what are Infix and postfix

Infix and postfix are two notations used to represent mathematical expressions.

1. Infix Notation:

- Infix notation is the most commonly used notation for writing mathematical expressions, where operators are placed between operands.

- In infix notation, operators have fixed positions relative to their operands.

- Examples of infix expressions:

- 2 + 3

- (4 - 1) \* 5

- 2 + 3 \* 4

2. Postfix Notation (also known as Reverse Polish Notation or RPN):

- Postfix notation is an alternative notation for writing mathematical expressions, where operators are placed after their operands.

- In postfix notation, operators do not have fixed positions relative to their operands. Instead, the order of operations is determined by the position of the operators.

- Examples of postfix expressions:

- 2 3 +

- 4 1 - 5 \*

- 2 3 4 \* +

The main difference between infix and postfix notations is the placement of operators relative to operands. Infix notation requires the use of parentheses to specify the order of operations, while postfix notation eliminates the need for parentheses by using the position of operators to determine the order of operations.

Converting an infix expression to postfix notation (infix to postfix conversion) can be useful for evaluating mathematical expressions or for simplifying expression parsing in computer programming.

Infix to postfix conversion Rulles

The rules for converting an infix expression to a postfix expression are as follows:

1. Start with an empty postfix expression and an empty stack.

2. Scan the infix expression from left to right.

3. If the scanned character is an operand (letter or digit), append it to the postfix expression.

4. If the scanned character is an opening parenthesis '(', push it to the stack.

5. If the scanned character is a closing parenthesis ')', pop operators from the stack and append them to the postfix expression until an opening parenthesis is encountered. Remove the opening parenthesis from the stack.

6. If the scanned character is an operator (+, -, \*, /, etc.), compare its precedence with the operator at the top of the stack.

- If the stack is empty or contains an opening parenthesis on top, push the current operator to the stack.

- If the precedence of the current operator is higher than the operator at the top of the stack, push the current operator to the stack.

- If the precedence of the current operator is lower than or equal to the operator at the top of the stack, pop operators from the stack and append them to the postfix expression until a lower precedence operator is encountered, then push the current operator to the stack.

7. After scanning the infix expression, pop any remaining operators from the stack and append them to the postfix expression.

8. The resulting postfix expression is the desired output.

Note: It's important to consider operator precedence when comparing operators. Operators with higher precedence should be evaluated first. In the case of equal precedence, the associativity of the operators (left-to-right or right-to-left) should be considered.

For example, let's convert the infix expression "a + b \* c - (d / e + f) \* g" to postfix using these rules:

|  |
| --- |
| Infix Expression: a + b \* c - (d / e + f) \* g  Scanned Character: Action:  a Append to postfix expression  + Push to stack  b Append to postfix expression  \* Push to stack  c Append to postfix expression  - Pop \* from stack and append to postfix expression, Push - to stack  ( Push to stack  d Append to postfix expression  / Push to stack  e Append to postfix expression  + Push to stack  f Append to postfix expression  ) Pop + and append to postfix expression, Pop / and append to postfix expression, Pop ( from stack  \* Push to stack  g Append to postfix expression  Pop \* and append to postfix expression |

Final Postfix Expression: "a b c \* + d e / f + g \* -"

The resulting postfix expression is "a b c \* + d e / f + g \* -".