# Infix to Postfix Java

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**](https://www.javatpoint.com/java-tutorial) **program.**

For performing the conversion, we use [**Stack** data structure](https://www.javatpoint.com/data-structure-stack). 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 Expression

Infix 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.

1. a+b
2. a/2+c\*d-e\*(f\*g)
3. a\*(b+c)/d

## Postfix Expression

Postfix expressions are those expressions in which the operator is written after their operands. For example, consider the following expression.

1. ab+
2. abc/de+\*+f-
3. ab+cd-\*

## Infix vs Postfix Expression

| **Infix Expression** | **Postfix Expression** |
| --- | --- |
| A\*B/C | AB\*C/ |
| (A/(B-C+D))\*(E-A)\*C | ABC-D+/EA-\*C\* |
| A/B-C+D\*E-A\*C | AB/C-DE\*AC\*- |

## Infix to Postfix Conversion Example

Convert the **(X - Y / (Z + U) \* V)** infix expression into postfix expression.

| **S.N.** | **Input** | **Operand Stack** | **Postfix Expression** |
| --- | --- | --- | --- |
| 1 | ( | ( | - |
| 2 | X | ( | X |
| 3 | - | ( - | X |
| 4 | Y | ( - | XY |
| 5 | / | ( - / | XY |
| 6 | ( | ( - / ( | XY |
| 7 | Z | ( - / ( | XYZ |
| 8 | + | ( - / ( + | XYZ |
| 9 | U | ( - / ( + | XYZU |
| 10 | ) | ( - / | XYZU+ |
| 11 | \* | ( - \* | XYZU+/ |
| 12 | V | ( - \* | XYZU+/V |
| 13 | ) | - | XYZU+/V\*- |

### Algorithm

1. Scan the infix notation from left to right one character at a time.
2. If the next symbol scanned as an operand, append it to the postfix string.
3. If the next symbol scanned as an operator, the:
   1. Pop and append to the postfix string every operator on the stack that:
      1. Is above the most recently scanned left parenthesis, and
      2. Has precedence higher than or is a right-associative operator of equal precedence to that of the new operator symbol.
   2. Push the new operator onto the stack
4. If a left parenthesis is scanned, push it into the stack.
5. 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.
6. 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 Expression

**InfixToPostfixConversion.java**

1. **import** java.io.\*;
2. **class** Stack
3. {
4. **char** a[]=**new** **char**[100];
5. **int** top=-1;
6. **void** push(**char** c)
7. {
8. **try**
9. {
10. a[++top]= c;
11. }
12. **catch**(StringIndexOutOfBoundsException e)
13. {
14. System.out.println("Stack full, no room to push, size=100");
15. System.exit(0);
16. }
17. }
18. **char** pop()
19. {
20. **return** a[top--];
21. }
22. **boolean** isEmpty()
23. {
24. **return** (top==-1)?**true**:**false**;
25. }
26. **char** peek()
27. {
28. **return** a[top];
29. }
30. }
31. **public** **class** InfixToPostfix
32. {
33. **static** Stack operators = **new** Stack();
34. **public** **static** **void** main(String argv[]) **throws** IOException
35. {
36. String infix;
37. //create an input stream object
38. BufferedReader keyboard = **new** BufferedReader (**new** InputStreamReader(System.in));
39. //get input from user
40. System.out.print("\nEnter the infix expression you want to convert: ");
41. infix = keyboard.readLine();
42. //output as postfix
43. System.out.println("Postfix expression for the given infix expression is:" + toPostfix(infix));
44. }
45. **private** **static** String toPostfix(String infix)
46. //converts an infix expression to postfix
47. {
48. **char** symbol;
49. String postfix = "";
50. **for**(**int** i=0;i<infix.length();++i)
51. //while there is input to be read
52. {
53. symbol = infix.charAt(i);
54. //if it's an operand, add it to the string
55. **if** (Character.isLetter(symbol))
56. postfix = postfix + symbol;
57. **else** **if** (symbol=='(')
58. //push (
59. {
60. operators.push(symbol);
61. }
62. **else** **if** (symbol==')')
63. //push everything back to (
64. {
65. **while** (operators.peek() != '(')
66. {
67. postfix = postfix + operators.pop();
68. }
69. operators.pop(); //remove '('
70. }
71. **else**
72. //print operators occurring before it that have greater precedence
73. {
74. **while** (!operators.isEmpty() && !(operators.peek()=='(') && prec(symbol) <= prec(operators.peek()))
75. postfix = postfix + operators.pop();
76. operators.push(symbol);
77. }
78. }
79. **while** (!operators.isEmpty())
80. postfix = postfix + operators.pop();
81. **return** postfix;
82. }
83. **static** **int** prec(**char** x)
84. {
85. **if** (x == '+' || x == '-')
86. **return** 1;
87. **if** (x == '\*' || x == '/' || x == '%')
88. **return** 2;
89. **return** 0;
90. }
91. }

**Output:**

**Infix to Postfix Java**

## Postfix to Infix Conversion Example

Convert the **AB + CD - /** postfix expression into infix expression.

| **S.N.** | **Input** | **Operand Stack** | **Infix Expression** |
| --- | --- | --- | --- |
| 1 | A | A | - |
| 2 | B | A, B | - |
| 3 | + | A + B | A+ B |
| 4 | C | A + B, C | - |
| 5 | D | A + B, C, D | - |
| 6 | - | A + B, C - D | C - D |
| 7 | / | A + B / C - D | A + B / C - D |

### Algorithm

1. Read the postfix expression from left to right one character at a time.
2. If it is operand push into operand stack.
3. If it is an operator, then:
   1. Pop two operand from the stack.
   2. From infix expression and push into the operand stack
4. If the expression is not ended go to the first step.
5. Pop operand stack and display.
6. Exit