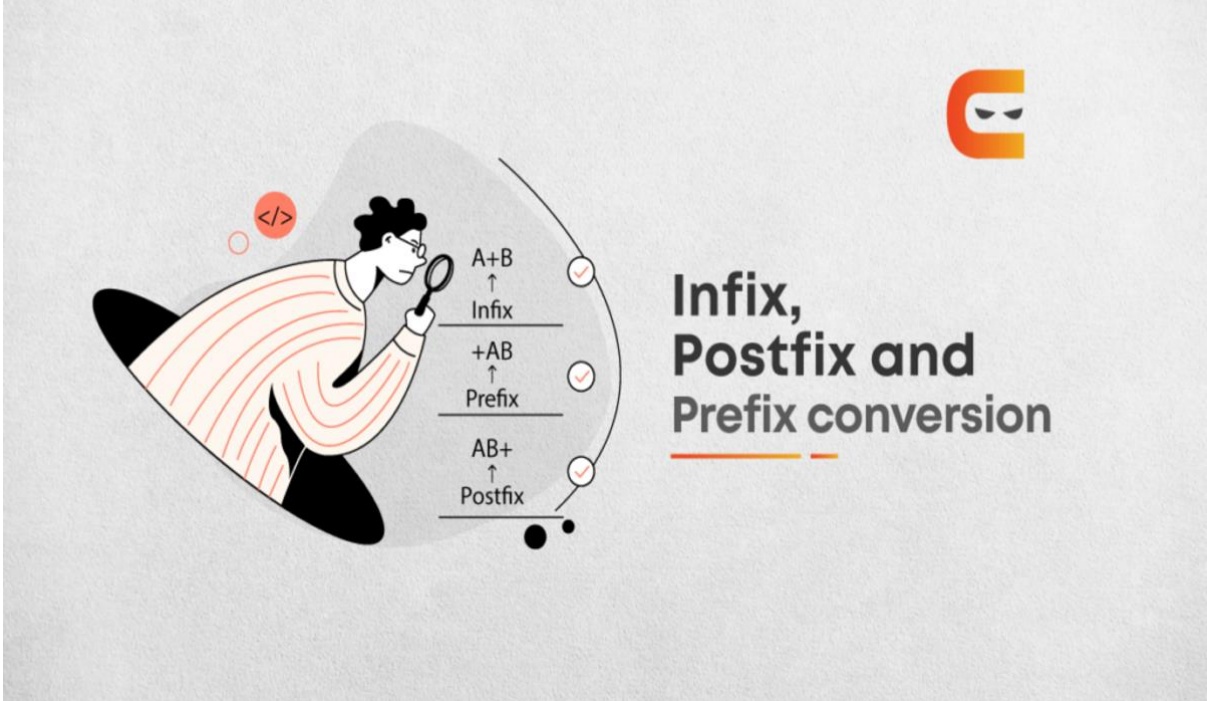
Infix, Postfix, and Prefix Conversion



Definition of Infix, Postfix, and Prefix

Infix: The typical mathematical form of expression that we encounter generally is known as infix notation. In infix form, an operator is written in between two operands.

For example:
An expression in the form of **A * (B + C) / D** is in infix form. This expression can be simply decoded as: *"Add B and C, then multiply the result by A, and then divide it by D for the final answer."*

Prefix: In prefix expression, **an operator is written before its operands.**

For example, The, above expression can be written in the prefix form as **/ * A + B C D**.

This type of expression cannot be simply decoded as infix expressions.

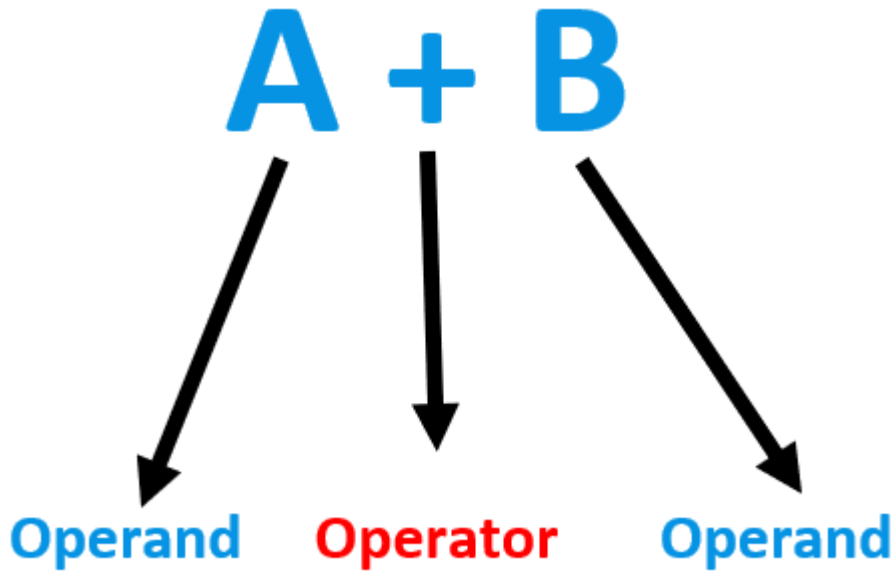
Postfix: In postfix expression, **an operator is written after its operands.**

For example, The, above expression can be written in the postfix form as **A B C + * D /**.

This type of expression cannot be simply decoded as infix expressions.

Refer to the table below to understand these expressions with some examples:

Infix	Prefix	Postfix
A+B	+AB	AB+



Conversion of Infix to Postfix

One of the applications of postfix notation is to build a calculator or evaluate expressions in a programming language. In addition, we can evaluate postfix expressions efficiently using a stack data structure.

Therefore, postfix notation is effective for implementing algorithms such as postfix notation evaluation and expression parsing.

The process of converting an infix expression to a postfix expression involves the following steps:

1. First, **we create an empty stack and an empty postfix expression**

2. Next, **we iterate through the infix expression from left to right and append operands to the postfix expression**

3. If an **operator is encountered, we pop operators from the stack and append them to the postfix expression until an operator with lower or equal precedence is found**

4. The **current operator is then pushed onto the stack**

5. **If a left parenthesis is encountered, we push it onto the stack**

6. **If a right parenthesis is encountered, we pop operators from the stack and append them to the postfix expression until a left parenthesis is found**

7. Finally, **we pop any remaining operators from the stack and append them to the postfix expression**