Data Structure Using JAVA Laboratory Manual

Course Code: B20CS0302

2nd YEAR (SEMESTER – III)

SCHOOL OF COMPUTER SCIENCE

AND

ENGINEERING

2021-22

CONTENTS

I. COURSE DESCRIPTION

Object-oriented programming (OOP) is a computer science term used to characterize a programming language that began development in the 1960's. The term 'object-oriented programming' was originally coined by Xerox PARC to designate a computer application that describes the methodology of using objects as the foundation for computation. By the 1980's, OOP rose to prominence as the programming language of choice, exemplified by the success of C++. Currently, OOPs such as Java, J2EE, C++,C#, Visual Basic.NET, Python and java Script are popular OOP programming languages that any career-oriented Software Engineer or developer should be familiar with.

OOP is widely accepted as being far more flexible than other computer programming languages. OOPs use three basic concepts as the fundamentals for the Abstraction, Polymorphism, Event Handling and Encapsulation are also significant concepts within object oriented programming languages that are explained in online tutorial describing the functionality of each concept in detail.

The java platform is undoubtedly fast moving and comprehensive. Its many application programming interfaces (APIs) provide a wealth of functionality for all aspects of application and system-level programming. Real-world developers never use one or two APIs to solve a problem, but bring together key functionality spanning a number of APIs, Knowing which APIs you need, which parts of which APIs you need, and how the APIs work together to create the best solution can be a daunting task.

II. LAB OBJECTIVES

The objectives of this lab course are to make students to:

1. Implement The Concept of matrix and palindrome.
2. Demonstrate the use of stacks, queues and lists in java.
3. Discuss and Implement the concept of Trees.

III. LAB OUTCOMES

After thecompletion of the course, the student will be able to:

CO#	Course Outcomes	POs	PSOs
CO1	Make use of Java Arrays to solve real world problems.	1 to 5,9,10,12	1
CO2	Develop a java program for implementing the linked list.	1 to 5,9,10,12	1,2
CO3	Build a real world application in Java using stacks and queues.	1 to 5,9,10,12	1,2
CO4	Apply the concepts of trees for solving real world problems.	1 to 5,9,10,12	1,2
CO5	Identify the most suitable data structure for real world application.	1 to 5,9,10,12	1,2,3
CO6	Experiment with all data structures in a high-level language for problem solving.	1 to 5,9,10,12	1,2,3

BLOOM'S LEVELOF THECOURSE OUTCOMES

CO#	Bloom's Level					
	Remember (L1)	Understand (L2)	Apply (L3)	Analyze (L4)	Evaluate (L5)	Create (L6)
CO1			√			
CO2			√			
CO3			√			
CO4			√			
CO5			√			
CO6			√			

COURSE ARTICULATION MATRIX

CO#/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	2	2	3	2				3	3		3	3		
CO2	3	2	3	3	2				3	3		3	3	3	
CO3	3	1	2	3	1				3	3		3	3	3	
CO4	3	1	3	3	2				3	3		3	3	3	
CO5	3	3	3	3	1				3	3		3	3	3	3
CO6	3	3	3	3	2				3	3		3	3	3	3

IV.GUIDELINES TO THE STUDENTS

1. Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
2. Students are supposed to occupy the systems allotted to them and are not supposed to talk ormake noise in the lab.
3. Students are required to carry their observation book and lab records with completed exerciseswhile entering the lab.
4. Lab records need to be submitted every week.
5. Students are not supposed to use pen drives in the lab.

V. LABREQUIREMENTS

Recommended System/Software Requirements:

- Intel based desktop PC with minimum of 2.6GHZ or faster processor with at least 256 MB RAM and 40GB free disk space.
- Operating system: Flavor of any WINDOWS.
- Software:j2sdk1.7.
- Eclipse or Netbeans.

Use eclipse or Netbean platform and acquaint with the various menus, create a testproject, add a test class and run it see how you can use auto suggestions, auto fill. Trycode formatter and code refactoring like renaming variables, methods and classes. Trydebug step by step with a small program of about 10 to 15 lines which contains at leastone if else condition and a for loop.

Program:-

```
public class Prog1
{
	public static void main(String[] args)
	{
		System.out.println("\n Prog. is showing even no");
		for(int i=2;i<=20;i++)
		{
```

```
                    if(i%2==0)
                    {
                        System.out.print("\n "+i);
                    }
                }
        }
}
```

Commands for executing a JAVA code in command line

Compile:

The command **javac**is used to compile the java code and it is used as shown below. Make sure that the name of the java file must be same as the public class name that contain the main() function.

D:>**javac** Prog_name.java

Run:

The command **java** is used to execute the java code in the Java Virtual Machine and it is used as shown below:

D:>**java** Prog_name

Program – 1

Problem Statement

Write a Java program using the data structure arrays to multiply two given matrices of same order.

Student Learning Outcomes

The students are able to develop an application using Array data structure.

Algorithm

The definition of matrix multiplication is that if $C = AB$ for an $n \times m$ matrix A and an $m \times p$ matrix B, then C is an $n \times p$ matrix with entries

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}.$$

From this, a simple algorithm can be constructed which loops over the indices i from 1 through n and j from 1 through p, computing the above using a nested loop:

- Input: matrices A and B
- Let C be a new matrix of the appropriate size
- For i from 1 to n:
  - For j from 1 to p:
    - Let sum = 0
    - For k from 1 to m:
      - Set sum ← sum + $A_{ik} \times B_{kj}$
    - Set C_{ij} ← sum
- Return C

Program

```
package Experiment1;
import java.util.*;
 public class MatrixApp
 {
      public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);
            int order;
            System.out.println("Matrix Multiplication of same
order");
            System.out.print("Enter the order: ");
            order = read.nextInt();
            Matrix A = new Matrix(order);
            Matrix B = new Matrix(order);
            System.out.println("Enter the elements of 1st Matrix");
            A.readMat(read);
            System.out.println("Enter the elements of 2nd Matrix");
            B.readMat(read);
            Matrix C = A.matMultipy(B);
            System.out.println("1st Matrix");
            A.printMat();
            System.out.println("2nd Matrix");
            B.printMat();
            System.out.println("Product Matrix");
            C.printMat();
```

```
            read.close();
      }
 }
class Matrix
{
      int mat[][];
      int row, col;

      Matrix()
      {
            mat = null;
            row = col = 0;
      }
      Matrix(int order)
      {
            mat = new int[order][order];
            row = col = order;
      }
      void readMat(Scanner read)
      {
            for(int r = 0; r < row; r++)
                  for(int c = 0; c < col; c++)
                        mat[r][c] = read.nextInt();
      }
      void printMat()
      {
            for(int r = 0; r < row; r++)
            {
                  for(int c = 0; c < col; c++)
                        System.out.print(mat[r][c] + " ");
                  System.out.println();
            }
      }
      Matrix matMultipy(Matrix m)
      {
            Matrix prod = new Matrix(this.row);
            for(int i = 0; i <this.row; i++)
                  for (int j = 0; j <this.col; j++)
                  {
                        prod.mat[i][j] = 0;
                        for(int k = 0; k <this.row; k++)
                              prod.mat[i][j] = prod.mat[i][j] +
this.mat[i][k] * m.mat[k][j];
                  }
            return prod;
      }
}
```

Description of the Program

class Matrix

1. Matrix() – The constructor where row and column variables are initialised
2. Matrix(int order)- The constructor where row and column variables are assigned with order of the matrix.
3. void readMat(Scanner read) – Function reads the matrix elements into two-dimensional array mat[r][c].

4. void printMat()- Function prints the array elements in the matrix form.
5. Matrix matMultipy(Matrix m)- Performs the matrix multiplication according to the algorithm.

Main Class MatrixApp

1. Reads the order of the matrix from the user.
2. Reads the array elements of 1st and 2nd matrix from the user using A.readMat(read) and B.readMat(read)
3. Print the array elements in the matrix form by using A.printMat()and B.printMat()
4. The Prod value returned by Matrix matMultipy(Matrix m)to Matrix C = A.matMultipy(B)in main function.
5. Print the product matrix using C.printMat()

Input Output

```
Matrix Multiplication of same order
Enter the order: 3
Enter the elements of 1st Matrix
4 5 3
9 2 1
1 2 4
Enter the elements of 2nd Matrix
6 4 2
1 3 5
7 8 9
1st Matrix
4 5 3
9 2 1
1 2 4
2nd Matrix
6 4 2
1 3 5
7 8 9
Product Matrix
50 55 60
63 50 37
36 42 48
```

Program – 2

Problem Statement

Develop a program in java to read a **sparse matrix**of integer values in the 2D array format and convert the sparse matrix to *<row, column, value>* format and search for an element specified by the user. Print the result of the search appropriately.

Student Learning Outcomes

The students are able to develop an application using Array data structure.

Algorithm

The sparse matrix or **sparse array** is a matrix in which most of the elements are zero.

- Input number of rows, number of columns and accordingly the elements in the spare matrix[rw][cl] where rw-no. of rows, cl-no.of columns.
- Let size=0, k=0, index=0.
- Define result matrix as int resultMatrix[][] = new int[3][size]
- For row from 0 to rw
  - For column from 0 to cl
    - If the spare matrix element is not equal to 0, then

Increment 'size'

resultMatrix[0][k] = row;
resultMatrix[1][k] = column;
resultMatrix[2][k] = sparseMatrix[row][column];

Increment 'k'.

- Input 'key' element to be searched.
- For index from 0 to size
  - If resultMatrix[2][index] == key, then

Print the element found at (resultMatrix[0][index] , resultMatrix[1][index]

Program

```
package Experiment2;
import java.util.*;
public class SparceMatrixApp
{
      public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);

            System.out.println("2D Array format for Sparse Matrix");
            System.out.print("Enter Number of Rows :");
            int rw = read.nextInt();
            System.out.print("Enter Number of Columns :");
            int cl = read.nextInt();
            int sparseMatrix[][] = new int[rw][cl];

           //Read Sparse Matrix in 2D array
           System.out.println("Enter the Values of Matrix:");
           for (int row = 0; row <rw; row++)
               for (int column = 0; column < cl; column++)
            sparseMatrix[row][column] = read.nextInt();

           // Finding total non-zero values in the sparse matrix
           int size = 0;
           for (int row = 0; row <rw; row++)
               for (int column = 0; column < cl; column++)
```

```
                if (sparseMatrix[row][column] != 0)
                    size++;

        // Defining result Matrix
        int resultMatrix[][] = new int[3][size];

        // Generating result matrix
        int k = 0;
        for (int row = 0; row <rw; row++)
            for (int column = 0; column < cl; column++)
                if (sparseMatrix[row][column] != 0)
                {
    resultMatrix[0][k] = row;
    resultMatrix[1][k] = column;
    resultMatrix[2][k] = sparseMatrix[row][column];
                    k++;
                }

        // Displaying result matrix
        System.out.println("Triplet Representation : ");
        for (int row=0; row<3; row++)
        {
          if (row == 0) System.out.print("Row:\t");
          if (row == 1) System.out.print("Column:\t");
          if (row == 2) System.out.print("Value:\t");
             for (int column = 0; column<size; column++)
    System.out.print(resultMatrix[row][column]+" ");
             System.out.println();
        }

        //Searching an element in sparse matrix
    System.out.print("Enter an element to search in sparse matrix:");
        int key = read.nextInt();
        int flag = 0;
        for(int index = 0; index < size; index++)
        {
          if(resultMatrix[2][index] == key)
          {
                System.out.println("Element found at (" +
resultMatrix[0][index] + "," +resultMatrix[1][index] + ")");
                flag = 1;
          }
        }
        if(flag == 0)
          System.out.println("Element not found");
        read.close();
    }
}
```

Description of the Program

1. Read the number of rows and columns of the sparse matrix from the user
2. Read the Sparse Matrix elements entered by user in 2D array
3. The variable ‘size’ counts the total number of non-zero elements by checking every element in the sparse matrix.
4. Define the result Matrix by int resultMatrix[][] = new int[3][size];
5. Generate and display the result matrix by following steps:

- Check every element in the spare matrix which is not equal to 0.
- If the element is non-zero, the corresponding row number, column number and the non-zero element will be stored in the resultMatrix[][](row0, row1 and row2 of column0).
- The next non-zero element of sparse matrix will be stored inresultMatrix[][](row0, row1 and row2 of column1) . This continues for all the non-zero elements.
- Display the generated resultMatrix[][].

6. Read the 'key' element to be searched from the user.
7. Match the 'key' element with the resultMatrix[2][index] for the index from 0 to 'size'
8. When the element is found, the corresponding row number from resultMatrix[0][index] and column number from resultMatrix[1][index] has to be displayed.

Input Output

```
2D Array format for Sparse Matrix
Enter Number of Rows :5
Enter Number of Columns :6
Enter the Values of Matrix:
0 0 0 0 9 0
0 8 0 0 0 0
4 0 0 2 0 0
0 0 0 0 0 5
0 0 2 0 0 0
Triplet Representation :
Row:   0 1 2 2 3 4
Column:4 1 0 3 5 2
Value: 9 8 4 2 5 2
Enter an element to search in sparse matrix:2
Element found at (2,3)
Element found at (4,2)
```

Program – 3

Problem Statement

Write Java programs to implement the STACK ADT using an array.

Student Learning Outcomes

The students are able to develop an application using Stack data structure.

Algorithm

Input the size of the stack, choice of the stack operation

1. Push element in stack
2. Pop an element from Stack
3. Display the Stack
4. Exit

Let array be int a[SIZE]
Let top = -1, MAXSIZE=SIZE

```
bool isfull()
if (top==MAXSIZE)return true
elsereturn false
endif

boolisempty()
if(top ==-1)returntrue;
elsereturnfalse;
endif

begin procedure push(data)
if stack is fullreturnnull
endif
top ← top +1
stack[top]← data
end procedure

begin procedure pop(data)
if stack is emptyreturnnull
endif
data ← stack[top]
top ← top -1
return data
end procedure

begin procedure print(data)
if stack is not empty
  for i ←top to -1
    return data
    else
          return null
endif
end procedure
```

Program

```
package Experiment3;
import java.util.Scanner;
public class ArrayStackApp
{
      public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);
            boolean flag = true;
            System.out.println("Stack Implementation Using Array");
            System.out.print("Enter the size of the Stack: ");
            int size = read.nextInt();
            ArrayStack stack = new ArrayStack(size);
            while(flag)
            {
                  System.out.println("1. Push element in stack");
                  System.out.println("2. Pop an element from Stack");
                  System.out.println("3. Display the Stack");
                  System.out.println("4. Exit");
                  System.out.print("Select your option: ");
                  int ch = read.nextInt();
                  switch(ch)
                  {
                  case 1: System.out.print("Enter the element to push:
");
                              int ele = read.nextInt();
                              if (stack.push(ele))
                                    System.out.println(ele + "
successfully pushed on stack");
                              else
                                    System.out.println("Stack Overflow");
                              break;
                  case 2: ele = stack.pop();
                              if(ele == -1)
                                    System.out.println("Stack Underflow");
                              else
                                    System.out.println(ele + " popped out
of the stack");
                              break;
                  case 3: System.out.println("Stack contents");
                              stack.printStack();
                              break;
                  case 4: flag = false;
                              break;
                  default: System.out.println("Inavlid Choice try
again...");
                  }
            }
            read.close();
      }
}
class ArrayStack
{
    int top;
    int maxSize;
    int a[];
    ArrayStack(int size)
```

```
    {
      a = new int[size];
         top = -1;
maxSize = size;
    }
    void printStack()
    {
      if(!isEmpty())
      {
            for(int i = top; i > -1; i--)
                 System.out.println(a[i]);
      }
      else
            System.out.println("Stack is Empty");
    }
    boolean isEmpty()
    {
        return (top < 0);
    }
    boolean isFull()
    {
      return (top >= (maxSize-1));
    }

    boolean push(int x)
    {
        if (isFull())
      return false;
        else
        {
            a[++top] = x;
            return true;
        }
    }

    int pop()
    {
        if (isEmpty())
            return -1;
        else
      return a[top--];
    }
}
```

Description of the Program

Class ArrayStack

- Create a one dimensional array (**int a[SIZE]**) with size read from the user to store stack elements.
- Define a integer variable '**top**' and initialize with '**-1**'. (**int top = -1**) and MAXSIZE=SIZE
- printStack() **- Prints the elements of a Stack according to the algorithm**
- **push(value) - Inserting value into the stack according to the algorithm**
- **pop() - Delete a value from the Stack according to the algorithm**

Class ArrayStackApp

- Read the size of stack from user.
- Read the option from the user 1.Push 2.Pop 3.Display 4.Exit
- Using Switch case,
  - Case 1: To push the element if the stack is not full. If it is full then print "Stack overflow"
  - Case 2:To pop the element if the stack is not empty. If it is empty then print "stack underflow"
  - Case 3:Print the contents of the stack.
  - Case 4:Exit.
- If any other option is given then print "Invalid Choice try again..."

Input Output

```
Stack Implementation Using Array
Enter the size of the Stack: 2
1. Push element in stack
2. Pop an element from Stack
3. Display the Stack
4. Exit
Select your option: 1
Enter the element to push: 10
10 successfully pushed on stack
1. Push element in stack
2. Pop an element from Stack
3. Display the Stack
4. Exit
Select your option: 1
Enter the element to push: 20
20 successfully pushed on stack
1. Push element in stack
2. Pop an element from Stack
3. Display the Stack
4. Exit
Select your option: 1
Enter the element to push: 30
Stack Overflow
1. Push element in stack
2. Pop an element from Stack
3. Display the Stack
4. Exit
Select your option: 3
Stack contents
20
10
1. Push element in stack
2. Pop an element from Stack
3. Display the Stack
4. Exit
Select your option: 4
```

Program - 4

Problem Statement

Write Java programs to implement the QUEUE ADT using an array.

Student Learning Outcomes

The students are able to develop an application using Queue data structure.

Algorithm

Input the size of the queue, choice of the queue operation

1. Enqueue element in stack
2. Dequeue an element from Stack
3. Display the queue
4. Exit

Let array be int arr[SIZE]

Let capacity = size, front = 0, rear = -1, count = 0

bool isfull()

if rear == MAXSIZE – 1 return true;
else return false;
endif

bool isempty()

if front < 0 or front > rear return true;
else return false;
endif

procedure enqueue(data)

if queue is full return overflow
endif
rear ← rear + 1
queue[rear] ← data
return true
end procedure

procedure dequeue

if queue is empty return underflow
end if
data = queue[front]
front ← front + 1
return true
end procedure

procedure printQueue(data)

if queue is not Empty
for i= front to rear
return data
else
return false
endif
end procedure

Program

```
package Experiment4;
import java.util.*;
public class ArrayQueueApp
```

```
{
      public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);
            boolean flag = true;
            System.out.println("Queue Implementation Using Array");
            System.out.print("Enter the size of the Queue: ");
            int size = read.nextInt();
            ArrayQueue queue = new ArrayQueue(size);
            while(flag)
            {
                  System.out.println("1. Add an element in to queue");
                  System.out.println("2. Remove an element from queue");
                  Sy
stem.out.println("3. Display the queue");
                  System.out.println("4. Exit");
                  System.out.print("Select your option: ");
                  int ch = read.nextInt();
                  switch(ch)
                  {
                  case 1: System.out.print("Enter the element to add:
");
                              int ele = read.nextInt();
                              if (queue.enqueue(ele))
                                    System.out.println(ele + "
successfully added to queue");
                              else
                                    System.out.println("Queue Overflow");
                              break;
                  case 2: ele = queue.dequeue();
                              if(ele == -1)
                                    System.out.println("Queue Underflow");
                              else
                                    System.out.println(ele + " dequeued
from queue");
                              break;
                  case 3: System.out.println("Queue contents");
                              queue.printQueue();
                              break;
                  case 4: flag = false;
                              break;
                  default: System.out.println("Inavlid Choice try
again...");
                  }
            }
            read.close();
      }
}
//Class for queue
class ArrayQueue
{
      private int arr[];          // array to store queue elements
      private int front;          // front points to front element in
the queue (if any)
      private int rear;           // rear points to last element in the
queue
```

```
    private int capacity; // maximum capacity of the queue
    private int count;          // current size of the queue

    // Constructor to initialize queue
    ArrayQueue(int size)
    {
        arr = new int[size];
        capacity = size;
        front = 0;
        rear = -1;
        count = 0;
    }
    // Utility function to remove front element from the queue
    public int dequeue()
    {
        // check for queue underflow
        if (isEmpty())
            return -1;
        else
        {
            int ele = arr[front];
            front = (front + 1) % capacity;
            count--;
            return ele;
        }
    }
    // Utility function to add an item to the queue
    public boolean enqueue(int item)
    {
        // check for queue overflow
        if (isFull())
            return false;
        else
        {
            rear = (rear + 1) % capacity;
            arr[rear] = item;
            count++;
            return true;
        }
    }
    // Utility function to return front element in the queue
    public void printQueue()
    {
        if(!isEmpty())
    {
        for(int i = front; i <= rear; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
    else
        System.out.println("Queue is Empty");
    }
    // Utility function to return the size of the queue
    public int size()
    {
        return count;
```

```
	}
	// Utility function to check if the queue is empty or not
	public Boolean isEmpty()
	{
		return (size() == 0);
	}
	// Utility function to check if the queue is empty or not
	public Boolean isFull()
	{
		return (size() == capacity);
	}
}
```

Description of the Program

Class ArrayQueue

- ArrayQueue(int size) is the Constructor to initialize queue
  - Initialise an array (**int arr[SIZE]**) with size read from the user to store queue elements.
  - Set capacity = size, front = 0, rear = -1, count = 0 (capacity – maximum size of the queue, count- current size of the queue)
- dequeue()- Function used to delete an element from the queue. In a queue, the element is always deleted from **front** position according to the algorithm.
- enqueue()- Function used to insert a new element into the queue. In a queue, the new element is always inserted at **rear** position according to the algorithm.
- printQueue() – Function used to print the contents of the queue according to the algorithm.

class ArrayQueueApp

- Read the size of queue from user.
- Read the option from the user 1.enqueue 2.dequeue 3.Display 4.Exit
- Using Switch case,
  - Case 1: To enqueue the element if the queue is not full. If it is full then print "Queue overflow"
  - Case 2:To dequeue the element from the queue if it is not empty. If it is empty then print "queue underflow"
  - Case 3:Print the contents of the queue.
  - Case 4:Exit.
- If any other option is given then print "Invalid Choice try again..."

Input Output

```
Queue Implementation Using Array
Enter the size of the Queue: 2
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 1
Enter the element to add: 10
10 successfully added to queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 1
Enter the element to add: 20
```

```
20 successfully added to queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 1
Enter the element to add: 30
Queue Overflow
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 3
Queue contents
10 20
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 2
10 dequeued from queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 2
20 dequeued from queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 4
```

Program - 5

Problem Statement

The compilers always convert infix expression into postfix to perform further operations like parsing, lexical analysis etc. Select an appropriate data structure and develop a program to convert an infix expression into postfix using **stack.**

Student Learning Outcomes

The students are able to develop an application using Stack data structure.

Algorithm

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

- *Push "(" onto Stack, and add ")" to the end of X.*
- *Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.*
- *If an operand is encountered, add it to Y.*
- *If a left parenthesis is encountered, push it onto Stack.*
- *If an operator is encountered ,then:*
  - *Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.*
  - *Add operator to Stack.*
    endif
- *If a right parenthesis is encountered ,then:*
  - *Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.*
  - *Remove the left Parenthesis.*
    end if
    end if
- *END.*

Program

```
package Experiment5;
import java.util.Scanner;
public class InfixToPostfixApp
{
	static Stack operators = new Stack();
	public static void main(String[] args)
	{
		String infix;
		Scanner read = new Scanner(System.in);
		System.out.print("\nEnter the algebraic expression in infix:
");
		infix = read.nextLine();
		//output as postfix
		System.out.println("The expression in postfix is:" +
toPostfix(infix));
		read.close();
	}
	private static String toPostfix(String infix)
	{
		char symbol;
		String postfix = "";
```

```
          for(int i=0;i<infix.length();++i) //while there is input to
be read
           {
                symbol = infix.charAt(i);       //if it's an operand,
add it to the string
                if (Character.isSpaceChar(symbol))
                     continue;
              if (Character.isLetter(symbol))
           postfix = postfix + symbol;
                else if (symbol=='(')           //push (
                     operators.push(symbol);
                else if (symbol==')')                 //push
everything back to (
                {
                     while (operators.peek() != '(')
                          postfix = postfix + operators.pop();
                     operators.pop();      //remove '('
                }
                else //print operators occurring before it that have
greater precedence
                {
                     while (!operators.isEmpty() &&
!(operators.peek()=='(') &&prec(symbol) <= prec(operators.peek()))
                          postfix = postfix + operators.pop();
                     operators.push(symbol);
                }
           }
           while (!operators.isEmpty())
                postfix = postfix + operators.pop();
           return postfix;
     }
     static int prec(char x)
     {
           if (x == '+' || x == '-')
                return 1;
           if (x == '*' || x == '/' || x == '%')
                return 2;
           return 0;
     }
}
class Stack
{
     char a[] = new char[100];
     int maxSize = 50;
     int top = -1;

     boolean isFull()
    {
     return (top >= (maxSize-1));
    }

    void push(char c)
    {
        if (isFull())
        {
     System.out.println("Stack full , no room to push , size=50");
```

```
                System.exit(0);
            }
            else
                a[++top] = c;
        }
        char pop()
        {
            return a[top--];
        }
        boolean isEmpty()
        {
            return (top == -1) ? true : false;
        }
        char peek()
        {
            return a[top];
        }
}
```

Description of the Program

class InfixToPostfixApp

- Input the algebraic expression in infix
- Declare String infix, postfix=""
- Length - length of the infix string
- Scan the infix string from 0 to length
- Character.isSpaceChar(symbol) – If the symbol is space then continue without doing anything.
- Character.isLetter(symbol) – If the symbol is operand then add it to the postfix expression
- If symbol is '(', then push '(' on to the stack
- If symbol is ')', then pop everything till ')' and add it to the postfix string.
- If the operator occurring before ')' have greater precedence, then,
    - The operator precedence is checked by using the function prec(char x). x is the operator. If x = '*' or x = '/' or x = '%' have higher precedence. If x = '+' or x = '-' have lower precedence.
    - Pop the operator which has higher precedence first and add it to the postfix string.
- Peek() - Method in Java is used to retrieve or fetch the first element of the Stack or the element present at the top of the Stack. The element retrieved does not get deleted or removed from the Stack.

class Stack – performs push and pop operation of the operators on to the stack

Input Output

```
Enter the algebraic expression in infix: a*b/(c+d)

The expression in postfix is: ab*cd+/
```

Program - 6

Problem Statement

Write Java programs to implement the STACK ADT using a singly linked list.

Student Learning Outcomes

The students are able to develop an application using Stack data structure.

Algorithm

Input the choice of the stack operation

1. Enqueue element in stack
2. Dequeue an element from Stack
3. Display the queue
4. Exit

Define node with data and next=null

procedure push(data)
Define a newNode with given value.
set newNode → next = top.
set top = newNode.
Return top is not equal to null
End procedure

bool isEmpty
return when top == null
end

procedure pop()
define ele = -1
if stack is Empty
return -1;
else if top not equal to null
set ele = top.data
set top = top.next
else
set top = null
return ele
end if
end procedure

procedure printStack(data)
Assign Node n=top
While n until not equal to null
Print n.data
Set n = n.next
End while
End procedure

Program

```
package Experiment6;
import java.util.*;
public class LinkedStackApp
{
      public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);
            System.out.println("Linked List Stack");
            LinkedListStack stack = new LinkedListStack();
            boolean flag = true;
            while(flag)
            {
                  System.out.println("1. Push element in stack");
                  System.out.println("2. Pop an element from stack");
                  System.out.println("3. Display the stack");
                  System.out.println("4. Exit");
                  System.out.print("Select your option: ");
                  int ch = read.nextInt();
                  switch(ch)
                  {
                  case 1: System.out.print("Enter the element to push:
");
                              int ele = read.nextInt();
                              if (stack.push(ele))
                                    System.out.println(ele + "
successfully pushed on stack");
                              else
                                    System.out.println("Stack Overflow");
                              break;
                  case 2: ele = stack.pop();
                              if(ele == -1)
                                    System.out.println("Stack Underflow");
                              else
                                    System.out.println(ele + " popped out
of the stack");
                              break;
                  case 3: System.out.println("Stack contents");
                              stack.printStack();
                              break;
                  case 4: flag = false;
                              break;
                  default: System.out.println("Inavlid Choice try
again...");
                  }
            }
            read.close();
      }
}
class LinkedListStack
{
    Node top;  // head of list
LinkedListStack() { top = null;}

    class Node /* Linked list Node*/
    {
      int data;
```

```
      Node next;

         Node(int d)
         {
      data = d;
      next = null;
         }
    }
    void printStack()
    {
        Node n = top;
        while (n != null)
        {
            System.out.println(n.data);
            n = n.next;
        }
    }
    boolean push(int data)
    {
        Node node = new Node(data);
node.next = top;
        top = node;
        return (top != null);
    }
    boolean isEmpty()
    {
      return (top == null);
    }
    int pop()
    {
      int ele = -1;
        if(isEmpty())
            return -1;
        else if(top != null)
        {
ele = top.data;
            top = top.next;
        }
        else
      top = null;
        return ele;
    }
}
```

Description of the Program

class LinkedListStack

- Define a '**Node**' structure with two members **data** and **next**.
- printStack() - Prints the elements of a Stack according to the algorithm
- push(value) - Inserting value into the stack according to the algorithm
- pop() - Delete a value from the Stack according to the algorithm

Class LinkedStackApp

- Read the option from the user 1.Push 2.Pop 3.Display 4.Exit
- Using Switch case,

- Case 1: To push the element if the stack is not full. If it is full then print "Stack overflow"
- Case 2:To pop the element if the stack is not empty. If it is empty then print "stack underflow"
- Case 3:Print the contents of the stack.
- Case 4:Exit.

- If any other option is given then print "Invalid Choice try again..."

Input Output

```
Linked List Stack
1. Push element in stack
2. Pop an element from stack
3. Display the stack
4. Exit
Select your option: 1
Enter the element to push: 10
10 successfully pushed on stack
1. Push element in stack
2. Pop an element from stack
3. Display the stack
4. Exit
Select your option: 1
Enter the element to push: 20
20 successfully pushed on stack
1. Push element in stack
2. Pop an element from stack
3. Display the stack
4. Exit
Select your option: 3
Stack contents
20
10
1. Push element in stack
2. Pop an element from stack
3. Display the stack
4. Exit
Select your option: 2
20 popped out of the stack
1. Push element in stack
2. Pop an element from stack
3. Display the stack
4. Exit
Select your option: 2
10 popped out of the stack
1. Push element in stack
2. Pop an element from stack
3. Display the stack
4. Exit
Select your option: 4
```

Program – 7

Problem Statement

Evaluation of postfix expressions is done by compilers during the compilation process. Design and Develop a program to evaluate a postfix expression using **stack**.

Student Learning Outcomes

The students are able to develop an application using Stack data structure.

Algorithm

- *Read postfix expression Left to Right from the user.*
  - *If operand is encountered,*
    - *push it onto Stack*
  - *End If*
  - *If operator is encountered, Pop two element*
    - *A -> Top element*
    - *B-> Next to Top element*
    - *Evaluate B operator A*
    - *push B operator A onto Stack*
- *End if*
  Set result = pop
- *END*

Program

```
package Experiment7;
import java.util.*;
public class PostfixEvalApp
{
      public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);
            System.out.print("\nEnter the expression in postfix: ");
            String exp = read.nextLine();
         System.out.println("Result = " + evaluatePostfix(exp));
         read.close();
      }
      static int evaluatePostfix(String exp)
     {
            Stack stack = new Stack();
         for(int i = 0; i <exp.length(); i++)
         {
             char c = exp.charAt(i);
             if(Character.isSpaceChar(c))
      continue;
             else if(Character.isAlphabetic(c))
             {
      System.out.print("\nExpression should conatin only digits");
      System.exit(0);
             }
             else if(Character.isDigit(c))
             {
                 int n = 0;
                 while(Character.isDigit(c))
                 {
                     n = n * 10 + (int)(c - '0');
                     i++;
```

```
                    c = exp.charAt(i);
                }
                i--;
                stack.push(n);
            }
            else
            {
                int val1 = stack.pop();
                int val2 = stack.pop();
                switch(c)
                {
                    case '+':    stack.push(val2+val1);
                break;
                    case '-':    stack.push(val2- val1);
                break;
                    case '/':    stack.push(val2/val1);
                break;
                    case '*':    stack.push(val2*val1);
                break;
                }
            }
        }
        return stack.pop();
    }
}
class Stack
{
    Node top;  // head of list
    Stack() { top = null;}

    class Node /* Linked list Node*/
    {
      int data;
      Node next;

         Node(int d)
         {
      data = d;
      next = null;
         }
    }
    void push(int data)
    {
        Node node = new Node(data);
node.next = top;
        top = node;
    }
    boolean isEmpty()
    {
      return (top == null);
    }
    int pop()
    {
      int ele = -1;
         if(isEmpty())
             return -1;
```

```
        else if(top != null)
        {
ele = top.data;
            top = top.next;
        }
        else
      top = null;
        return ele;
    }
    int peek()
    {
     return top.data;
    }
}
```

Description of the Program

evaluatePostfix(String exp)

- for I =0 to length of the expression,
- c=exp(i)
- Character.isSpaceChar(c) – If 'c' is space, then continue without doing anything.
- Character.isAlphabetic(c) – If 'c' is alphabet, then print "Expression should contain only digits"
- If 'c' is digit, then push it on to the stack
- If 'c' is operator then
    - Pop two items say val1 and val2 from the stack
    - Switch case does +, -, \,* operations between the popped values.
- End of for loop
- Pop the result value on top of the stack

class Stack:

- Define a '**Node**' structure with two members **data** and **next**.
- push(value) - Inserting value into the stack according to the stack implementation using linked list algorithm
- pop() - Delete a value from the Stack according to the stack implementation using linked list algorithm

class PostfixEvalApp:

- Read the postfix expression from the user
- The value returned by the evaluatePostfix(String exp) is assigned to the variable result and it is printed.

Input Output

```
Enter the expression in postfix: 20 35 + 60 *

Result = 3300
```

Program - 8

Problem Statement

Write Java programs to implement the QUEUE ADT using a singly linked list.

Student Learning Outcomes

The students are able to develop an application using Linked List data structure.

Algorithm

Input the choice of the stack operation

1. Enqueue element in stack
2. Dequeue an element from Stack
3. Display the queue
4. Exit

Define node with members data and next=null

procedure enqueue(val) *// Insertion at end of the list*

- *Define a new node PTR*
- *Set ptr -> data = val*
- *If front = null*

  Set front = rear = ptr

  else

  Set rear -> next = ptr

  Set rear = ptr

  Set rear -> next = null

  End if

End procedure

procedure dequeue() *// Deletion at beginning of the list*

- *If front = null then*

  return false
- *End if*
- *Set ptr = front*
- *Set front = front -> next*
- *If front = null*

return temp.data

- *End If*

End procedure

procedure printStack(data)

- *Assign Node n=front*
- *While n until not equal to null*

  Print n.data

  Set n = n.next
- *End while*

End procedure

Program

```
package Experiment8;
import java.util.*;
public class LinkedQueueApp
{
	public static void main(String[] args)
	{
		Scanner read = new Scanner(System.in);
		System.out.println("Linked List Queue");
		LinkedListQueue queue = new LinkedListQueue();
		boolean flag = true;
		while(flag)
		{
			System.out.println("1. Add an element in to queue");
			System.out.println("2. Remove an element from queue");
			System.out.println("3. Display the queue");
			System.out.println("4. Exit");
			System.out.print("Select your option: ");
			int ch = read.nextInt();
			switch(ch)
			{
			case 1: System.out.print("Enter the element to add:
");
				int ele = read.nextInt();
				if (queue.enqueue(ele))
					System.out.println(ele + "
successfully added to the queue");
				else
					System.out.println("Queue Overflow");
				break;
			case 2: ele = queue.dequeue();
				if(ele == -1)
					System.out.println("Queue Underflow");
				else
					System.out.println(ele + " removed out
of queue");
				break;
			case 3: System.out.println("Queue contents");
				queue.printQueue();
				break;
			case 4: flag = false;
				break;
			default: System.out.println("Inavlid Choice try
again...");
			}
		}
		read.close();
	}
}
class LinkedListQueue
{
	private class Node
	{
		int data;	// integer data
		Node next;	// pointer to the next node
		public Node(int data)
		{
```

```
                // set the data in allocated node and return the node
                this.data = data;
                this.next = null;
          }
     }
     private Node rear, front;
     LinkedListQueue()
     {
          front = rear = null;
     }
     int dequeue()    // delete at the beginning
     {
          if (front == null)
               return -1;
          Node temp = front;
          // advance front to the next node
          front = front.next;
          // if list becomes empty
          if (front == null)
               rear = null;
          return temp.data;
     }
     boolean enqueue(int item)  // insertion at the end
     {
          Node node = new Node(item);
          // special case: queue was empty
          if (front == null)
          {
               front = node;
               rear = node;
          }
          else
          {
               rear.next = node;
               rear = node;
          }
          return (rear != null);
     }
     void printQueue()
     {
          Node n = front;
        while (n != null)
        {
System.out.print(n.data + " ");
            n = n.next;
        }
        System.out.println();
     }
     boolean isEmpty() {
          return rear == null && front == null;
     }
}
```

Description of the Program

Brief Description of the Program:
class LinkedListQueue

- Define a '**Node**' structure with two members **data** and **next**.
- printQueue() - Prints the elements of a Stack according to the algorithm
- enqueue(value) - Inserting value into the stack according to the algorithm
- dequeue() - Delete a value from the Stack according to the algorithm

Class LinkedQueueApp

- Read the option from the user 1.Enqueue 2.Dequeue 3.Display 4.Exit
- Using Switch case,
  - Case 1: To enqueue the element if the queue is not full. If it is full then print "Queue overflow"
  - Case 2:To dequeue the element if the queue is not empty. If it is empty then print "queue underflow"
  - Case 3:Print the contents of the queue.
  - Case 4:Exit.
- If any other option is given then print "Invalid Choice try again..."

Input Output

```
Linked List Queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 1
Enter the element to add: 10
10 successfully added to the queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 1
Enter the element to add: 20
20 successfully added to the queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 3
Queue contents
10 20
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 2
10 removed out of queue
1. Add an element in to queue
2. Remove an element from queue
3. Display the queue
4. Exit
Select your option: 4
```

Program – 9

Problem Statement

Write a java program that determines whether parenthetic symbols ( ), { } and [ ] are nested correctly in a string of characters (use stack ADT).

Student Learning Outcomes

The students are able to develop an application using Stack data structure.

Algorithm

- *procedure checkbalance*
- *Input the string with parenthesis [ ], { }, or ( )*
- *Declare a character stack.*
- *Now traverse the expression string str.*
- *For i=0 to length of str*
  1. *If the current character is a starting bracket (**'(' or '{' or '['**) then push it to stack.*
  2. *If the current character is a closing bracket (**')' or '}' or ']'**) then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.*
- *After complete traversal, if there is some starting bracket left in stack then "not balanced" otherwise "balanced"*
- *End procedure*

Program

```
package Experiment9;
import java.util.*;
public class BracketTestApp
{
public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);
      System.out.println("Enter string with parenthesis [],{},or( )");
            String str = read.nextLine();
            if (str.isEmpty())
                  System.out.println("Empty String");
            else
                  System.out.println(checkBalance(str));
            read.close();
      }
      public static String  checkBalance(String str)
      {
            Stack stack = new Stack();
            for (int i = 0; i < str.length(); i++)
            {
                  char ch = str.charAt(i);
                  if (ch == '[' || ch == '(' || ch == '{')
                        stack.push(ch);
      else if ((ch==']' || ch=='}' || ch == ')') && (!stack.isEmpty()))
if(((char)stack.peek()=='(' && ch==')')
                                    ||((char) stack.peek()=='{'&&ch=='}')
                                    || ((char) stack.peek() == '[' && ch
== ']'))
```

```
                              stack.pop();
                        else
                              return "Not Balanced";
                  else
                        if ((ch == ']' || ch == '}' || ch == ')'))
                              return "Not Balanced";
            }
            if (stack.isEmpty())
                  return "Balanced Parenthesis";
            else
                  return "Not Balanced";
      }
}
class Stack
{
      private Object[] data;
      private int top = 0;
      private int size = 0;
      Stack()
      {
            this.size = 30; /* default stack size of 30 */
            data = new Object[this.size];
      }
      void push(Object o)
      {
            if (top >= size)
                  this.increaseSize();
            this.data[top] = o;
            top++;
      }
      Object pop()
      {
            if (top != 0)
            {
                  Object obj = data[top - 1];
                  this.data[top - 1] = null; // Deleted
                  top--;
                  return obj;
            }
            else
                  return null;
      }
      Object peek()
      {
            if (top != 0)
                  return this.data[top - 1];
            else
                  return null;
      }
      boolean isEmpty()
      {
            return top == 0 ? true : false;
      }
      int getStackSize()
      {
            return top;
```

```
	}
	private void increaseSize()
	{
		Object[] temp = new Object[size];
		size = size * 2;
		for (int i = 0; i < top; i++)
			temp[i] = this.data[i];
		this.data = new Object[this.size];
		for (int i = 0; i < top; i++)
			this.data[i] = temp[i];
	}
}
```

Description of the Program

checkBalance(String str)

- Define a new stack
- For I=0 to length of the string
- Scan the scan the string using char ch = str.charAt(i)
- If ch is '[' or '(' or '{' push on to the stack
- If ch is ']' or '}' or ')', check for the corresponding opening bracket. If it is there then pop both opening and closing parenthesis.
- If the stack is empty after complete traversal, then print “Balanced Parenthesis” otherwise print “Not Balanced”

class Stack

- this.size = 30 sets the default stack size to be 30.
- increaseSize() – doubles the size of the stack.
- Push() – if top is equal to size, then increase the size of stack. Then the object parenthesis can be pushed to the top of the stack by this.data[top] = o.
- Pop() – If the top is not null, then pop the object.

BracketTestApp

- Read the input string with parenthesis [ ], { }, or ( ).
- Checks whether the string is empty. If it empty, then print “Empty String”.
- Else call the checkBalance function which will return whether the parenthesis is balanced or not.

Input Output

```
Enter string with parenthesis [ ], { }, or ( )
Jingle (bell) [hello] {[(gone with the wind)])
Not Balanced
Enter string with parenthesis [ ], { }, or ( )
Jingle (bell) [hello] {[(gone with the wind)]}
Balanced Parenthesis
```

Program – 10

Problem Statement

Write a java program that uses both stack and queue to test whether the given string is a palindrome (Use Java Utility).

Student Learning Outcomes

The students are able to develop an application using Stack and Queue data structure.

Algorithm

Input the string and enter the choice

a) *Check palindrome using stack*
b) *Check palindrome using queue*

boolean usingStack(String string)

- *Define new stack*
- *For i=0 to string.lenght()*
  Push(i)
- *Let reverseString = ""*
- *While stack is not empty*
  reverseString=Pop()
- *end while*
- *if string equals reverseString*
  return true
  else
  return false
- *end if*
- *end usingStack*

boolean usingQueue(String string)

- *Define new Queue*
- *For i=string.length()-1 to 0*
  enqueue(i)
- *Let reverseString = ""*
- *While queue is not empty*
  reverseString=dequeue()
- *end while*
- *if string equals reverseString*
  return true
  else
  return false
- *end if*
- *end usingQueue*

Program

```
package Experiment10;
import java.util.*;
public class PalindromeApp
{
      public static void main(String[] args)
      {
            System.out.println("String Palindrome Test");
            Scanner in=new Scanner(System.in);
            System.out.print("Enter any string: ");
            String inputString = in.nextLine();
            System.out.println("a. Check palindrome using stack");
            System.out.println("b. Check palindrome using queue");
            System.out.print("Enter choice: ");
            String ch = in.nextLine();
            switch(ch.charAt(0))
            {
            case 'a':
            case 'A':  if(usingStack(inputString))
                                    System.out.println("The input String "
+ inputString + " is a palindrome.");
                  else
                        System.out.println("The input String " +
inputString + " is not a palindrome.");
                              break;
            case 'b':
            case 'B':  if(usingQueue(inputString))
                                    System.out.println("The input String "
+ inputString + " is a palindrome.");
                              else
                                    System.out.println("The input String "
+ inputString + " is not a palindrome.");
                              break;
            default:   System.out.println("Invalid Choice enter the
character! (TRY AGAIN)");
                              break;
            }
            in.close();
      }
      static boolean usingStack(String string)
      {
            Stack<Character> stack = new Stack<Character>();
         for (int i = 0; i <string.length(); i++)
             stack.push(string.charAt(i));
         String reverseString = "";
         while (!stack.isEmpty())
reverseString = reverseString + stack.pop();
         if (string.equals(reverseString))
      return true;
         else
      return false;
      }
      static boolean usingQueue(String string)
      {
            Queue<Character> queue = new LinkedList<Character>();
```

```
        for (int i = string.length() - 1; i >= 0; i--)
queue.add(string.charAt(i));
        String reverseString = "";
        while (!queue.isEmpty())
reverseString = reverseString + queue.remove();
        if (string.equals(reverseString))
      return true;
        else
      return false;
      }
}
```

Description of the Program

boolean usingStack(String string)

- Define a new stack
- For i=0 to stringlength -1
  - Push the string character by character on to the stack using stack.push(string.charAt(i)).
- Define string reverseString to be "".
- While stack is not empty - As stack is LIFO, the pop operation of the stack will reverse the string and assign it to reverseString
- (string.equals(reverseString)) checks whether the input string matches with the reverseString. If matches then return true otherwise return false.

boolean usingQueue(String string)

- Define a new queue
- For i=stringlength to 0 (For the String in reverse order)
  - Enqueue the reversed string character by character on to the queue using queue.add(string.charAt(i)).
- Define string reverseString to be "".
- While queue is not empty - As queue is FIFO, the dequeue operation will get the string as it is and assign it to reverseString
- (string.equals(reverseString)) checks whether the input string matches with the reverseString. If matches then return true otherwise return false.

class PalindromeApp

- Read the input string from the user and choice
  a) Check palindrome using stack
  b) Check palindrome using queue
- switch case A: Check palindrome using stack by calling the function usingStack(String string) which will return the string is palindrome or not.
- Switch case B: Check palindrome using queue by calling the function usingQueue(String string)which will return the string is palindrome or not.

Input Output

```
String Palindrome Test
Enter any string: MADAM
a. Check palindrome using stack
b. Check palindrome using queue
Enter choice: a
The input String MADAM is a palindrome.

String Palindrome Test
Enter any string: DRAGON
a. Check palindrome using stack
b. Check palindrome using queue
Enter choice: A
```

```
The input String DRAGON is not a palindrome.

String Palindrome Test
Enter any string: MADAM
a. Check palindrome using stack
b. Check palindrome using queue
Enter choice: B
The input String MADAM is a palindrome.

String Palindrome Test
Enter any string: DRAGON
a. Check palindrome using stack
b. Check palindrome using queue
Enter choice: b
The input String DRAGON is not a palindrome.
```

Program – 11

Problem Statement

Files are stored in memory in tree structure directory. Design and develop a program to create a directory having files with unique file-id in the hard disk and display the files in all three traversal orders using Binary Search Tree (BST).

Student Learning Outcomes

The students are able to develop an application using Tree data structure.

Algorithm

Algorithm to implement Preorder

Step1: start P(TreeNode)
Step2: Output(TreeNode.value)
Step3: IF LeftPointer(TreeNode) != NULL Then
Step4: P(TreeNode.LeftNode)
Step 6: IF RightPointer(TreeNode) != NULL Then
Step 7: P(TreeNode.RightNode)
Step9: END

Algorithm to implement Inorder

Step1: start P(TreeNode)
Step 2:IF LeftPointer(TreeNode) != NULL Then
Step 3: P(TreeNode.LeftNode)
Step 5: Output(TreeNode.value)
Step 6: IF RightPointer(TreeNode) != NULL Then
Step 7: P(TreeNode.RightNode)
Step 8:END

Algorithm to implement Postorder

Step 1: start P(TreeNode)
Step 2: IF LeftPointer(TreeNode) != NULL Then
Step 3: P(TreeNode.LeftNode)
Step 5: IF RightPointer(TreeNode) != NULL Then
Step 6: P(TreeNode.RightNode)
Step 8: Output(TreeNode.value)
Step 9: END

Program

```
package Experiment11;
import java.util.*;
public class BSTApp {
      public static void main(String[] args)
      {
            Scanner read = new Scanner(System.in);
            boolean b = true;
            BST btree = new BST();
            while (b)
            {
                  System.out.println("1. Create Files \n2. Traverse \n3.
Exit");
                  System.out.print("Enter Choice: ");
                  int choice = read.nextInt();
                  switch(choice)
                  {
                        case 1:   System.out.println("File Creation:");
                                    System.out.print("Enter file
name: ");
                                    read.nextLine();
                                 String str = read.nextLine();
                  btree.insert(str);
                                    break;
                        case 2:   System.out.print("Traversal of file
structure in:");
                                    System.out.println("Inorder,
Preorder and Postorder");
                                    System.out.println("Inorder:-");
                                    btree.inorder(btree.getRoot());
                                    System.out.println("Preorder:-");
                                    btree.preorder(btree.getRoot());
                                    System.out.println("Postorder:-
");
                                    btree.postorder(btree.getRoot());
                                    break;
                        case 3:   b = false;
                                    break;
                        default: System.out.println("Invalid choice try
again:");
                  }

            }
            read.close();
      }
}
class File
{
      static final Random gen = new Random();
      static int c = 0;
      static int [] random;
      private String fname;
      private int fID;

      {
            random = randomNumbers(50, 100);
      }
```

```
      File(String n)
      {
            fname = new String(n);
            fID = random[c++];
      }
      static int[] randomNumbers(int n, int maxRange)
      {
          int[] result = new int[n];
          Set<Integer> used = new HashSet<Integer>();
          for (int i = 0; i < n; i++)
          {
              int newRandom;
              do
              {
      newRandom = gen.nextInt(maxRange+1);
              } while (used.contains(newRandom));
              result[i] = newRandom;
      used.add(newRandom);
          }
          return result;
      }
      int id()
      {
            return fID;
      }
      public String toString()
      {
            return String.format("File ID -> " + fID + "\tFile name -> "
+ fname);
      }
}
class BST
{
    private class Node
    {
        File data;
        Node left;
        Node right;
        Node(File file)
        {
      data = file;
      left = right = null;
        }
    }
    private Node root;
    private Node insert(File x, Node t)
    {
        if(t == null)
            t = new Node(x);
        else if(x.id() < t.data.id())
t.left = insert(x, t.left);
        else if(x.id() > t.data.id())
t.right = insert(x, t.right);
        return t;
    }
```

```
    void insert(String x)
    {
     File f = new File(x);
        root = insert(f, root);
    }
    void inorder(Node t)
    {
        if(t == null)
            return;
inorder(t.left);
        System.out.println(t.data + " ");
inorder(t.right);
    }
    void preorder(Node t)
    {
        if(t == null)
            return;
        System.out.println(t.data + " ");
preorder(t.left);
preorder(t.right);
    }
    void postorder(Node t)
    {
        if(t == null)
            return;
        postorder(t.left);
        postorder(t.right);
        System.out.println(t.data + " ");
    }
    BST()
    {
        root = null;
    }
    Node getRoot()
    {
     return root;
    }
}
```

Description of the Program

The class BST is used to create a binary search tree.
The class File is used to create files with random file ID.
The insert function insert the file into proper location based on the fileID.
The postorder traversal function traverse the fileIDs in post order traversal till the root is null.
The preorder traversal function traverse the fileIDs in pre order traversal till the root is null.
The inorder traversal function traverse the fileIDs in In order traversal till the root is null.

Input Output

```
1. Create Files
2. Traverse
3. Exit
Enter Choice: 1
File Creation:
Enter file name: Alpha
1. Create Files
2. Traverse
3. Exit
Enter Choice: 1
```

```
File Creation:
Enter file name: Beta
1. Create Files
2. Traverse
3. Exit
Enter Choice: 1
File Creation:
Enter file name: Gamma
1. Create Files
2. Traverse
3. Exit
Enter Choice: 1
File Creation:
Enter file name: Delta
1. Create Files
2. Traverse
3. Exit
Enter Choice: 1
File Creation:
Enter file name: Zeta
1. Create Files
2. Traverse
3. Exit
Enter Choice: 2
Traversal of file structure in:Inorder, Preorder and Postorder
Inorder:-
File ID -> 32       File name -> Delta
File ID -> 58       File name -> Beta
File ID -> 63       File name -> Gamma
File ID -> 76       File name -> Alpha
File ID -> 80       File name -> Zeta
Preorder:-
File ID -> 76       File name -> Alpha
File ID -> 58       File name -> Beta
File ID -> 32       File name -> Delta
File ID -> 63       File name -> Gamma
File ID -> 80       File name -> Zeta
Postorder:-
File ID -> 32       File name -> Delta
File ID -> 63       File name -> Gamma
File ID -> 58       File name -> Beta
File ID -> 80       File name -> Zeta
File ID -> 76       File name -> Alpha
1. Create Files
2. Traverse
3. Exit
Enter Choice: 3
```

Program – 12

Problem Statement

Consider a class having 100 students where, the details of each student like name, roll number and marks of 3 subjects is to be stored. Design and develop a program to construct a singly linked list to enter records of different students in list, display the list and calculate the percentage of each student. Also count the number of students passed (scored >40 in all the subjects).

Student Learning Outcomes

The students are able to develop an application using Linked List data structure.

Algorithm

Algorithm of creation of a Linked List

CREATE---In this algorithm a Linked List of nodes is created. The list is pointed by pointer first, the last node of the list points to NULL., indicating the end of the list. Each node is having two parts DATA and NEXT. Let us assume that a linked list of N number of nodes is to be created. The operator new will be used for the dynamic allocation of node. A variable I is being used as a counter to count the number of nodes in the created list.

STEPS:

1.first=new node;{create the 1st node of the list pointed by first};
2.Read(Data(first));
3.NEXT(First)=NULL;
4.Far a First; [point Far to the First]
5. For I=1 to N-1 repeat steps 6 to 10
6.X=new node;
7.Read(Data(X))
8.NEXT(X)=NULL;
9.NEXT(Far)=X; {connect the nodes}
10.Far=X;[shift the pointer to the last node of the list]
[end of For Loop]
11.END

TRAVERSING A LINKED LIST

Many a times, it is required to traverse whole of a linked list. For Example counting of nodes in a list, printing data of all the nodes etc.TRAVEL: In this algorithm a linked list, pointed by first, is traversed. The number of nodes in the list is also counted during the traverse. A pointer ptr is being used to visit the various nodes in the list. A variable count is used to keep track of the number of nodes visited during the traverse. The traverse stops when a NULL is encountered.

STEPS:

1.If First=NULL then {print "List empty" STOP};
2.count=0;
3.ptr=First; {point ptr to the 1st node}
4.While ptr<> NULL repeat Steps 5 to 6
5.count=count+1;
6.ptr=NEXT(ptr) [shift ptr to the next node]
7.print ('Number of nodes=', count)
8.END

In the above algorithm , step 6 is worth noting i.eptr=NEXT(ptr). This step means that the pointer Ptr should be shifted to the node which is being pointed by NEXT(ptr);

SEARCHING A LINKED LIST

Search is an operation in which an item is searched in a linked list. This operation is similar to traveling the list. An algorithm for search operation is given below:

SEARCH:

In this algorithm a linked list, pointed by first, is traversed. While traversing the data part of each

vivited node is compared with an item 'x'. If the item is found then the search stops otherwise the process continues til the end of the list(i.e NULL) is encountered. A pointer ptr is being used to visit the various nodes in the list.

STEPS:

1.If first=NULL then{
Print "List empty"; STOP;}
2.ptr=First; [point ptr to the 1st node]
3.while (ptr<>NULL) repeat steps 4 to 5
4.If (DATA (ptr)= 'X')
Then {print "item found";
STOP
}
5.ptr=NEXT (ptr); [shift ptr to the next node]
[end of while]
6.Print "item not found";
7.END

It may be noted in the above algorithm that if the item 'X' is found then the search stops.

INSERTION.....

In this algorithm a node X is inserted at the beginning of a linked list. The Linked List is being pointed by a pointer First at the beginning.

STEPS:

1.X=new node;
2.Read(DATA(X);
3.If (FIRST=NULL) then
{
First=X;
NEXT(X)=NULL;
}
Else
{
NEXT(X)=First;
First=X;
}
4.END

Program

```
package Experiment12;
import java.util.Scanner;
public class StudentApp
{
      public static void main(String[] args)
      {
            LinkedList list = new LinkedList();
            Scanner sc = new Scanner(System.in);
            System.out.println("Student Data entry");
            System.out.print("Enter the number of records: ");
            int n = sc.nextInt();
            Student data;
            for(int i = 0; i < n; i++)
            {
                  data = new Student(sc);
                  list.create(data);
            }
```

```
            System.out.println("The Student Details are:");
            list.printList();
            System.out.println("The number of Students passed is " +
list.traverse());
            sc.close();
      }
}
class Student
{
      String name;
      String roll;
      double marks1, marks2, marks3, percent;
      Student(Scanner sc)
      {
            System.out.println("Enter student details");
            System.out.print("Name: ");
            sc.nextLine();
            name = sc.nextLine();
            System.out.print("Roll Number: ");
            roll = sc.nextLine();
            System.out.print("Marks in subject-1: ");
            marks1 = sc.nextDouble();
            System.out.print("Marks in subject-2: ");
            marks2 = sc.nextDouble();
            System.out.print("Marks in subject-3: ");
            marks3 = sc.nextDouble();
            percent = (marks1 + marks2 + marks3) / 3;
      }
      boolean greater40()
      {
            return (marks1 > 40 && marks2 > 40 && marks3 > 40);
      }
      public String toString()
      {
            String str = String.format("Name :" + name);
            str += String.format("\nRoll Number:" + roll);
            str += String.format("\nSubject-1 marks:" + marks1);
            str += String.format("\nSubject-2 marks:" + marks2);
            str += String.format("\nSubject-3 marks:" + marks3);
            str += String.format("\nPercentage: %.2f", percent);
            return str;
      }
}
class LinkedList
{
    Node head;  // head of list
    LinkedList() { head = null;}
    /* Linked list Node*/
    class Node
    {
     Student data;
     Node next;

        Node(Student d)
        {
      data = d;
```

```
      next = null;
         }
         public String toString()
         {
      return data.toString();
         }
    }
    public void printList()
    {
        Node n = head;
        while (n != null)
        {
            System.out.println(n);
            n = n.next;
        }
    }
    void create(Student data)
    {
        Node node = new Node(data);
        /* If the Linked List is empty, then make the new node as head
*/
        if (head == null)
        {
            head = node;
            return;
        }
        /* traverse till the last node */
        Node last = head;
        while (last.next != null)
            last = last.next;
        /* Change the next of last node */
last.next = node;
        return;
    }
    int traverse()
    {
      Node iter = head;
      int count = 0;
      while(iter != null)
      {
            if(iter.data.greater40()) count++;
            iter = iter.next;
      }
      return count;
    }
}
```

Description of the Program

The class student create a student by reading a student name, roll number, marks of three subjects and calculate percentage.
The class linked list maintain the list of students in the form of linked records.
The create function makethe new node as head if Linked List is empty otherwise traverse till last

node.

Input Output

```
Student Data entry
Enter the number of records: 2
Enter student details
Name: Rajan
Roll Number: 1
Marks in subject-1: 60
Marks in subject-2: 30
Marks in subject-3: 45
Enter student details
Name: Ram
Roll Number: 2
Marks in subject-1: 60
Marks in subject-2: 70
Marks in subject-3: 75
The Student Details are:
Name :Rajan
Roll Number:1
Subject-1 marks:60.0
Subject-2 marks:30.0
Subject-3 marks:45.0
Percentage: 45.00
Name :Ram
Roll Number:2
Subject-1 marks:60.0
Subject-2 marks:70.0
Subject-3 marks:75.0
Percentage: 68.33
The number of Students passed is 1
```