Name of Student :- Pavan Ratan Gole

UID:- Na

Batch:- C4

Exp. No. 4

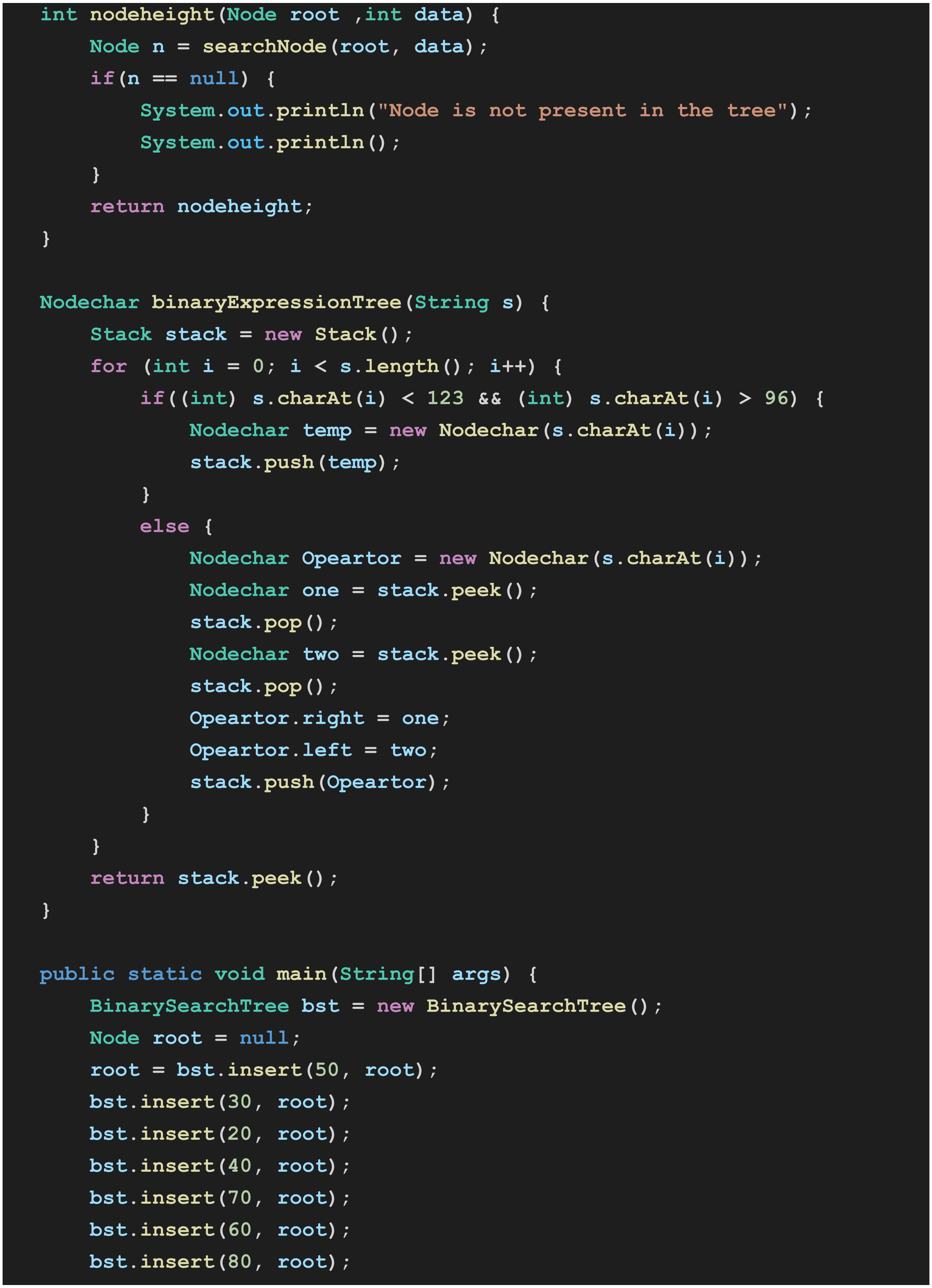
Aim:- Perform the following operations on a binary tree:

1-Creation of binary tree and display using any one traversal
2- counting no. of nodes in a binary tree
3- counting no leaf nodes in a binary tree
4- counting the height of a given node in a binary tree
5- create an Arithmetic expression tree from a given postfix expression
show the intermediate stages of output for each function

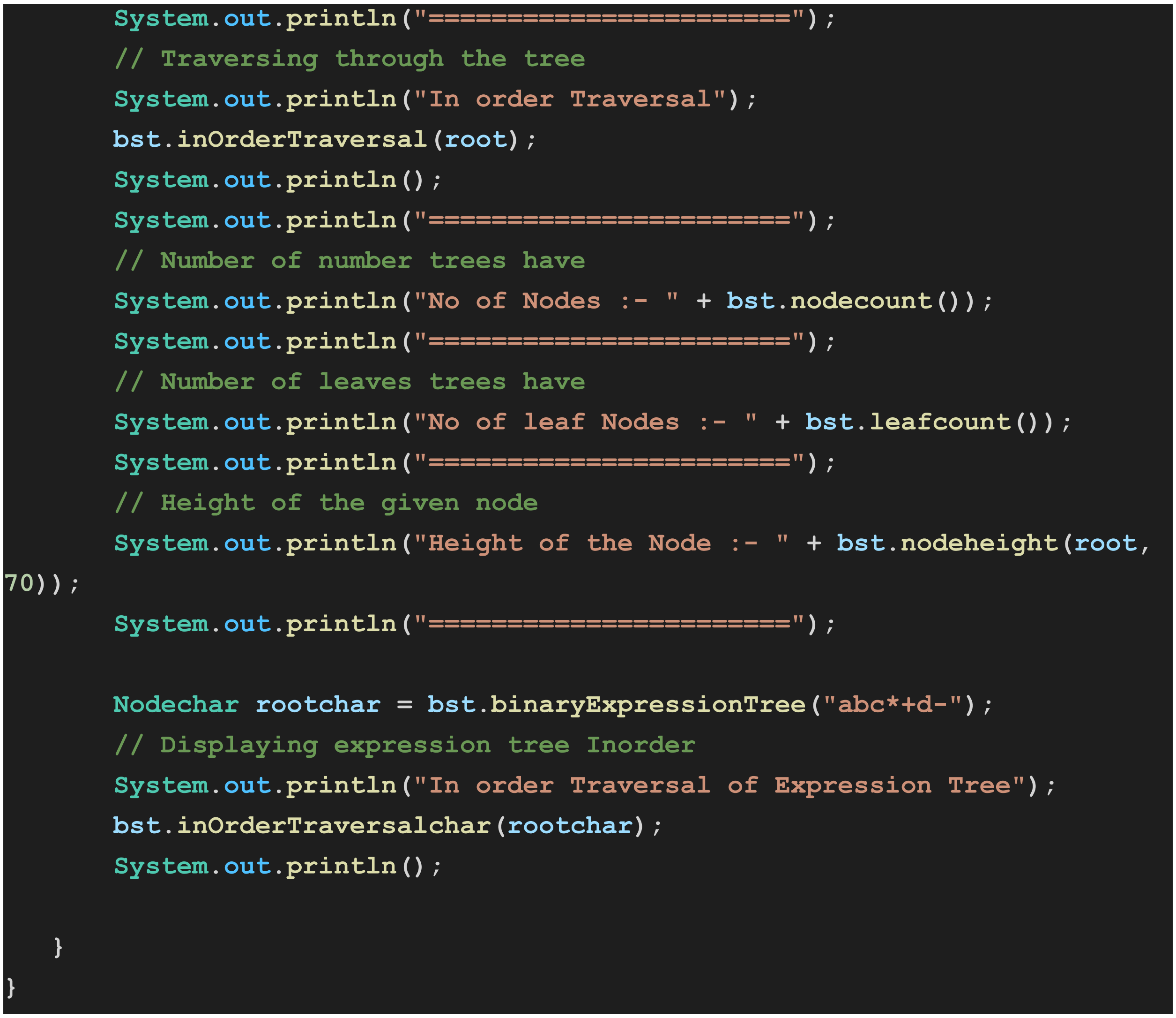
Program:-

```
/**
* BinarySearchTree
*/

class Stack {
   private  int top = -1;
   private  Nodechar[] data = new Nodechar[10];

   boolean isEmpty() {
       if(top != -1) {
           return false;
       }
       return true;
   }

   boolean isFull() {
       if(top != 5) {
           return false;
       }
       return true;
   }
```

```
    Nodechar push(Nodechar item) {
        if(isFull()) {
            System.out.println("Stack Overflowed");
            return null;
        }
        ++top;
        data[top] = item;
        return null;
    }

    int pop() {
        if(isEmpty()) {
            System.out.println("Stack Underflowed");
            return 0;
        }
        --top;
        return 0;
    }

    Nodechar peek() {
        if(isEmpty()) {
            System.out.println("Stack is Empty");
            return null;
        }
        return data[top];
    }
}

class Node {
    int data;
    Node left = null;
    Node right = null;

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

class Nodechar {
    char data;
    Nodechar left = null;
    Nodechar right = null;

    Nodechar(char data) {
        this.data = data;
    }
}

public class BinarySearchTree {
    Node root = null;
    private  int nodeheight = 0;
    private  int nodecount = 0;
    private  int leafcount = 0;

    Node insert(int data, Node root) {
        if (root == null) {
            root = new Node(data);
            return root;
        } else if (root.data < data) {
            root.left = insert(data, root.left);
        } else {
            root.right = insert(data, root.right);
        }
        return root;
    }

    void inOrderTraversal(Node root) {
        if(root != null) {
            inOrderTraversal(root.left);
            nodecount++;
            if(root.right == null && root.left == null) {
                leafcount++;
            }
            System.out.print(root.data + "  ");
            inOrderTraversal(root.right);
        }
    }

    void inOrderTraversalchar(Nodechar root) {
        if(root != null) {
            inOrderTraversalchar(root.left);
            nodecount++;
            if(root.right == null && root.left == null) {
                leafcount++;
            }
            System.out.print(root.data + " ");
            inOrderTraversalchar(root.right);
        }
    }



    int nodecount() {
        return nodecount;
    }

    int leafcount() {
        return leafcount;
    }

    Node searchNode(Node root, int data) {
        if(root == null) {
            return root;
        }
        if(root.data == data) {
            return root;
        }
        else if(root.data > data) {
            nodeheight++;
            return searchNode(root.right, data);
        }
        else {
            nodeheight++;
            return searchNode(root.left, data);
        }
    }
```



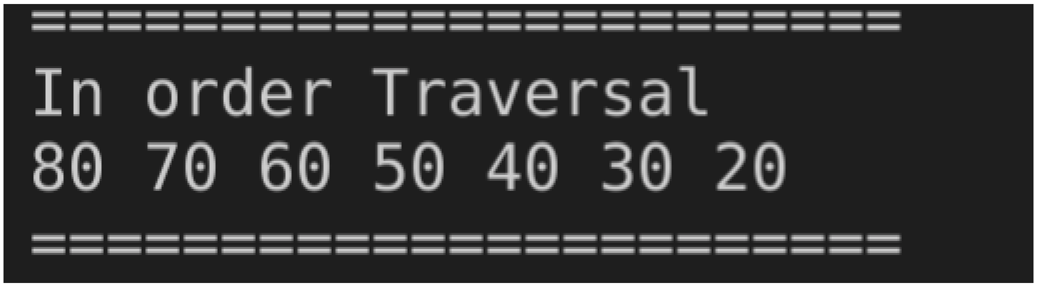
```java
int nodeheight(Node root ,int data) {
    Node n = searchNode(root, data);
    if(n == null) {
        System.out.println("Node is not present in the tree");
        System.out.println();
    }
    return nodeheight;
}

Nodechar binaryExpressionTree(String s) {
    Stack stack = new Stack();
    for (int i = 0; i < s.length(); i++) {
        if((int) s.charAt(i) < 123 && (int) s.charAt(i) > 96) {
            Nodechar temp = new Nodechar(s.charAt(i));
            stack.push(temp);
        }
        else {
            Nodechar Opeartor = new Nodechar(s.charAt(i));
            Nodechar one = stack.peek();
            stack.pop();
            Nodechar two = stack.peek();
            stack.pop();
            Opeartor.right = one;
            Opeartor.left = two;
            stack.push(Opeartor);
        }
    }
    return stack.peek();
}

public static void main(String[] args) {
    BinarySearchTree bst = new BinarySearchTree();
    Node root = null;
    root = bst.insert(50, root);
    bst.insert(30, root);
    bst.insert(20, root);
    bst.insert(40, root);
    bst.insert(70, root);
    bst.insert(60, root);
    bst.insert(80, root);
```



```
        System.out.println("=======================");
        // Traversing through the tree
        System.out.println("In order Traversal");
        bst.inOrderTraversal(root);
        System.out.println();
        System.out.println("=======================");
        // Number of number trees have
        System.out.println("No of Nodes :- " + bst.nodecount());
        System.out.println("=======================");
        // Number of leaves trees have
        System.out.println("No of leaf Nodes :- " + bst.leafcount());
        System.out.println("=======================");
        // Height of the given node
        System.out.println("Height of the Node :- " + bst.nodeheight(root,
70));
        System.out.println("=======================");

        Nodechar rootchar = bst.binaryExpressionTree("abc*+d-");
        // Displaying expression tree Inorder
        System.out.println("In order Traversal of Expression Tree");
        bst.inOrderTraversalchar(rootchar);
        System.out.println();

    }
}
```

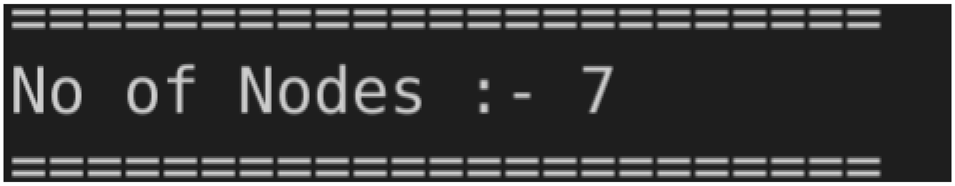
Output:-

1)



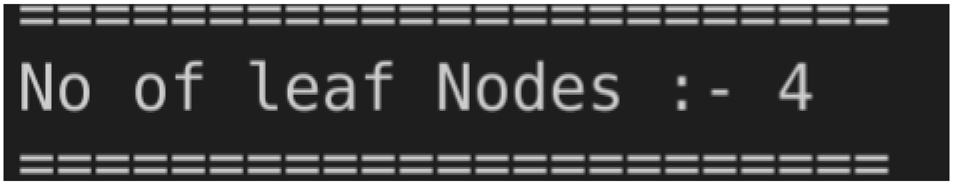
```
=======================
In order Traversal
80 70 60 50 40 30 20
=======================
```

2)



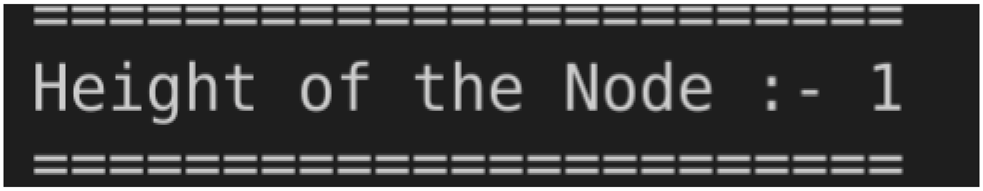
```
=======================
No of Nodes :- 7
=======================
```

3)



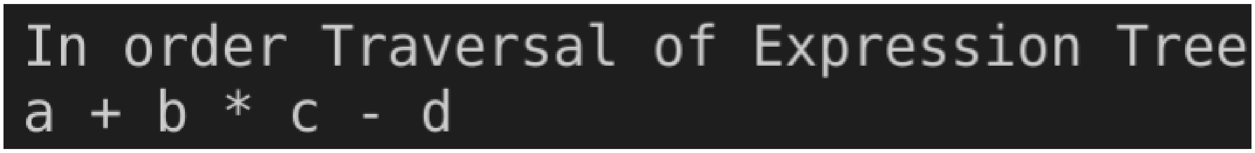
```
=======================
No of leaf Nodes :- 4
=======================
```

4)



```
=======================
Height of the Node :- 1
=======================
```

5)



```
In order Traversal of Expression Tree
a + b * c - d
```