Name – Yash Daga Roll Number – 20BCE7323

Assignment 3 2

1. Write a Java program to construct a binary tree from the given inorder and postorder traversals.

For example,

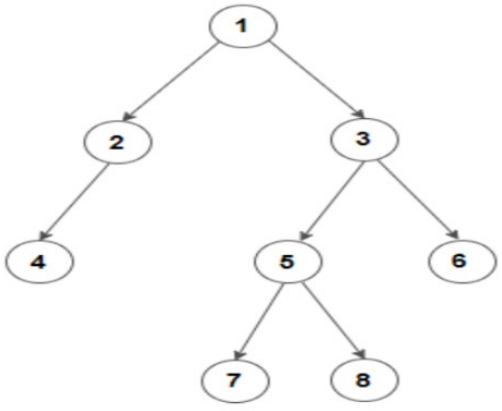
Input:

Inorder Traversal[] : { 4, 2, 1, 7, 5, 8, 3, 6 }

Postorder Traversal[] : { 4, 2, 7, 8, 5, 6, 3, 1 }

Output: print preorder traversal (Hint: you have to create tree first)

The idea is to start with the root node, which would be the last item in the postorder sequence, and find the boundary of its left and right subtree in the inorder sequence. To find the boundary, search for the index of the root node in the inorder sequence. All keys before the root node in the inorder sequence become part of the left subtree, and all keys after the root node become part of the right subtree. Repeat this recursively for all nodes in the tree and construct the tree in the process.

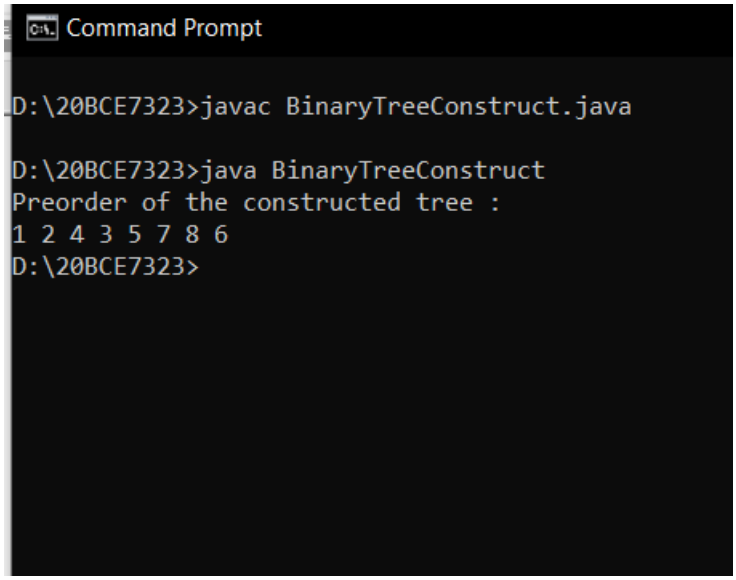


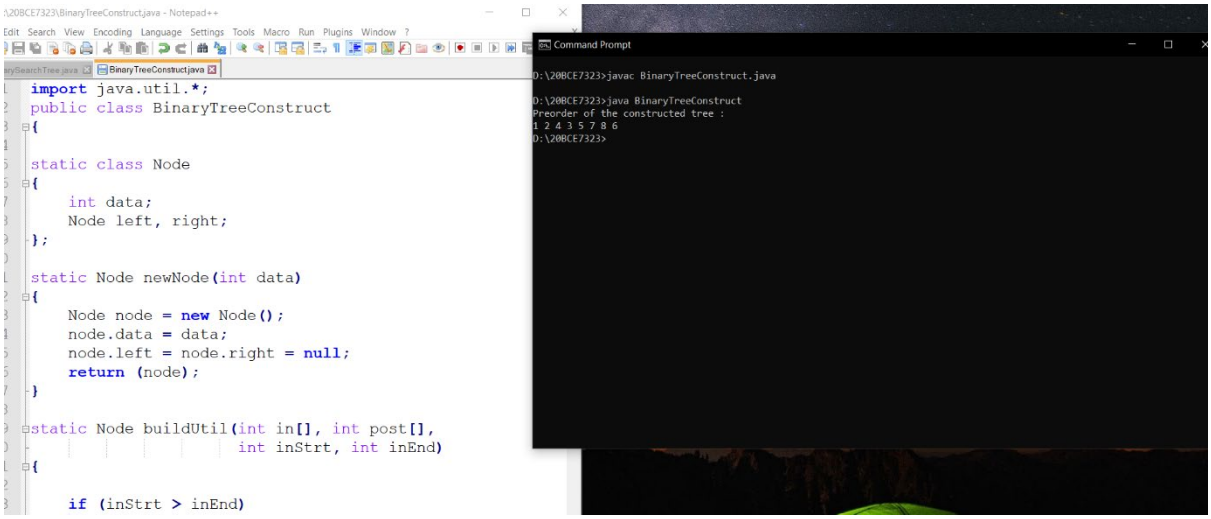
Code:

```
import java.util.*;
public class BinaryTreeConstruct
{

static class Node
{
   int data;
   Node left, right;
};

static Node newNode(int data)
{
   Node node = new Node();
   node.data = data;
   node.left = node.right = null;
   return (node);
}

static Node buildUtil(int in[], int post[],
                  int inStrt, int inEnd)
{

   if (inStrt > inEnd)
      return null;
```

```
    int curr = post[index];
    Node node = newNode(curr);
    (index)--;

    if (inStrt == inEnd)
        return node;

    int iIndex = mp.get(curr);

    node.right = buildUtil(in, post, iIndex + 1,
                           inEnd);
    node.left = buildUtil(in, post, inStrt,
                          iIndex - 1);
    return node;
}
static  HashMap<Integer,Integer> mp = new HashMap<Integer,Integer>();
static int index;

static Node buildTree(int in[], int post[], int len)
{

    for (int i = 0; i < len; i++)
        mp.put(in[i],  i);

     index = len - 1;
    return buildUtil(in, post, 0, len - 1 );
```

```
}

static void preOrder(Node node)
{
    if (node == null)
        return;
    System.out.printf("%d ", node.data);
    preOrder(node.left);
    preOrder(node.right);
}


public static void main(String[] args)
{
    int in[] = { 4, 2, 1, 7, 5, 8, 3, 6 };
    int post[] = { 4, 2, 7, 8, 5, 6, 3, 1 };
    int n = in.length;
    Node root = buildTree(in, post, n);
    System.out.print("Preorder of the constructed tree : \n");
    preOrder(root);
}
}
```


Command Prompt
D:\20BCE7323>javac BinaryTreeConstruct.java
D:\20BCE7323>java BinaryTreeConstruct
Preorder of the constructed tree :
1 2 4 3 5 7 8 6
D:\20BCE7323>


\20BCE7323\BinaryTreeConstruct.java - Notepad++
Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
BinaryTreeConstruct.java
import java.util.*;
public class BinaryTreeConstruct
{
static class Node
{
int data;
Node left, right;
};
static Node newNode(int data)
{
Node node = new Node();
node.data = data;
node.left = node.right = null;
return (node);
}
static Node buildUtil(int in[], int post[],
int inStrt, int inEnd)
{
if (inStrt > inEnd)
Command Prompt
D:\20BCE7323>javac BinaryTreeConstruct.java
D:\20BCE7323>java BinaryTreeConstruct
Preorder of the constructed tree :
1 2 4 3 5 7 8 6
D:\20BCE7323>

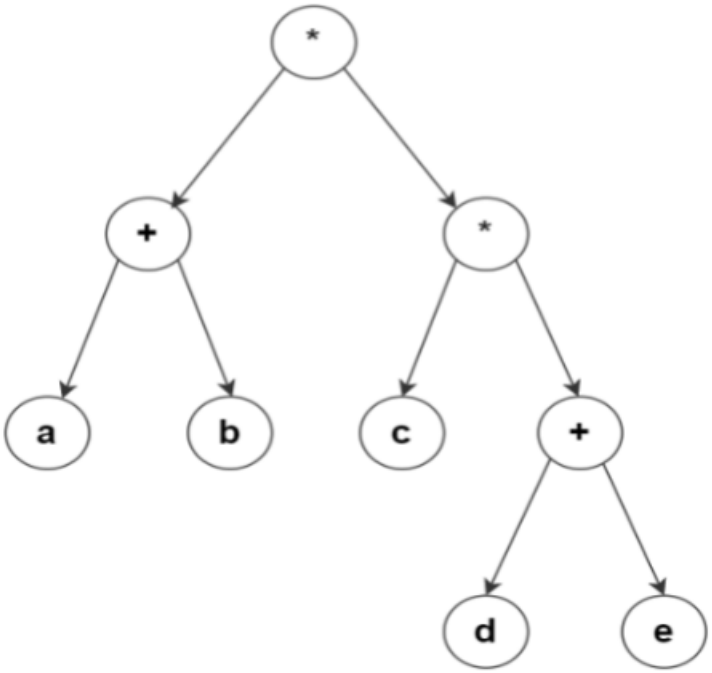
2. Write a Java Program to construct an expression tree from postfix expression.

For constructing an expression tree we can use a stack. We loop through input expression and do the following for every character.

1. If a character is an operand push that into the stack

2. If a character is an operator pop two values from the stack make them its child and push the current node again.

Input: ab+de+c **



Code:

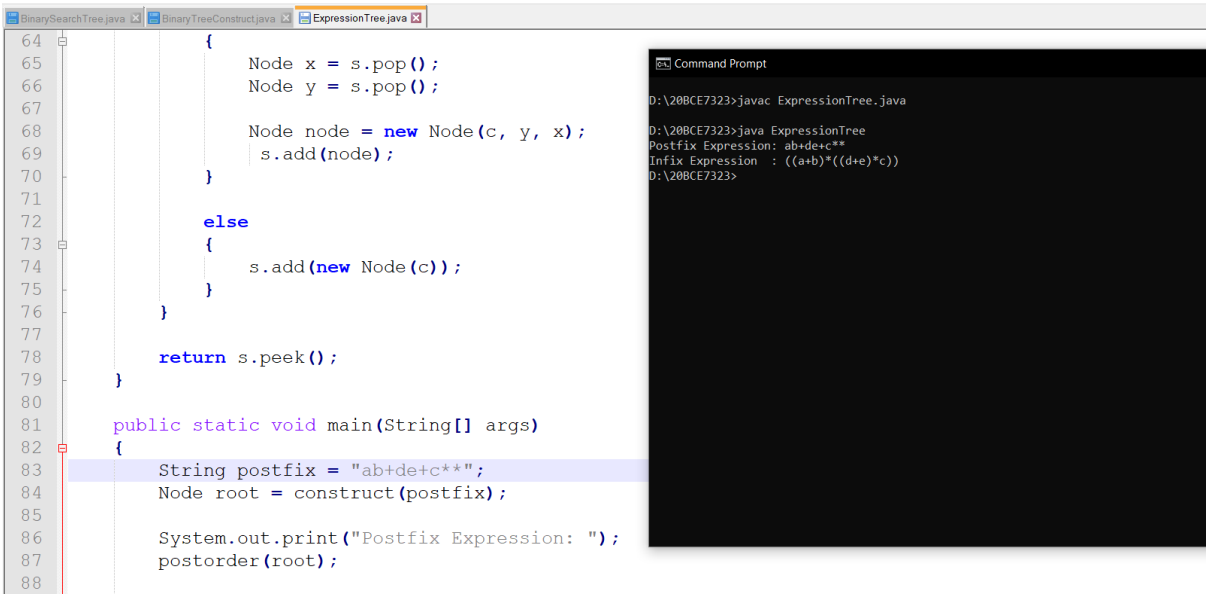
```
import java.util.*;
class Node
{
   char data;
   Node left, right;

   Node(char data)
   {
      this.data = data;
      this.left = this.right = null;
   }

   Node(char data, Node left, Node right)
   {
      this.data = data;
      this.left = left;
      this.right = right;
   }
}

class ExpressionTree
{
   public static boolean isOperator(char c)
   {
      return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
```

```
}

public static void postorder(Node root)
{
   if (root == null) {
      return;
   }
   postorder(root.left);
   postorder(root.right);
   System.out.print(root.data);
}

public static void inorder(Node root)
{
   if (root == null) {
      return;
   }

   if (isOperator(root.data)) {
      System.out.print("(");
   }

   inorder(root.left);
   System.out.print(root.data);
   inorder(root.right);
```

```
    if (isOperator(root.data)) {
        System.out.print(")");
    }
}

public static Node construct(String postfix)
{
    Stack<Node> s = new Stack<Node>();

    for (char c: postfix.toCharArray())
    {
        if (isOperator(c))
        {
            Node x = s.pop();
            Node y = s.pop();

            Node node = new Node(c, y, x);
             s.add(node);
        }

        else
        {
            s.add(new Node(c));
        }
    }
```

```
        return s.peek();
    }

    public static void main(String[] args)
    {
        String postfix = "ab+de+c**";
        Node root = construct(postfix);

        System.out.print("Postfix Expression: ");
        postorder(root);

        System.out.print("\nInfix Expression  : ");
        inorder(root);
    }
}
```


Command Prompt
D:\20BCE7323>javac ExpressionTree.java
D:\20BCE7323>java ExpressionTree
Postfix Expression: ab+de+c**
Infix Expression : ((a+b)*((d+e)*c))
D:\20BCE7323>


BinarySearchTree.java
BinaryTreeConstruct.java
ExpressionTree.java
{
Node x = s.pop();
Node y = s.pop();
Node node = new Node(c, y, x);
s.add(node);
}
else
{
s.add(new Node(c));
}
}
return s.peek();
}
public static void main(String[] args)
{
String postfix = "ab+de+c**";
Node root = construct(postfix);
System.out.print("Postfix Expression: ");
postorder(root);
Command Prompt
D:\20BCE7323>javac ExpressionTree.java
D:\20BCE7323>java ExpressionTree
Postfix Expression: ab+de+c**
Infix Expression : ((a+b)*((d+e)*c))
D:\20BCE7323>