import java.util.LinkedList;  
 import java.util.Queue;  
 import java.util.Scanner;  
 import java.util.Stack;  
  
public class assign2 {  
  
 static Node *root*;  
  
 static class Node {  
  
 char data;  
 Node left, right;  
  
 public Node(char data) {  
 this.data = data;  
 left = right = null;  
 }  
 }  
  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.*in*);  
 int continues = 1;  
 while (continues == 1) {  
 System.*out*.println("Menu:");  
 System.*out*.println("1: Enter expression");  
 System.*out*.println("2: Recursive Inorder");  
 System.*out*.println("3: Non-Recursive Inorder");  
 System.*out*.println("4: Recursive Preorder");  
 System.*out*.println("5: Non-Recursive Preorder");  
 System.*out*.println("6: Recursive Postorder");  
 System.*out*.println("7: Non-Recursive Postorder");  
 System.*out*.println("8: BFS");  
 System.*out*.println("<Other Keys>: Exit");  
 System.*out*.print("Enter choice: ");  
 int choice;  
 try {  
 choice = in.nextInt();  
 } catch (Exception e) {  
 choice = 9;  
 }  
 System.*out*.println("\n");  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter expression: ");  
 String str = in.next();  
 *convert*(str);  
 break;  
 case 2:  
 System.*out*.println("Recursive Inorder: ");  
 *recurinorder*(*root*);  
 break;  
 case 3:  
 System.*out*.println("Non-Recursive Inorder: ");  
 *nonrecurinorder*(*root*);  
 break;  
 case 4:  
 System.*out*.println("Recursive Preorder:");  
 *recurpreorder*(*root*);  
 break;  
 case 5:  
 System.*out*.println("Non-recursive Preorder:");  
 *nonrecurpreorder*(*root*);  
 break;  
 case 6:  
 System.*out*.println("Recursive Postorder:");  
 *recurpostorder*(*root*);  
 break;  
 case 7:  
 System.*out*.println("Non-Recursive Postorder:");  
 *nonrecurpostorder*(*root*);  
 break;  
 case 8:  
 System.*out*.println("BFS:");  
 *bfs*(*root*);  
 break;  
 default:  
 continues = 0;  
 break;  
 }  
 System.*out*.println("\n\n");  
 }  
 in.close();  
 }  
  
 public static boolean isOperator(char c) {  
 return (c == '+' || c == '-' || c == '\*' || c == '/');  
 }  
  
 public static void convert(String postfix) {  
 Stack<Node> stack = new Stack<>();  
 for (char ch : postfix.toCharArray()) {  
 if (!*isOperator*(ch)) {  
 Node temp = new Node(ch);  
 stack.push(temp);  
 } else {  
 Node op1, op2, temp;  
 temp = new Node(ch);  
 op1 = stack.pop();  
 op2 = stack.pop();  
 temp.left = op2;  
 temp.right = op1;  
 stack.push(temp);  
 }  
 }  
 *root* = stack.pop();  
 }  
  
 public static void recurinorder(Node root) {  
 if (root == null) {  
 return;  
 } else {  
 Node temp = root;  
 *recurinorder*(temp.left);  
 System.*out*.print(temp.data + " ");  
 *recurinorder*(temp.right);  
 }  
 }  
  
 public static void recurpreorder(Node root) {  
 if (root == null) {  
 return;  
 } else {  
 Node temp = root;  
 System.*out*.print(temp.data + " ");  
 *recurpreorder*(temp.left);  
 *recurpreorder*(temp.right);  
 }  
 }  
  
 public static void recurpostorder(Node root) {  
 if (root == null) {  
 return;  
 } else {  
 Node temp = root;  
 *recurpostorder*(temp.left);  
 *recurpostorder*(temp.right);  
 System.*out*.print(temp.data + " ");  
 }  
 }  
  
 public static void nonrecurinorder(Node root) {  
 Stack<Node> stack = new Stack<>();  
 Node current = root;  
  
 while (current != null || !stack.isEmpty()) {  
 while (current != null) {  
 stack.push(current);  
 current = current.left;  
 }  
 current = stack.pop();  
 System.*out*.print(current.data + " ");  
 current = current.right;  
 }  
 }  
  
 public static void nonrecurpreorder(Node root) {  
 Stack<Node> stack = new Stack<>();  
 stack.push(root);  
 while (!stack.isEmpty()) {  
 Node current = stack.pop();  
 System.*out*.print(current.data + " ");  
 if (current.right != null) {  
 stack.push(current.right);  
 }  
 if (current.left != null) {  
 stack.push(current.left);  
 }  
 }  
 }  
  
 public static void nonrecurpostorder(Node root) {  
 Stack<Node> stack1 = new Stack<>();  
 Stack<Node> stack2 = new Stack<>();  
 stack1.push(root);  
 while (!stack1.isEmpty()) {  
 Node current = stack1.pop();  
 stack2.push(current);  
 if (current.left != null) {  
 stack1.push(current.left);  
 }  
 if (current.right != null) {  
 stack1.push(current.right);  
 }  
 }  
 while (!stack2.isEmpty()) {  
 System.*out*.print(stack2.pop().data + " ");  
 }  
 }  
  
 public static void bfs(Node root) {  
 if (root == null) {  
 System.*out*.println("The tree is empty.");  
 return;  
 }  
  
 Queue<Node> queue = new LinkedList<>();  
 queue.add(root);  
 while (!queue.isEmpty()) {  
 Node current = queue.poll();  
 System.*out*.print(current.data + " ");  
 if (current.left != null) {  
 queue.add(current.left);  
 }  
 if (current.right != null) {  
 queue.add(current.right);  
 }  
 }  
 }  
}