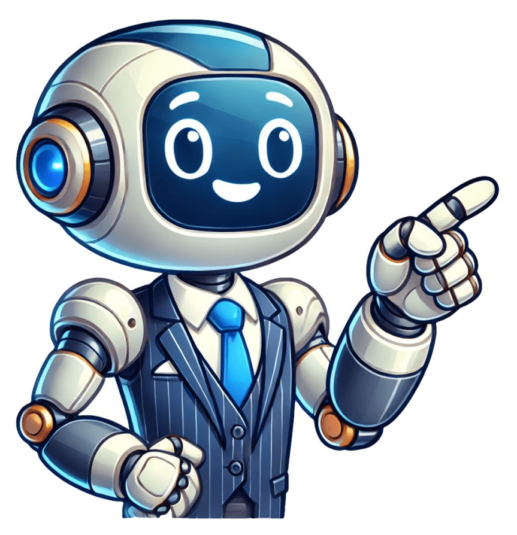
Continue



In this post, we will see how to convert infix to postfix expression in java. This is an important problem related to Stack Data Structure Let us quickly look into Infix and Postfix expression: Infix expressions are the expression where operators are written in between every pair of operands. It is the usual way to write an expression. For Ex: An expression like A + B is Infix. Postfix expressions are the expressions where operands precede operators. Here operators are written after operands. The Expression AB+ is Postfix and is the Postfix representation of the above shown A+B. The evaluation order is from left to right. The expressions we (humans) write are Infix Expression, but for the machines to understand these expression they need to be converted. The compiler scans an expression from left to right. If an infix expression such as A*(B+C) is provided, the compiler will scan the expression to evaluate B+C then scan again to multiply the result with A. This results in multiple scanning. So, it is recommended to use Postfix notations which for the given expression is: ABC+* . A compiler can then easily evaluate the result in one go. Now how is the conversion done? We will look into the algorithm for this along with an example. Read also: Check for balanced parentheses in expression java We will use two Stacks : One for Operators (Character), One for Operands (String). The Operand stack will contain the resultant Postfix expression after traversing string. The Infix expression will be given as a String input. We will take decisions when we encounter Parentheses, Operators and Operands. So the steps are: Step 1: We will iterate throughout the String length. For each character, there are three cases to consider : 1. If the Current character is a Operand. 2. If the character is a Open or Close Parentheses. 3. If the character is an Operator. Step 2: If the Character at each iteration is a Operand. We simply push it into the operand or Postfix stack. Step 3: If the character encountered is : '(' , i.e. Opening Parentheses, we push it into Operator Stack. Step 4: Now, if we encounter ')' i.e. Closing Parenthesis, we are going to pop the elements out of Operator Stack until we get the opening '('. For each operator we pop its two operands and process them. Step 5: The process for each operator is : We pop two elements from Postfix or operand Stack we concatenate them in reverse order with its operator and add the result again to Postfix stack for future evaluation until we get the total Postfix expression. Step 6: Now if we get an Operator as the current character, we check whether the precedence of current operator is lower than the operator present at top of the stack. If the condition is true, we pop the operator present at the top and process its operands following Step 5. Then we push the current operator into the stack. Step 7: At last after traversing the whole string if still we are left with any operators we pop them and continue Step 5 until the Operator Stack is empty. At last, the Postfix stack will have only one element which will be our resultant Postfix Expression. Note: The Precedence of * and / are equal and higher than + and - operation. Let us understand this Algorithm with an example: Consider the Infix : A * (B-C) / D + EWe take 2 Stacks: Operator and Postfix, and do the following steps: 1. We start iterating through each character (ignoring the spaces). At i=0, char = A an Operand. We push it into Postfix stack. At i=1, char = * an operator, we push it into operator stack. The stacks look like: 2. We continue at i=2, char = (, an opening parenthesis we push it into operator stack, at i=3, char =B we push it into Postfix stack . At i=4, char = - we push it into Operator stack, at i=5, char = C we push into the Postfix stack. The stacks now look like: 3. We continue iterating, at i=6; char =) i.e. a Closing Parenthesis, so now we follow Step 4 and Step 5 of the algorithm . We pop elements from operator stack until we get the Opening Parenthesis. Then, we pop two elements from Postfix stack (C and B), concatenate them in reverse order with the operator in reverse order (BC-) and add the result into the Postfix stack again. Then we pop). The stacks now look like : Now BC- is an operand in Postfix Stack. 4. Then at i=7, char = /, an operator, we need to add it to our Operator stack but before doing it we check whether there is an operator in the stack with precedence higher or equal to current operator . There is * operator in stack with precedence equal to / ,so we process it following Step 5. Then push / operator. So we pop * and add it with operands (A and BC-) using Step 5. The stacks now look like : 5. We continue, at i=8, char =D, we push it into Postfix stack. At i=9, we have + operator, we again check the stack for any operator with greater precedence. / has higher precedence, so we process it first like shown above using Step 5 then push + into the Stack. The stacks now are: 6. Finally at i=10, we have char = E, so we push it in the postfix stack. 7. Now this is an important point, After traversing the string if there are still any operators left in the stack, we process them repeating Step 5 until we get the final expression. In this case, after iterating the string the + is still remaining in Operator Stack. After processing the last operator the stacks now look like : This is Postfix Representation for the given Infix Expression. Note: The Postfix stack at last will hold the Postfix expression and will be the only element. Now let us look at the implementation code in JAVA: import java.util.*;public class InfixtoPostfix{public static int precedence(char ch){ if(ch=='+' || ch=='-') return 1; else if(ch=='*' || ch=='/') return 2; // * and / divide have higher precedence. return 0;}public static String convertToPostfix(String exp){ Stack operators = new Stack(); Stack postFix = new Stack(); for(int i=0;i='a' && ch='A' && ch0 && operators.peek()!='(' && precedence(ch) 0) { char op = operators.pop(); String first = postFix.pop(); String second = postFix.pop(); String new_postFix = second+first+op; postFix.push(new_postFix); } return postFix.pop(); // return resultant Postfix.}public static void main(String args[]){ // We pass Uppercase Infix String infixExpression = "A*(B-C)/D+E"; System.out.println("The Infix Expression is: "+infixExpression); String result = convertToPostfix(infixExpression); System.out.println("The Postfix of the given Infix Expression is: "+result); System.out.println(); //We also check for Lowercase Infix infixExpression = "a*(b-c+d)/e"; System.out.println("The Infix Expression is: "+infixExpression); result = convertToPostfix(infixExpression); System.out.println("The Postfix of the given Infix Expression is: "+result);}} public class InfixtoPostfixpublic static int precedence(char ch) else if(ch=='*' || ch=='/') return 2; // * and / divide have higher precedence.public static String convertToPostfix(String exp) Stack operators = new Stack(); Stack postFix = new Stack(); for(int i=0;i='a' && ch='A' && ch0 && operators.peek()!='(' && precedence(ch) 0)char op = operators.pop();String first = postFix.pop();String second = postFix.pop();String new_postFix = second+first+op;postFix.push(new_postFix);return postFix.pop(); // return resultant Postfix.public static void main(String args[])// We pass Uppercase InfixString infixExpression = "A*(B-C)/D+E";System.out.println("The Infix Expression is: "+infixExpression);String result = convertToPostfix(infixExpression);System.out.println("The Postfix of the given Infix Expression is: "+result);//We also check for Lowercase InfixinfixExpression = "a*(b-c+d)/e";System.out.println("The Infix Expression is: "+infixExpression);result = convertToPostfix(infixExpression);System.out.println("The Postfix of the given Infix Expression is: "+result); Output: The Infix Expression is: A*(B-C)/D+EThe Postfix of the given Infix Expression is: ABC-*D/E+The Infix Expression is: a*(b-c+d)/eThe Postfix of the given Infix Expression is: abc-d+*e/ The Infix Expression is: A*(B-C)/D+EThe Postfix of the given Infix Expression is: ABC-*D/E+The Infix Expression is: a*(b-c+d)/eThe Postfix of the given Infix Expression is: abc-d+*e/ This the output for the above discussed example and a sample. Let us look into the the time complexity of our approach. Time Complexity: We do a single traversal of the string to convert it into Postfix expression so the time complexity is O(n) , n is the length of Infix Expression. Thats all about how to convert infix to postfix in java. You can try out this with various examples and execute this code for a clear idea. Let us know if this post was helpful. Feedbacks are monitored on daily basis. Please do provide feedback as that\'s the only way to improve. The infix and postfix expressions can have the following operators: '+', '-', '%','*', '/' and alphabets from a to z. The precedence of the operators (+, -) is lesser than the precedence of operators (*, /, %). Parenthesis has the highest precedence and the expression inside it must be converted first. In this section, we will learn how to convert infix expression to postfix expression and postfix to infix expression through a Java program.For performing the conversion, we use Stack data structure. The stack is used to store the operators and parenthesis to enforce the precedence Start parsing the expression from left to right. Before moving ahead in this section, ensure that you are friendly with the stack and its operations. Let's have a look at infix and postfix expressions.Infix ExpressionInfix expressions are those expressions in which the operator is written in-between the two or more operands. Usually, we use infix expression. For example, consider the following expression.a+ba/2+c*d-e*(f*g)a*(b+c)/dPostfix ExpressionPostfix expressions are those expressions in which the operator is written after their operands. For example, consider the following expression.Infix Vs. Postfix ExpressionInfix ExpressionPostfix ExpressionA*B/CAB*C/(A/(B-C+D))*(E-A)*CABC-D+/EA-*C*A/B-C+D*E-A*CAB/C-DE*AC*-Infix to Postfix Conversion ExampleConvert the (X - Y / (Z + U) * V) infix expression into postfix expression.S.N.InputOperand StackPostfix Expression1((-2X(X3-(-X4Y(-XY5/(- /XY6((- / (XY7Z(- / (XYZ8+(- / (+XYZ9U(- / (+XYZU10)(- /XYZU+11*(- *XYZU+/12V(- *XYZU+/V13)-XYZU+/V*-AlgorithmScan the infix notation from left to right one character at a time.If the next symbol scanned as an operand, append it to the postfix string.If the next symbol scanned as an operator, the:Pop and append to the postfix string every operator on the stack that:Is above the most recently scanned left parenthesis, andHas precedence higher than or is a right-associative operator of equal precedence to that of the new operator symbol.Push the new operator onto the stackIf a left parenthesis is scanned, push it into the stack.If a right parenthesis is scanned, all operators down to the most recently scanned left parenthesis must be popped and appended to the postfix string. Furthermore, the pair of parentheses must be discarded.When the infix string is fully scanned, the stack may still contain some operators. All the remaining operators should be popped and appended to the postfix string.Let's implement the above algorithm in a Java program.Java Program to Convert Infix Expression into Postfix ExpressionInfixToPostfixConversion.javaimport java.io.*;class Stack{char a[]=new char[100];int top=-1;void push(char c){try{a[++top]= c;}catch(StringIndexOutOfBoundsException e){System.out.println("Stack full, no room to push, size=100");System.exit(0);}}char pop(){return a[top--];}boolean isEmpty(){return (top==-1)?true:false;}char peek(){return a[top];}}public class InfixToPostfix{static Stack operators = new Stack();public static void main(String argv[]) throws IOException{String infix;//create an input stream objectBufferedReader keyboard = new BufferedReader (new InputStreamReader(System.in));//get input from userSystem.out.print("Enter the infix expression you want to convert: ");infix = keyboard.readLine();//output as postfixSystem.out.println("Postfix expression for the given infix expression is:" + toPostfix(infix));}private static String toPostfix(String infix)//converts an infix expression to postfix{char symbol;String postfix = "";for(int i=0;i= 'a' && c = 'A' && c = '0' && c