to insert
Insert data at the beginning of the list
Display success message
2. Insert at the end:
Prompt for data to insert
Insert data at the end of the list
Display success message
3. Delete from the beginning:
Delete node from the beginning of the list
Display success message or error if list is empty
4. Delete from the end:
Delete node from the end of the list
Display success message or error if list is empty
5. Display the list:
Display all elements in the linked list
6. Exit:
Display exit message
End

// PROGRAM

```
package Datastructures;
//Java Program to implement Single Linked List
import java.io.*;
import java.util.Scanner;
class linkedlist {
	int data;
	linkedlist next;
	linkedlist(int value) {
		this.data = value;
	}
	void display() {
		System.out.println(data);
	}
}
class linked {
	public linkedlist fstnode, lastnode;
	linked() {
		fstnode = null;
		lastnode = null;
	}
	/* Insert node or create linked list */
	void insertnode(int value) {
		linkedlist node = new linkedlist(value);
		node.next = null;
		if(fstnode == null) {
			fstnode = lastnode = node;
			System.out.println("Linked list created successfully!");
		}
		else {
			lastnode.next = node;
			lastnode = node;
			System.out.println("Node inserted successfully!");
		}
	}
	/* Delete node from linked list */
	void delete() {
		int count = 0, number, i;
		linkedlist node, node1;
		Scanner input = new Scanner(System.in);

		for(node = fstnode; node != null; node = node.next)
```

```
                count++;
            display();
            node = node1 = fstnode;
            System.out.println(count+" nodes available here!");
            System.out.println("Enter the node number which you want to delete:");
            number = Integer.parseInt(input.nextLine());
            if(number != 1) {
                if(number <= count) {
                    for(i = 2; i <= number; i++)
                        node = node.next;
                    for(i = 2; i <= number-1; i++)
                        node1 = node1.next;
                    node1.next = node.next;
                    node.next = null;
                    node = null;
                }
                else
                    System.out.println("Invalid node number!\n");
            }
            else {
                fstnode = node.next;
                node = null;
            }

            System.out.println("Node has been deleted successfully!\n");
        }
        /* Display linked list */
        void display() {
            linkedlist node = fstnode;
            System.out.println("List of node:");
            while(node != null) {
                node.display();
                node = node.next;
            }
        }
}
class singlylinkedlist {
        public static void main(String args[ ])
{
            linked list = new linked();
            Scanner input = new Scanner(System.in);
            int op = 0;
            while(op != 4)
```

```
{
                        System.out.println("1. Insert 2. Delete 3. Display 4. Exit");
                        System.out.println("Enter your choice:");
                        op = Integer.parseInt(input.nextLine());
                        switch(op)
{
                                case 1:
                        System.out.println("Enter the position value for Linked list:");
                        list.insertnode(Integer.parseInt(input.nextLine()));
                                        break;
                                case 2:
                                        list.delete();
                                        break;
                                case 3:
                                        list.display();
                                        break;
                                case 4:
                                        System.out.println("Bye Bye!");
                                        System.exit(0);
                                        break;
                                default:
                                        System.out.println("Invalid choice!");

                        }
                }
        }
}
```

Input/output

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

1

Enter the position value for Linked list:

0

Linked list created successfully!

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

1

Enter the position value for Linked list:

1

Node inserted successfully!

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

1

Enter the position value for Linked list:

2

Node inserted successfully!

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

1

Enter the position value for Linked list:

3

Node inserted successfully!

1. Insert
2. Delete
3. Display
4. Exit

Enter your choice:

1
Enter the position value for Linked list:
5
Node inserted successfully!
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the position value for Linked list:
8
Node inserted successfully!
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the position value for Linked list:
45
Node inserted successfully!
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
2
List of node:
0
1
2
3
5
8
45
8 nodes available here!
Enter the node number which you want to delete:
1
Node has been deleted successfully!

1. Insert
2. Delete

3. Display
4. Exit
Enter your choice:
3
List of node:
1
2
3
5
8
45
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
4

Conclusion

Single linked list operations was successfully implemented for given input values.

/* 7. PROGRAM FOR TO DISPLAY ADJACENCY MATRIX */

Aim: Algorithm to implement Adjacency Matrix representations of a Graph in Java using runtime acceptance of inputs

Description: The Adjacency matrix is the way to represent the graphs using the 2D array. It is the fundamental data structure in the graph theory. It can provide efficient ways to analyze connectivity and the relationship within the graph.

Algorithm:

1. User Input:

- The program starts by prompting the user to enter the number of vertices (num Vertices) and the number of edges (num Edges)

2. Adjacency Matrix Initialization:

- An adjacency Matrix of size num Vertices x num Vertices is initialized with all elements set to 0.

3.Edge Input:

- The program then accepts input for each edge in the format u v, where u and v are vertices connected by an edge. For each edge (u, v), it sets adjacency Matrix[u][v] and adjacency Matrix[v][u] to 1 to represent an undirected edge.
-

4.Printing the Adjacency Matrix:

- Finally, it prints out the adjacency matrix to display the graph structure.

// PROGRAM

```
package Datastructures;
import java.util.Scanner;
public class GraphAdjacency
{
 private final int vertices;
 private int[][] adjacency_matrix;
 public GraphAdjacency(int v)
 {
    vertices = v;
    adjacency_matrix = new int[vertices + 1][vertices + 1];
 }
 public void makeEdge(int to, int from, int edge)
 {
    try
    {
       adjacency_matrix[to][from] = edge;
    }
    catch (ArrayIndexOutOfBoundsException index)
    {
       System.out.println("The vertices does not exists");
    }
 }
 public int getEdge(int to, int from)
 {
    try
    {
       return adjacency_matrix[to][from];
    }
    catch (ArrayIndexOutOfBoundsException index)
    {
       System.out.println("The vertices does not exists");
    }
    return -1;
 }
 public static void main(String args[])
 {
    int v, e, count = 1, to = 0, from = 0;
    Scanner sc = new Scanner(System.in);
    GraphAdjacency graph;
    try
    {
       System.out.println("Enter the number of vertices: ");
```

```
        v = sc.nextInt();
        System.out.println("Enter the number of edges: ");
        e = sc.nextInt();
        graph = new GraphAdjacency(v);
        System.out.println("Enter the edges: <to> <from>");
        while (count <= e)
        {
            to = sc.nextInt();
            from = sc.nextInt();
            graph.makeEdge(to, from, 1);
            count++;
        }
        System.out.println("The adjacency matrix for the given graph is: ");
        System.out.print("  ");
        for (int i = 1; i <= v; i++)
            System.out.print(i + " ");
        System.out.println();
        for (int i = 1; i <= v; i++)
        {
            System.out.print(i + " ");
            for (int j = 1; j <= v; j++)
                System.out.print(graph.getEdge(i, j) + " ");
            System.out.println();
        }
    }
    catch (Exception E)
    {
        System.out.println("Somthing went wrong");
    }

    sc.close();
  }
}
```

Input/output

Enter the number of vertices:
4
Enter the number of edges:
4
Enter the edges: <to> <from>
4
3
2
1
2
3
4
1
The adjacency matrix for the given graph is:

```
    1 2 3 4
1   0 0 0 0
2   1 0 1 0
3   0 0 0 0
4   1 0 1 0
```

Conclusion

Adjacency matrix was successfully was successfully Constructed for the given input values.

/* 8. PROGRAM TO CONVERT INFIX TO PREFIX & POSTFIX FORM */

Aim:- To Implement a Menu driven program in Java to convert infix expression to prefix & postfix form.

Description:- An infix and postfix are the expressions. An expression consists of constants, variables, and symbols. Symbols can be operators or parenthesis.
All these components must be arranged according to a set of rules so that all these expressions can be evaluated using the set of rules.

What is infix notation?
When the operator is written in between the operands, then it is known as **infix notation**.
Operand does not have to be always a constant or a variable; it can also be an expression itself.
For example,
(p + q) * (r + s)

Syntax of infix notation is given below:
<operand> <operator> <operand>

What is Postfix Notation?
The postfix expression is an expression in which the operator is written after the operands. **For example,**
The postfix expression of infix notation (2+3) can be written as 23+.

Syntax of Postfix notation is given below:
<operand> <operand><operator>

What is Prefix Notation?
A prefix notation is another form of expression but it does not require other information such as precedence and associativity, whereas an infix notation requires information of precedence and associativity.
It is also known as **polish notation**.
In prefix notation, an operator comes before the operands.
Syntax of Postfix notation is given below:
<operator> <operand> <operand>

Algorithm:-

1.Define Operator Precedence and Associativity:

- Define the precedence and associativity of operators (+, -, *, /, (,)).
- For example, * and / have higher precedence than + and -, and operators are left-associative except for ^ which is right-associative.

2. Define Utility Methods:

- Is Operator(char c): Check if a character is an operator.
- precedence(char op): Return the precedence of an operator.
- Is Higher Precedence(char op1, char op2): Check if op1 has higher precedence than op2.

3. Conversion Methods:

- infix To Prefix(expression):
    - Use a stack to convert the infix expression to prefix form.
- Infix To Postfix(expression):
    - Use a stack to convert the infix expression to postfix form.

4. Menu-Driven Program:

- Display a menu with options:
    - Convert infix to prefix
    - Convert infix to postfix
    - Exit

//PROGRAM

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
class Stack {
   private char[] a;
   private int top, m;
   public Stack(int max) {
      m = max;
      a = new char[m];
      top = -1;
   }
   public void push(char key) {
      a[++top] = key;
   }
   public char pop() {
      return a[top--];
   }
   public char peek() {
      return a[top];
   }
   public boolean isEmpty() {
      return (top == -1);
   }
}
class Evaluation {
   private Stack s;
   private String input;
   private String Input/output = "";
   public Evaluation(String str) {
      input = str;
      s = new Stack(str.length());
   }
   public String inToPre() {
      for (int i = input.length() - 1; i >= 0; i--) {
         char ch = input.charAt(i);
         switch (ch) {
            case '+':
            case '-':
               gotOperator(ch, 1, ')');
               break;
```

```
            case '*':
            case '/':
               gotOperator(ch, 2, ')');
               break;
            case ')':
               s.push(ch);
               break;
            case '(':
               gotParenthesis(')');
               break;
            default:
               Input/output = ch + Input/output;
         }
      }
      while (!s.isEmpty())
         Input/output = s.pop() + Input/output;
      return Input/output;
   }
   public String inToPost() {
      for (int i = 0; i < input.length(); i++) {
         char ch = input.charAt(i);
         switch (ch) {
            case '+':
            case '-':
               gotOperator(ch, 1, '(');
               break;
            case '*':
            case '/':
               gotOperator(ch, 2, '(');
               break;
            case '(':
               s.push(ch);
               break;
            case ')':
               gotParenthesis('(');
               break;
            default:
               Input/output = Input/output + ch;
         }
      }
      while (!s.isEmpty())
         Input/output = Input/output + s.pop();
      return Input/output;
```

```
}
public String preToPost() {
   Stack s = new Stack(input.length());
   StringBuilder Input/output = new StringBuilder();
   for (int i = input.length() - 1; i >= 0; i--) {
      char ch = input.charAt(i);
      if (Character.isLetterOrDigit(ch)) {
         s.push(ch);
      } else {
         char op1 = s.pop();
         char op2 = s.pop();
         Input/output.insert(0, op1);
         Input/output.insert(0, op2);
         Input/output.insert(0, ch);
         for (int j = 0; j < Input/output.length(); j++) {
            s.push(Input/output.charAt(j));
         }
         Input/output.setLength(0);
      }
   }
   while (!s.isEmpty()) {
      Input/output.insert(0, s.pop());
   }
   return Input/output.toString();
}
private void gotOperator(char opThis, int prec1, char x) {
   while (!s.isEmpty()) {
      char opTop = s.pop();
      if (opTop == x) {
         s.push(opTop);
         break;
      } else {
         int prec2;
         if (opTop == '+' || opTop == '-')
            prec2 = 1;
         else
            prec2 = 2;
         if (prec2 < prec1 && x == '(') {
            s.push(opTop);
            break;
         } else if (prec2 <= prec1 && x == ')') {
            s.push(opTop);
            break;
```

```
            } else {
               if (x == ')')
                  Input/output = opTop + Input/output;
               else
                  Input/output = Input/output + opTop;
            }
         }
      }
      s.push(opThis);
   }
   private void gotParenthesis(char x) {
      while (!s.isEmpty()) {
         char ch = s.pop();
         if (ch == x)
            break;
         else {
            if (x == ')')
               Input/output = ch + Input/output;
            else
               Input/output = Input/output + ch;
         }
      }
   }
}
public class ExpressionApp {
   public static void main(String args[]) throws IOException {
      BufferedReader inp = new BufferedReader(new InputStreamReader(System.in));
      String check = "y";
      while (check.equals("y")) {
         System.out.println("MENU");
         System.out.println("---------");
         System.out.println("1. Infix to Prefix");
         System.out.println("2. Infix to Postfix");
         System.out.println("Enter your choice:");
         int n = Integer.parseInt(inp.readLine());
         Evaluation inf;
         switch (n) {
            case 1:
               System.out.println("Enter the infix expression:");
               String s1 = inp.readLine();
               inf = new Evaluation(s1);
               System.out.println("Prefix expression: " + inf.inToPre());
               break;
```

```
            case 2:
               System.out.println("Enter the infix expression:");
               String s2 = inp.readLine();
               inf = new Evaluation(s2);
               System.out.println("Postfix expression: " + inf.inToPost());
               break;
            default:
               System.out.println("Invalid input");
         }
         System.out.println("Press y to continue:");
         check = inp.readLine();
      }
   }
}
```

Input/output

```
MENU
---------
1.Infix to Prefix
2.Infix to Postfix
Enter your choice
1
Enter the infix expression
a+b+c
Prefix expression:- ++abc
Press y to continue
y
MENU
---------
1.Infix to Prefix
2.Infix to Postfix
Enter your choice
2
Enter the infix expression
a+b+c
Postfix expression:- ab+c+
Press y to continue
```

Conclusion

Java program for converting infix to postfix, infix to prefix Notation are successfully Implemented for given input.

/* 9. Operations on Binary Search Tree */

Aim:- To Implement a Java Program for performing different operations on Binary Search Tree.

Description:- A binary search tree follows some order to arrange the elements. In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node.
This rule is applied recursively to the left and right sub-trees of the root.

Algorithm:-

1.Define Tree-Node Class:

- Define a class Tree-Node to represent each node in the BST.
- Each node should have data (data), left child (left), and right child (right).

2. Define Binary-Search-Tree Class:

- Define a class Binary-Search-Tree to manage operations on the BST:
  - Attributes:
    - root: Points to the root node of the BST.
  - Methods:
    - insert(data): Insert a node with data into the BST.
    - delete(data): Delete a node with data from the BST.
    - search(data): Search for a node with data in the BST.
    - inorderTraversal(): Perform in order traversal of the BST.
    - preorderTraversal(): Perform preorder traversal of the BST.
    - postorderTraversal(): Perform post order traversal of the BST.
    - displayMenu(): Display a menu of operations.
    - executeOperation(choice): Execute an operation based on user choice.

3.Menu-Driven Program:

- Display a menu with options:
  - Insert a node
  - Delete a node
  - Search for a node
  - In-order traversal
  - Preorder traversal
  - Post-order traversal
  - Exit
- Prompt the user for choice and input data (where necessary).
- Perform the corresponding operation based on user choice using the methods defined in Binary-Search-Tree.

//PROGRAM

```
import java.util.Scanner;
 /* Class BSTNode */
class BSTNode
{
   BSTNode left, right;
   int data;
   /* Constructor */
   public BSTNode()
   {
      left = null;
      right = null;
      data = 0;
   }
   /* Constructor */
   public BSTNode(int n)
   {
      left = null;
      right = null;
      data = n;
   }
   /* Function to set left node */
   public void setLeft(BSTNode n)
   {
      left = n;
   }
   /* Function to set right node */
   public void setRight(BSTNode n)
   {
      right = n;
   }
   /* Function to get left node */
   public BSTNode getLeft()
   {
      return left;
   }
   /* Function to get right node */
   public BSTNode getRight()
   {
      return right;
   }
```

```
/* Function to set data to node */
   public void setData(int d)
   {
      data = d;
   }
   /* Function to get data from node */
   public int getData()
   {
      return data;
   }
}
/* Class BST */
class BST
{
   private BSTNode root;
   /* Constructor */
   public BST()
   {
      root = null;
   }
   /* Function to check if tree is empty */
   public boolean isEmpty()
   {
      return root == null;
   }
   /* Functions to insert data */
   public void insert(int data)
   {
      root = insert(root, data);
   }
   /* Function to insert data recursively */
   private BSTNode insert(BSTNode node, int data)
   {
      if (node == null)
         node = new BSTNode(data);
      else
      {
         if (data <= node.getData())
            node.left = insert(node.left, data);
         else
            node.right = insert(node.right, data);
      }
```

```
    return node;
}
/* Functions to delete data */
public void delete(int k)
{
    if (isEmpty())
        System.out.println("Tree Empty");
    else if (search(k) == false)
        System.out.println("Sorry "+ k +" is not present");
    else
    {
        root = delete(root, k);
        System.out.println(k+ " deleted from the tree");
    }
}
private BSTNode delete(BSTNode root, int k)
{
    BSTNode p, p2, n;
    if (root.getData() == k)
    {
        BSTNode lt, rt;
        lt = root.getLeft();
        rt = root.getRight();
        if (lt == null && rt == null)
            return null;
        else if (lt == null)
        {
            p = rt;
            return p;
        }
        else if (rt == null)
        {
            p = lt;
            return p;
        }
        else
        {
            p2 = rt;
            p = rt;
            while (p.getLeft() != null)
                p = p.getLeft();
            p.setLeft(lt);
            return p2;
```

```
        }
    }
    if (k < root.getData())
    {
        n = delete(root.getLeft(), k);
        root.setLeft(n);
    }
    else
    {
        n = delete(root.getRight(), k);
        root.setRight(n);
    }
    return root;
}
/* Functions to count number of nodes */
public int countNodes()
{
    return countNodes(root);
}
/* Function to count number of nodes recursively */
private int countNodes(BSTNode r)
{
    if (r == null)
        return 0;
    else
    {
        int l = 1;
        l += countNodes(r.getLeft());
        l += countNodes(r.getRight());
        return l;
    }
}
/* Functions to search for an element */
public boolean search(int val)
{
    return search(root, val);
}
/* Function to search for an element recursively */
private boolean search(BSTNode r, int val)
{
    boolean found = false;
    while ((r != null) && !found)
    {
```

```
            int rval = r.getData();
            if (val < rval)
                r = r.getLeft();
            else if (val > rval)
                r = r.getRight();
            else
            {
                found = true;
                break;
            }
            found = search(r, val);
        }
        return found;
    }
    /* Function for inorder traversal */
    public void inorder()
    {
        inorder(root);
    }
    private void inorder(BSTNode r)
    {
        if (r != null)
        {
            inorder(r.getLeft());
            System.out.print(r.getData() +" ");
            inorder(r.getRight());
        }
    }
    /* Function for preorder traversal */
    public void preorder()
    {
        preorder(root);
    }
    private void preorder(BSTNode r)
    {
        if (r != null)
        {
            System.out.print(r.getData() +" ");
            preorder(r.getLeft());
            preorder(r.getRight());
        }
    }
```

```
    /* Function for postorder traversal */
    public void postorder()
    {
        postorder(root);
    }
    private void postorder(BSTNode r)
    {
        if (r != null)
        {
            postorder(r.getLeft());
            postorder(r.getRight());
            System.out.print(r.getData() +" ");
        }
    }
}

/* Class BinarySearchTree */
public class BinarySearchTree
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        /* Creating object of BST */
        BST bst = new BST();
        System.out.println("Binary Search Tree Test\n");
        char ch;
        /*  Perform tree operations  */
        do
        {
            System.out.println("\nBinary Search Tree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. delete");
            System.out.println("3. search");
            System.out.println("4. count nodes");
            System.out.println("5. check empty");
             int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
                System.out.println("Enter integer element to insert");
                bst.insert( scan.nextInt() );
                break;
```

```
            case 2 :
              System.out.println("Enter integer element to delete");
              bst.delete( scan.nextInt() );
              break;
            case 3 :
              System.out.println("Enter integer element to search");
              System.out.println("Search result : "+ bst.search( scan.nextInt() ));
              break;
            case 4 :
              System.out.println("Nodes = "+ bst.countNodes());
              break;
            case 5 :
              System.out.println("Empty status = "+ bst.isEmpty());
              break;
            default :
              System.out.println("Wrong Entry \n ");
              break;
            }
            /*  Display tree  */
            System.out.print("\nPost order : ");
            bst.postorder();
            System.out.print("\nPre order : ");
            bst.preorder();
            System.out.print("\nIn order : ");
            bst.inorder();

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

Input/output

```
Binary Search Tree Test
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
5
Empty status = true
Post order :
Pre order :
In order :
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
8
Post order : 8
Pre order : 8
In order : 8
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
5
Post order : 5 8
Pre order : 8 5
In order : 5 8
Do you want to continue (Type y or n)
```

y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
3
Post order : 3 5 8
Pre order : 8 5 3
In order : 3 5 8
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
7
Post order : 3 7 5 8
Pre order : 8 5 3 7
In order : 3 5 7 8
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
10
Post order : 3 7 5 10 8
Pre order : 8 5 3 7 10
In order : 3 5 7 8 10
Do you want to continue (Type y or n)
y

```
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
15
Post order : 3 7 5 15 10 8
Pre order : 8 5 3 7 10 15
In order : 3 5 7 8 10 15
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
2
Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
4
Nodes = 7
Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
```

2. delete
3. search
4. count nodes
5. check empty
3
Enter integer element to search
24
Search result : false
Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
3
Enter integer element to search
7
Search result : true
Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
2
2 deleted from the tree
Post order : 3 7 5 15 10 8
Pre order : 8 5 3 7 10 15
In order : 3 5 7 8 10 15
Do you want to continue (Type y or n)
y

Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
8
8 deleted from the tree
Post order : 3 7 5 15 10
Pre order : 10 5 3 7 15
In order : 3 5 7 10 15
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
10
10 deleted from the tree
Post order : 3 7 5 15
Pre order : 15 5 3 7
In order : 3 5 7 15
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
5
5 deleted from the tree
Post order : 3 7 15
Pre order : 15 7 3
In order : 3 7 15

Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
15
15 deleted from the tree
Post order : 3 7
Pre order : 7 3
In order : 3 7
Do you want to continue (Type y or)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
3
3 deleted from the tree
Post order : 7
Pre order : 7
In order : 7
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
77
Sorry 77 is not present
Post order : 7

Pre order : 7
In order : 7
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
7
7 deleted from the tree
Post order :
Pre order :
In order :
Do you want to continue (Type y or n)
y
Binary Search Tree Operations
1. insert
2. delete
3. search
4. count nodes
5. check empty
5
Empty status = true
Post order :
Pre order :
In order :

Do you want to continue (Type y or n)
N

Conclusion:-Program for Binary search tree operations was successfully implemented for Given input values.

/*10 A. Java Program to Implement Breadth First Search*/

Aim:- Java Program to Implement Graph Traversal Technique using Breadth First Search(BFS).

Description:-

- Graph traversal is a technique used for searching a vertex in a graph.
- The graph traversal is also used to decide the order of vertices is visited in the search process.
- A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

  There are two graph traversal techniques and they are as follows:

1. **DFS (Depth First Search)**
2. **BFS (Breadth First Search)**

BFS (Breadth First Search)

- BFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops.
- We use **Queue data structure** with maximum size of total number of vertices in the graph to implement BFS traversal.

Algorithm:- We use the following steps to implement BFS traversal.

Step 1 - Define a Queue of size total number of vertices in the graph.

Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.
Step 3 - Visit all the non-visited **adjacent** vertices of the vertex which is at front of the Queue and insert them into the Queue.
Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
Step 5 - Repeat steps 3 and 4 until queue becomes empty.

Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

// PROGRAM

```
import java.util.InputMismatchException;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
public class BFS
{
   private Queue<Integer> queue;
   public BFS()
   {
      queue = new LinkedList<Integer>();
   }
   public void bfs(int adjacency_matrix[][], int source)
   {
      int number_of_nodes = adjacency_matrix[source].length - 1;
      int[] visited = new int[number_of_nodes + 1];
      int i, element;
      visited[source] = 1;
      queue.add(source);
      while (!queue.isEmpty())
      {
         element = queue.remove();
         i = element;
         System.out.print(i + "\t");
         while (i <= number_of_nodes)
         {
            if (adjacency_matrix[element][i] == 1 && visited[i] == 0)
            {
               queue.add(i);
               visited[i] = 1;
            }
            i++;
         }
      }
   }

   public static void main(String... arg)
   {
      int number_no_nodes, source;
      Scanner scanner = null;
      try
```

```
        {
            System.out.println("Enter the number of nodes in the graph");
            scanner = new Scanner(System.in);
            number_no_nodes = scanner.nextInt();
            int adjacency_matrix[][] = new int[number_no_nodes + 1][number_no_nodes + 1];
            System.out.println("Enter the adjacency matrix");
            for (int i = 1; i <= number_no_nodes; i++)
                for (int j = 1; j <= number_no_nodes; j++)
                    adjacency_matrix[i][j] = scanner.nextInt();
            System.out.println("Enter the source for the graph");
            source = scanner.nextInt();
            System.out.println("The BFS traversal of the graph is ");
            BFS bfs = new BFS();
            bfs.bfs(adjacency_matrix, source);
        } catch (InputMismatchException inputMismatch)
        {
            System.out.println("Wrong Input Format");
        }
        scanner.close();
    }
}
```

Input/output

```
$javac BFS.java
$java BFS
Enter the number of nodes in the graph
4
Enter the adjacency matrix
0 1 0 1
0 0 1 0
0 1 0 1
0 0 0 1
Enter the source for the graph
1
The BFS traversal of the graph is
1       2       4       3
```

Conclusion

Breadth First Search Graph Traversal technique was successfully implemented for Given input values.

/*10 B. Program for Depth First Search*/

Aim:- Java Program to Implement Graph Traversal Technique using Depth First Search(DFS).

Description:-

- Graph traversal is a technique used for searching a vertex in a graph.
- The graph traversal is also used to decide the order of vertices is visited in the search process.
- A graph traversal finds the edges to be used in the search process without creating loops. That means using graph traversal we visit all the vertices of the graph without getting into looping path.

There are two graph traversal techniques and they are as follows:

3. **DFS (Depth First Search)**
4. **BFS (Breadth First Search)**

DFS (Depth First Search)

- DFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops.
- We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal

Algorithm:-

Step 1 - Define a Stack of size total number of vertices in the graph.

Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.

Step 3 - Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to the stack.

Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.

Step 5 - When there is no new vertex to visit then use **back tracking** and pop one vertex from the stack.

Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.

Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph.

// PROGRAM

```
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.Stack;
public class DFS
{
   private Stack<Integer> stack;
   public DFS()
   {
      stack = new Stack<Integer>();
   }
   public void dfs(int adjacency_matrix[][], int source)
   {
      int number_of_nodes = adjacency_matrix[source].length - 1;
      int visited[] = new int[number_of_nodes + 1];
      int element = source;
      int i = source;
      System.out.print(element + "\t");
      visited[source] = 1;
      stack.push(source);
      while (!stack.isEmpty())
      {
         element = stack.peek();
         i = element;
             while (i <= number_of_nodes)
             {
                if (adjacency_matrix[element][i] == 1 && visited[i] == 0)
                {
               stack.push(i);
               visited[i] = 1;
               element = i;
               i = 1;
               System.out.print(element + "\t");
                   continue;
            }
            i++;
             }
         stack.pop();
      }
   }
   public static void main(String...arg)
   {
      int number_of_nodes, source;
```

```
        Scanner scanner = null;
            try
        {
            System.out.println("Enter the number of nodes in the graph");
          scanner = new Scanner(System.in);
          number_of_nodes = scanner.nextInt();
            int adjacency_matrix[][] = new int[number_of_nodes + 1][number_of_nodes + 1];
            System.out.println("Enter the adjacency matrix");
            for (int i = 1; i <= number_of_nodes; i++)
               for (int j = 1; j <= number_of_nodes; j++)
              adjacency_matrix[i][j] = scanner.nextInt();
            System.out.println("Enter the source for the graph");
          source = scanner.nextInt();
          System.out.println("The DFS Traversal for the graph is given by ");
          DFS dfs = new DFS();
          dfs.dfs(adjacency_matrix, source);
        }catch(InputMismatchException inputMismatch)
        {
          System.out.println("Wrong Input format");
        }
        scanner.close();
      }
    }
```

Input/output

```
$javac DFS.java
$java DFS
Enter the number of nodes in the graph
4
Enter the adjacency matrix
0 1 0 1
0 0 1 0
0 1 0 1
0 0 0 1
Enter the source for the graph
1
The DFS Traversal for the graph is given by
1       2       3       4
```

Conclusion

Depth First Search Graph Traversal technique was successfully implemented for Given input values.

/* 11. Program For Linear Search (Strings)*/

Aim:-Java Program Implementation for Searching a string from given Input Strings.

Description:-

Linear search is also called as sequential search algorithm.

It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found.

If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

It is widely used to search an element from the unordered list, i.e., the list in which items are not sorted.

The worst-case time complexity of linear search is O(n).

Algorithm:-

1.Input:

- An array or collection of input strings.
- The string to search for (search String).

2. Initialization:

- Initialize a Boolean variable found to false to track if the string is found.
- Initialize an integer variable index to store the index of the found string.

3.Iterate Through Input Strings:

- Loop through each string in the input strings collection.

4.Search Operation:

- Compare each string with search String to check for a match.
- Depending on the type of search (exact match, substring match, pattern match), use appropriate comparison methods (equals() for exact match, contains() for substring match, regular expressions for pattern match).

5.Update Found Status:

- If a match is found:
    - Set found to true.
    - Store the index of the found string in index.

6.Output:

- If found is true, output the index where the string was found.
- If found is false, indicate that the string was not found in the input strings.

//PROGRAM

```
import java.util.Scanner;
class LinearSearch {
   public static void main(String args[]) {
      int n;
      String search;
      String array[];
      Scanner in = new Scanner(System.in);
      System.out.println("Enter number of elements");
      n = in.nextInt();
      in.nextLine(); // Consume newline left-over
      array = new String[n];
      System.out.println("Enter " + n + " strings");
      for (int c = 0; c < n; c++) {
         array[c] = in.nextLine();
      }
      System.out.println("Enter value to find");
      search = in.nextLine();
      int c;
      for (c = 0; c < n; c++) {
         if (array[c].equals(search)) { // Searching element is present
        System.out.println(search + " is present at location " + (c + 1) +".");
            break;
         }
      }
      if (c == n) { // Element to search isn't present
         System.out.println(search + " isn't present in array.");
      }
   }
}
```

Input/output

```
D:\DS>javac LinearSearch.java
D:\DS>java LinearSearch
Enter number of elements
4
Enter 4 strings
siva
mani
anatha
baskar
Enter value to find
baskar
baskar is present at location 4.
```

Conclusion

Linear Search technique was successfully Implemented for Given input Strings.

/* 12. Program for Binary Search(Integers)*/

Aim:- Java Program Implementation for Searching an integer from given Input Values.

Description:-

Binary search is the search technique that works efficiently on sorted lists.
Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.
Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list.
If the match is found then, the location of the middle element is returned.
Otherwise, we search into either of the halves depending upon the result produced through the match.

Algorithm:-
Binary_Search(a, lower_bound, upper_bound, val) // 'a' is the given array, 'lower_bound' is the index of the first array element, 'upper_bound' is the index of the last array element, 'val' is the value to search .

Step 1: set beg = lower_bound, end = upper_bound, pos = - 1
Step 2: repeat steps 3 and 4 while beg <=end
Step 3: set mid = (beg + end)/2
Step 4: if a[mid] = val
set pos = mid
print pos
go to step 6
else if a[mid] > val
set end = mid - 1
else
set beg = mid + 1
[end of if]
[end of loop]
Step 5: if pos = -1
print "value is not present in the array" [end of if]
Step 6: exit

//PROGRAM

```
package Datastructures;
import java.util.Scanner;
class BinarySearch
{
  public static void main(String args[])
  {
    int c, first, last, middle, n, search, array[];
     Scanner in = new Scanner(System.in);
    System.out.println("Enter number of elements");
    n = in.nextInt();
    array = new int[n];
     System.out.println("Enter " + n + " integers");
     for (c = 0; c < n; c++)
      array[c] = in.nextInt();
     System.out.println("Enter value to find");
    search = in.nextInt();
     first  = 0;
    last   = n - 1;
    middle = (first + last)/2;
     while( first <= last )
    {
      if ( array[middle] < search )
        first = middle + 1;
      else if ( array[middle] == search )
      {
        System.out.println(search + " found at location " + (middle + 1) + ".");
        break;
      }
      else
         last = middle - 1;

      middle = (first + last)/2;
   }
   if ( first > last )
      System.out.println(search + " isn't present in the list.\n");
  }
}
```

Input/output

Enter number of elements
5
Enter 5 integers
2
3
4
5
6
Enter value to find
3
3 found at location 2.

Conclusion

Binary Search technique was successfully implemented for given input Integer Values.

/* 13. Program to Implement Bubble Sort on Given Character*/

Aim:- To Implement a Java Code for Sorting of given characters using Bubble Sort Technique.

Description:- Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order.

- It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water.
- Bubbles in water rise up to the surface; similarly, the array elements in bubble sort move to the end in each iteration.
- Although it is simple to use, it is primarily used as an educational tool because the performance of bubble sort is poor in the real world.
- It is not suitable for large data sets. The average and worst-case complexity of Bubble sort is **$O(n^2)$,** where **n** is a number of items.

Bubble sort is majorly used where –

- complexity does not matter
- simple and short code is preferred

Algorithm:-

1. Start
2. Initialize array of characters
3. Display original array
4. Set n = length of array
5. Outer Loop (i = 0 to n-2)
6. Inner Loop (j = 0 to n-i-2)
7. Compare arr[j] and arr[j+1]
8. If arr[j] > arr[j+1], then
9. Swap arr[j] and arr[j+1]
10. End of Inner Loop
11. End of Outer Loop
12. Display sorted array
13. End

// PROGRAM

```
package Datastructures;
import java.util.Scanner;
class BubbleSort {
  public static void main(String []args) {
    int num, i, j;
    char temp;
    Scanner input = new Scanner(System.in);
    System.out.println("Enter the number of characters to sort:");
    num = input.nextInt();
    char array[] = new char[num];
    System.out.println("Enter " + num + " characters: ");
    for (i = 0; i < num; i++)
      array[i] = input.next().charAt(0);
    for (i = 0; i < ( num - 1 ); i++) {
      for (j = 0; j < num - i - 1; j++) {
        if (array[j] > array[j+1])
        {
          temp = array[j];
          array[j] = array[j+1];
          array[j+1] = temp;
        }
      }
    }
    System.out.println("Sorted list of characters:");
    for (i = 0; i < num; i++)
      System.out.println(array[i]);
  }
}
```

Input/output

```
D:\DS>javac BubbleSort.java
D:\DS>java BubbleSort
Enter the number of characters to sort:
5
Enter 5 characters:
g
a
r
t
c
Sorted list of characters:
a
c
g
r
t
```

Conclusion

The program bubble sort was implemented and verified successfully for given inputs characters.

/* 14. Program for Sorting of Strings using Quick Sort Technique*/

Aim:- To Implement a Java Code for Sorting of given characters using Quick Sort Technique.

Description:-

Quick sort is the widely used sorting algorithm that makes n log n comparisons in average case for sorting an array of n elements. It is a faster and highly efficient sorting algorithm.

This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into sub problems, then solving the sub problems, and combining the results back together to solve the original problem.

Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

Conquer: Recursively, sort two sub arrays with Quick sort.

Combine: Combine the already sorted array.

Quick sort picks an element as pivot, and then it partitions the given array around the picked pivot element.

In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.

After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.

Algorithm:-

1. Start
2. Initialize array of strings
3. Display original array
4. Call quickSort(arr, 0, n-1)
5. quickSort(arr, low, high)
6. if (low < high)
7. partition Index = partition(arr, low, high)
8. quick Sort(arr, low, partition Index - 1)
9. quick Sort(arr, partition Index + 1, high)
10. End if
11. Display sorted array
12. End

//PROGRAM

```
package Datastructures;
import java.util.Scanner;
/** Class QuickSort **/
public class QuickSort
{
   /** Quick Sort function **/
   public static void sort(String[] arr)
   {
      quickSort(arr, 0, arr.length - 1);
   }
   /** Quick sort function **/
   public static void quickSort(String arr[], int low, int high)
   {
      int i = low, j = high;
      String temp;
      String pivot = arr[(low + high) / 2];
       /** partition **/
      while (i <= j)
      {
         while (arr[i].compareTo(pivot) < 0)
            i++;
         while (arr[j].compareTo(pivot) > 0)
            j--;
         if (i <= j)
         {
            /** swap **/
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
             i++;
            j--;
         }
      }
      /** recursively sort lower half **/
      if (low < j)
         quickSort(arr, low, j);
      /** recursively sort upper half **/
      if (i < high)
         quickSort(arr, i, high);
   }
   /** Main method **/
   public static void main(String[] args)
```

```
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Quick Sort Test\n");
        int n, i;
        /** Accept number of elements **/
        System.out.println("Enter number of string elements");
        n = scan.nextInt();
        scan.nextLine(); // Consume newline
        /** Create array of n elements **/
        String arr[] = new String[n];
        /** Accept elements **/
        System.out.println("\nEnter " + n + " string elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextLine();
        /** Call method sort **/
        sort(arr);
        /** Print sorted Array **/
        System.out.println("\nStrings after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
}
```

Input/output

```
D:\DS>javac QuickSort.java
D:\DS>java QuickSort

Quick Sort Test
Enter number of string elements
4
Enter 4 string elements
ravi
bharath
anil
jagan
Strings after sorting
anil bharath jagan ravi
```

Conclusion

The program Quick sort is implemented and verified successfully on given input strings.

/* 15. Program for Sorting the Elements Using Merge Sort*/

Aim:- To implement a Java Code for Sorting the given elements in an Array using Merge Sort Technique.

Description:-

- Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithm.

- It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the **merge()** function to perform the merging.

- The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process.

- The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

Algorithm:-

1. Start
2. Initialize array of integers
3. Display original array
4. Call mergeSort(arr, 0, n-1)
5. mergeSort(arr, low, high)
6. if (low < high)
7. mid = (low + high) / 2
8. mergeSort(arr, low, mid)
9. mergeSort(arr, mid + 1, high)
10. merge(arr, low, mid, high)
11. End if
12. Display sorted array
13. End

//PROGRAM

```
package Datastructures;
import java.util.Scanner;

/* Class MergeSort */
public class MergeSort
{
   /* Merge Sort function */
   public static void sort(int[] a, int low, int high)
   {
      int N = high - low;
      if (N <= 1)
         return;
      int mid = low + N/2;
      // recursively sort
      sort(a, low, mid);
      sort(a, mid, high);
      // merge two sorted subarrays
      int[] temp = new int[N];
      int i = low, j = mid;
      for (int k = 0; k < N; k++)
      {
         if (i == mid)
            temp[k] = a[j++];
         else if (j == high)
            temp[k] = a[i++];
         else if (a[j]<a[i])
            temp[k] = a[j++];
         else
            temp[k] = a[i++];
      }
      for (int k = 0; k < N; k++)
         a[low + k] = temp[k];
   }
   /* Main method */
   public static void main(String[] args)
   {
      Scanner scan = new Scanner( System.in );
      System.out.println("Merge Sort Test\n");
      int n, i;
      /* Accept number of elements */
      System.out.println("Enter number of integer elements");
```

```
        n = scan.nextInt();
        /* Create array of n elements */
        int arr[] = new int[ n ];
        /* Accept elements */
        System.out.println("\nEnter "+ n +" integer elements");
        for (i = 0; i < n; i++)
           arr[i] = scan.nextInt();
        /* Call method sort */
        sort(arr, 0, n);
        /* Print sorted Array */
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
           System.out.print(arr[i]+" ");
        System.out.println();
    }
}
```

Input/output

Merge Sort Test
Enter number of integer elements
7
Enter 7 integer elements
34
23
12
87
45
26
89
Elements after sorting
12 23 26 34 45 87 89

Conclusion

The program merge sort is implemented and verified successfully for given input values.