PROBLEMS on STACK

*(contains 3 problems)*

**1. Infix to postfix conversion Algorithm:**

**Transform the algebraic expression with brackets into RPN form (Reverse Polish Notation). Two-argument operators: +, -, \*, /, ^ (priority from the lowest to the highest), brackets ( ). Operands: only letters: a,b,...,z. Assume that there is only one RPN form (no expressions like a\*b\*c).**

Input:

3 //test cases

(a+(b\*c))

((a+b)\*(z+x))

((a+t)\*((b+(a+c))^(c+d)))

Output:

abc\*+

ab+zx+\*

at+bac++cd+^\*

**Algorithm:**

**1.**  Scan the infix expression from left to right.  
**2.** If the scanned character is an operand, output it.  
**3.** Else,  
 …..**3.1**  If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a ‘(‘ ), push it.

…..**3.2**  Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)  
**4.**  If the scanned character is an ‘(‘, push it to the stack.  
**5.**  If the scanned character is an ‘)’, pop the stack and and output it until a ‘(‘ is encountered, and discard both the parenthesis.  
**6.** Repeat steps 2-6 until infix expression is scanned.  
**7.** Print the output  
**8.** Pop and output from the stack until it is not empty.

**CODE:**

|  |
| --- |
| import java.util.Stack;  class Test  {  // A utility function to return precedence of a given operator  // Higher returned value means higher precedence  static int Prec(char ch)  {  switch (ch)  {  case '+':  case '-':  return 1;    case '\*':  case '/':  return 2;    case '^':  return 3;  }  return -1;  }    // The main method that converts given infix expression  // to postfix expression.  static String infixToPostfix(String exp)  {  // initializing empty String for result  String result = new String("");    // initializing empty stack  Stack<Character> stack = new Stack<>();    for (int i = 0; i<exp.length(); ++i)  {  char c = exp.charAt(i);    // If the scanned character is an operand, add it to output.  if (Character.isLetterOrDigit(c))  result += c;    // If the scanned character is an '(', push it to the stack.  else if (c == '(')  stack.push(c);    // If the scanned character is an ')', pop and output from the stack  // until an '(' is encountered.  else if (c == ')')  {  while (!stack.isEmpty() && stack.peek() != '(')  result += stack.pop();    if (!stack.isEmpty() && stack.peek() != '(')  return "Invalid Expression"; // invalid expression  else  stack.pop();  }  else // an operator is encountered  {  while (!stack.isEmpty() && Prec(c) <= Prec(stack.peek()))  result += stack.pop();  stack.push(c);  }    }    // pop all the operators from the stack  while (!stack.isEmpty())  result += stack.pop();    return result;  }    // Driver method  public static void main(String[] args)  {  String exp = "a+b\*(c^d-e)^(f+g\*h)-i";  System.out.println(infixToPostfix(exp));  }  } |

## 2. STPAR - Street Parade

**For sure, the love mobiles will roll again on this summer's street parade. Each year, the organisers decide on a fixed order for the decorated trucks. Experience taught them to keep free a side street to be able to bring the trucks into order.   
  
The side street is so narrow that no two cars can pass each other. Thus, the love mobile that enters the side street last must necessarily leave the side street first. Because the trucks and the ravers move up closely, a truck cannot drive back and re-enter the side street or the approach street.   
  
You are given the order in which the love mobiles arrive. Write a program that decides if the love mobiles can be brought into the order that the organisers want them to be.**

### Input

**There are several test cases. The first line of each test case contains a single number n, the number of love mobiles. The second line contains the numbers 1 to n in an arbitrary order. All the numbers are separated by single spaces. These numbers indicate the order in which the trucks arrive in the approach street. No more than 1000 love mobiles participate in the street parade. Input ends with number 0.**

### Output

**For each test case your program has to output a line containing a single word "yes" if the love mobiles can be re-ordered with the help of the side street, and a single word "no" in the opposite case.**

### Example

**Sample input:**

5

5 1 2 4 3

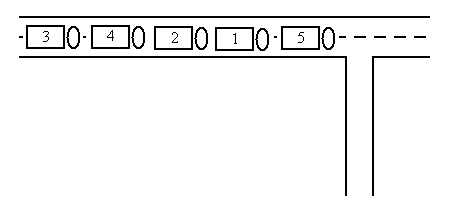
0

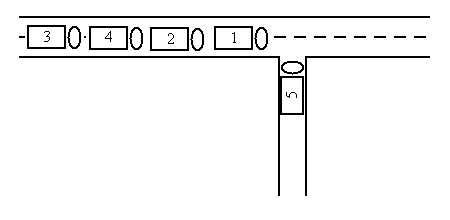
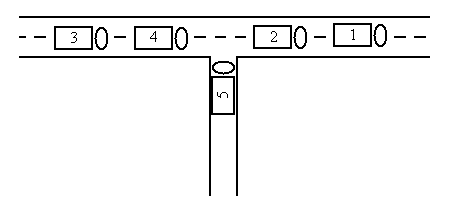
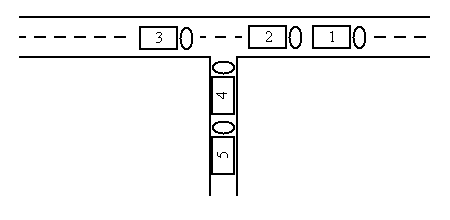
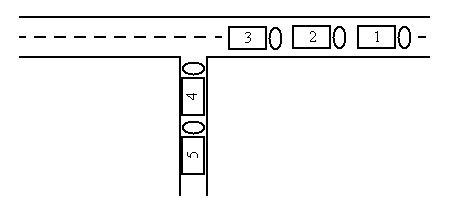
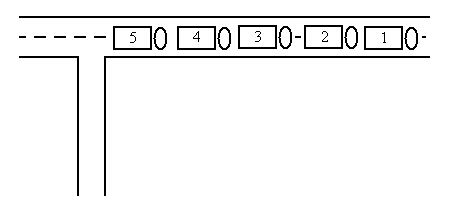
**Sample output:**

yes

### Illustration

**The sample input reflects the following situation:**



**The five trucks can be re-ordered in the following way:**    

**Code:**

|  |
| --- |
| stack<int> lane;int need = 1;bool state = true;for (int i = 0; i < order.size(); i++) {while (!lane.empty() && lane.top() == need) {need++;lane.pop();}if (order[i] == need) {need++;} else if (!lane.empty() && lane.top() < order[i]) {state = false;break;} else {lane.push(order[i]);}}if (state) cout << "yes" << "\n";else cout << "no" << "\n"; |

**3. Finding Molecular mass of Molecules which consist of only C,H,O,digit (0-9), (,);**

**example : CH(CHO2)3, COOH**

**Code:**

|  |  |
| --- | --- |
|  | import java.util.Stack;  import java.util.Scanner; |
|  |  |
|  | public class MolMass { |
|  |  |
|  | static Stack<Integer> st = new Stack<Integer>(); |
|  |  |
|  |  |
|  | public static int addMass(char ch) { |
|  |  |
|  | if(ch=='H') |
|  | return 1; |
|  |  |
|  | else if(ch=='C') |
|  | return 12; |
|  |  |
|  | else if(ch=='O') |
|  | return 16; |
|  |  |
|  | else |
|  | return 0; |
|  |  |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  |  |
|  |  |
|  | Scanner scanner = new Scanner(System.in); |
|  |  |
|  | System.out.println("MASS IN"); |
|  |  |
|  | String s = scanner.nextLine(); |
|  |  |
|  | int v; |
|  |  |
|  | for(int i=0;i<s.length();i++) { |
|  |  |
|  | if(Character.isDigit(s.charAt(i))) { |
|  |  |
|  | v = st.peek(); |
|  | st.pop(); |
|  |  |
|  | st.push((v\*(s.charAt(i)-'0'))); |
|  |  |
|  | } |
|  |  |
|  | else if(s.charAt(i)=='(') |
|  | st.push(-1); |
|  |  |
|  | else if(s.charAt(i)==')'){ |
|  |  |
|  | v=0; |
|  | while(!st.empty() && st.peek()!=-1) { |
|  |  |
|  | v+=st.peek(); |
|  | st.pop(); |
|  |  |
|  | } |
|  |  |
|  | st.pop(); |
|  | st.push(v); |
|  |  |
|  | } |
|  |  |
|  | else if(Character.isAlphabetic(s.charAt(i))) { |
|  |  |
|  | st.push(addMass(s.charAt(i))); |
|  |  |
|  | } |
|  | } |
|  |  |
|  | v=0; |
|  |  |
|  | while(!st.empty()) { |
|  |  |
|  | v+=st.peek(); |
|  | st.pop(); |
|  |  |
|  | } |
|  |  |
|  | System.out.println("MASS OUT"); |
|  | System.out.println(v); |
|  |  |
|  | scanner.close(); |
|  |  |
|  | } |
|  | } |
|  |