Introduction to Chapter 3: Stack - II

→ In this chapter, we learn about different ways of writing mathematical expressions and how computers process them.

→ When we write expressions like **A + B**, the **operator (+)** is placed between the two **operands (A and B)**. This is called **infix notation**. However, when we have more complex expressions, such as **A + B * C**, it becomes confusing whether to **add (A + B) first** or **multiply (B * C) first**.

To solve this problem, we follow **operator precedence rules**:

- **Multiplication (*) and division (/)** have higher priority than **addition (+) and subtraction (-)**.
- If two operators have the same priority, we follow **left-to-right associativity** (except for exponentiation).

→ Using these rules, the correct interpretation of **A + B * C** is:

1. **Multiply** B and C first → (B * C)
2. **Then add** A to the result → A + (B * C)

→ To make the order of operations clear, we can use **parentheses**:

- (A + B) * C → **Addition first, then multiplication**
- A + (B * C) → **Multiplication first, then addition (correct order)**

→ To avoid confusion, computers use **prefix and postfix notations**, where the order of operations is **automatically determined** by the position of operators.

→ The main focus of this chapter is learning how to **convert** between infix, prefix, and postfix notations using **stacks**, ensuring expressions are processed correctly without ambiguity.

- Here is a simple table showing examples of **Infix, Prefix, and Postfix** notations:

Infix Expression	Prefix (Polish Notation)	Postfix (Reverse Polish Notation)
A + B	+ A B	A B +
A + B * C	+ A * B C	A B C * +
(A + B) * C	* + A B C	A B + C *
A - B + C	+ - A B C	A B - C +
A * (B + C)	* A + B C	A B C + *
(A + B) * (C - D)	* + A B - C D	A B + C D - *

Application of Stack: Infix, Prefix, and Postfix

→ **For writing complex mathematical expressions, we generally prefer parentheses to make them more readable.** In computers, expressions with various parentheses add unnecessary work while solving. So, to minimize the computational work, various notations have been made. **Prefix and Postfix are one of them.**

→ **In this chapter, we'll be learning about infix, postfix, and prefix conversion in detail.** You'll find questions on infix, postfix, and prefix conversion in the frequently asked interview questions of almost every top tech-based company.

Definition of Operators and Operands

- **Operator:** An operator is a symbol that tells the compiler or interpreter to perform specific mathematical, logical, or relational operations.
  Example: +, -, *, /, %

- **Operand:** An operand is the value (or variable) on which the operator acts. **Example:** In the expression A + B, **A** and **B** are operands, while + is the operator.

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. This notation is also known as "Polish notation".**

→ 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. This notation is also known as "Reverse Polish notation".**

→ 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.

How to Check if an Expression is Valid or Not?

→ When working with mathematical expressions, it's important to check if the expression is valid before converting it into another form (like prefix or postfix).

Rule for a Valid Expression

A mathematical expression is **valid** if:

Rank : Total number of operands - Total number of operators = 1

Understanding Operands and Operators

- **Operands** are the values or variables in an expression (e.g., a, b, c).
- **Operators** are the symbols used for operations (e.g., +, -, *, /).

Examples of Valid and Invalid Expressions

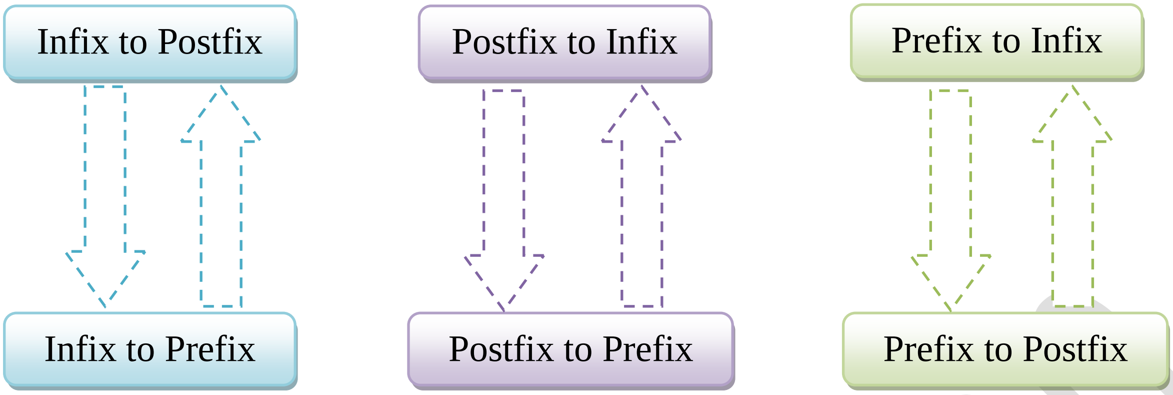
Infix Expression	Prefix Expression	Rank (Operands - Operators)	Valid or Invalid
a + * b	ab*+	0	**Invalid**
a - b * c	abc*-	1	**Valid**
ab + c	abc+	2	**Invalid**
(a + b) * (c - d)	ab+cd-*	1	**Valid**
a + b / d -	abd/+ -	0	**Invalid**

How to Check Validity?

1. **Count the operands and operators** in the given expression.
2. **Subtract the number of operators from the number of operands.**
3. **If the result is 1, the expression is valid**; otherwise, it's invalid.

This simple rule helps in verifying whether an expression is correctly written!

Expression Evaluation & Conversion


Infix to Postfix

Infix to Prefix

Postfix to Infix

Postfix to Prefix

Prefix to Infix

Prefix to Postfix

Infix to Postfix Conversion

- → The order of the numbers or operands remains unchanged. But the order of the operators gets changed in the conversion.
- → Stacks are used for converting an infix expression to a postfix expression. The stack that we use in the algorithm will change the order of operators from infix to Postfix.
- → Postfix expressions do not contain parentheses.

we can divide Infix expression into two parts:

1) Infix expression with parenthesis

2) Infix expression without parenthesis

a. Infix to Postfix Conversion Without Parenthesis

→ **Steps to convert Infix to postfix expression without parenthesis:**

Step 1: Initialize stack contents to the special symbol #.

Step 2: Scan the leftmost symbol in the infix expression and denote it as the current input symbol.

Step 3: Repeat through Step 6 while the current input symbol is not #.

Step 4: Remove and output all stack symbols whose precedence values are greater than or equal to the precedence of the current input symbol.

Step 5: Push the current input symbol onto the stack.

Step 6: Scan the leftmost symbol in the infix expression and let it be the current input symbol.

Use of Tabular Method with Following Rules

- **Scan the Given Infix from Left To Right Always.**
- **If Given Infix is without () then initialize stack with # and end with #**
- **If Given Infix is with () then initialize stack (and end with)**
- **Operators + , - , * , \ , % they cannot sit with each other in Stack.**
- **Operators ^, $, ↑ they can sit with each other in Stack.**
- **Postfix answer will not have any ()**

Example 1: Convert the infix expression x * y + z to postfix.

Character Scanned	Content of Stack	Postfix	Rank
#	#		
x	#x		
*	#*	x	1
y	#*y	x	1
+	#+	xy*	1
z	#+z	xy*	1
#	#	xy*z+	1

Example 2: Convert the infix expression p - q * r to postfix.

Character Scanned	Content of Stack	Postfix	Rank
#	#		
p	#p		
-	#-	p	1
q	#-q	p	1
*	#-*	pq	2
r	#-*r	pq	2
#	#	pqr*-	1

Example 3: Convert the infix expression a + b / c * d to postfix.

Character Scanned	Content of Stack	Postfix	Rank
#	#		
a	#a		
+	#+	a	1
b	#+b	a	1
/	#+/	ab	2
c	#+/c	ab	2
*	#+*	abc/	2
d	#+*d	abc/	2
#	#	abc/d*+	1

Example 4:Convert the Following String from Infix to Postfix:

Expression: a + b * c – d * e – f + g

Character scanned	Content of stack	Postfix	Rank
#	#		
a	#a		
+	#+	a	1
b	#+b	a	1
*	#+*	ab	2
c	#+*c	ab	2
-	#-	abc*+	1
d	#-d	abc*+	1
*	#-*	abc*+d	2
e	#-*e	abc*+d	2
-	#-	abc*+de*-	1
f	#-f	abc*+de*-	1
+	#+	abc*+de*-f-	1
g	#+g	abc*+de*-f-	1
#	#	abc*+de*-f-g+	1

Example 5 – Convert the Following String from Infix to Postfix:

Expression: e – x * c / b – e + f

Character scanned	Content of stack	Postfix	Rank
#	#		
e	#e		
-	#-	e	1
x	#-x	e	1
*	#-*	ex	2
c	#-*c	ex	2
/	#-/	exc*	2
b	#-/b	exc*	2
-	#-	exc*b/-	1
e	#-e	exc*b/-	1
+	#+	exc*b/-e-	1
f	#+f	exc*b/-e-	1
#	#	**exc*b/-e-f+**	1

Example 6 – Convert the Following String from Infix to Postfix: a^b^c/e

Character Scanned	Content of Stack	Postfix	Rank
#	#		
a	#a		
^	#^	a	1
b	#^b	a	1
^	#^^	ab	2
c	#^^c	ab	2
/	#/	abc^^	1
e	#/e	abc^^	1
#	#	**abc^^e/**	1

Example 7 – Convert following infix expressions to the postfix expressions.

A/B$C+D*E/F-G+H

Character Scanned	Content of Stack	Postfix	Rank
#	#		
A	#A		
/	#/	A	1
B	#/B	A	1

$	#/$	A B	2
C	#/$C	A B	2
+	#+	A B C $ /	1
D	#+D	A B C $ /	1
*	#+*	A B C $ / D	2
E	#+*E	A B C $ / D	2
/	#+/	A B C $ / D E *	2
F	#+/F	A B C $ / D E *	2
-	#-	A B C $ / D E * F / +	1
G	#-G	A B C $ / D E * F / +	1
+	#+	A B C $ / D E * F / + G -	1
H	#+H	A B C $ / D E * F / + G -	1
#	#	**A B C $ / D E * F / + G - H +**	1

Example 8 – Conversion of the given infix expression K + L - M * N + O ^ P * W / U / V + T + Q to postfix using the table method.

Character Scanned	Content of Stack	Postfix	Rank
#	#		
K	#K		
+	#+#	K	1
L	#+L	K	1
-	#-	K L +	1
M	#-M	K L +	1
*	#-*	K L + M	2
N	#-*N	K L + M	2
+	#+#	K L + M N *	1
O	#+O	K L + M N *	1
^	#+^	K L + M N * O	2
P	#+^P	K L + M N * O	2
*	#+*	K L + M N * O P ^	2
W	#+*W	K L + M N * O P ^	2
/	#+/	K L + M N * O P ^ W *	2
U	#+/U	K L + M N * O P ^ W *	2
/	#+/	K L + M N * O P ^ W * U /	2
V	#+/V	K L + M N * O P ^ W * U /	2
+	#+	K L + M N * O P ^ W * U / V /	1
T	#+T	K L + M N * O P ^ W * U / V /	1
+	#+	K L + M N * O P ^ W * U / V / T +	1
Q	#+Q	K L + M N * O P ^ W * U / V / T +	1
#	#	**K L + M N * O P ^ W * U / V / T + Q +**	1

❖ Infix to Postfix Conversion With Parenthesis

Table of Precedence:

Symbol	Input Precedence(Higher the int, higher the precedence)	Stack Precedence	Rank
a, b, c…	7	8	1
↑ (Exponent)	6	5	-1
*, /	3	4	-1
+, -	1	2	-1
((Left Paren..)	9	0	-
) (Right Paren..)	0	-	-

Example 9 – Convert the Following String from Infix to Postfix:

(a + b ↑ c ↑ d) * (e + f / g)

Character Scanned	Content of Stack	Postfix	Rank
(	(		
(	((		
a	((a		
+	((+	a	1
b	((+b	a	1
↑	((+↑	ab	2
c	((+↑c	ab	2
↑	((+↑↑	abc	3
d	((+↑↑d	abc	3
)	(	abcd↑↑++	1
*	*	abcd↑↑++	1
(	*(	abcd↑↑++	1
e	*(e	abcd↑↑++	1
+	*(+	abcd↑↑++e	2
f	*(+f	abcd↑↑++e	2
/	*(+/	abcd↑↑++ef	3
g	*(+/g	abcd↑↑++ef	3
)	*	abcd↑↑++efg/+	2
)		**abcd↑↑++efg/+***	**1**

Example 10 – Convert the Following String from Infix to Postfix:

(a + b) / c * d

Character Scanned	Content of Stack	Postfix	Rank
(	(		
(	((		
a	((a		
+	((+	a	1
b	((+b	a	1
)	(	ab+	1
/	/	ab+	1
c	/c	ab+	1
*	*	ab+/	1
d	*d	ab+/	1
)		**ab+c/d***	**1**

Example 11 – Convert the Following String from Infix to Postfix:

(a + b) / (c - d) * e

Character Scanned	Content of Stack	Postfix	Rank
(	(		
(	((		
a	((a		
+	((+	a	1
b	((+b	a	1
)	(	ab+	1
/	/	ab+	1
(	/(	ab+	1
c	/(c	ab+	1
-	/(-	ab+c	2
d	/(-d	ab+c	2
)	/	ab+cd-/	1
*	*	ab+cd-/	1
e	*e	ab+cd-/	1
)		**ab+cd-/e***	**1**

Example 12 – Convert the Following String from Infix to Postfix: a – (b+c) / (d/e)

Character scanned	Content of stack	Postfix	Rank
	(		
a	(a	a	1
-	(-	a	1
(	(- (	a	1
b	(- (b	a	1
+	(- (+	ab	2
c	(- (+c	ab	2
)	(-	abc+	2
/	(- /	abc+	2
(	(- / (	abc+	2
d	(- / (d	abc+	2
/	(- / (/	abc+d	3
e	(- / (/ e	abc+d	3
)	(- /	abc+de/	1
)		**abc+de/-**	1

Example 13 – Convert 2 * 3 / (2 - 1) + 5 * 3 into Postfix form

Character Scanned	Content of Stack	Postfix	Rank
	(		
2	(2		
*	(*	2	1
3	(*3	2	1
/	(/	23*	1
(	(/(	23*	1
2	(/(2	23*	1
-	(/(-	23*2	2
1	(/(- 1	23*2	2
)	(/	23*21-	2
+	(+	23*21-/	1
5	(+5	23*21-/	1
*	(+*	23*21-/5	2
3	(+*3	23*21-/5	5
)		23*21-/53*+	1

Example 14 – Convert the infix notation (A – B + C × (D × E – F)) / G + H × K into postfix notation?

Character Scanned	Content of Stack	Postfix Expression	Rank
	(		
(	((		
A	((A		
-	((-	A	1
B	((- B	A	1
+	((+	A B -	1
C	((+ C	A B -	1
×	((+ ×	A B - C	1
(	((+ × (	A B - C	1
D	((+ × (D	A B - C	1
×	((+ × (×	A B - C D	1
E	((+ × (× E	A B - C D	1
-	((+ × (-	A B - C D E ×	2
F	((+ × (- F	A B - C D E ×	2
)	((+ ×	A B - C D E × F -	2
)	(	A B - C D E × F - ×+	2
/	(/	A B - C D E × F - ×+	2
G	(/ G	A B - C D E × F - ×+	2
+	(+	A B - C D E × F - × +G /	1
H	(+ H	A B - C D E × F - ×+ G /	1
×	(+ ×	A B - C D E × F - × +G / H	1
K	(+ × K	A B - C D E × F - ×+ G / H	1
)		A B - C D E × F - ×+ G / H K × +	1

Infix to Prefix Conversion

- → **While we use infix expressions in our day-to-day lives, computers have trouble understanding this format because they need to keep in mind rules of operator precedence and also brackets.**
- → **Prefix and Postfix expressions are easier for a computer to understand and evaluate.**
- → **Given two operands a and b and an operator +**
- → **When the operator is placed before the operands, i.e., +ab, the expression is in prefix notation.**
- → **Given any infix expression, we can obtain the equivalent prefix format.**

Steps to convert Infix to Prefix expression:

Step 1: Reverse the infix expression, i.e., A + B * C will become C * B + A. Note: While reversing, each (will become) and each) becomes (.

Step 2: Obtain the "nearly" postfix expression of the modified expression, i.e., CB* A+.

Step 3: Reverse the postfix expression. Hence, in our example, the prefix is +A * BC.

Rules:-

- **Reverse the Given Infix Expression. (Scan the Given Infix from Right To Left Always)**
- **Find the Nearly Postfix of that reverse expression.**
- **If Given Infix is without () then initialize stack with # and end with #**
- **If Given Infix is with () then initialize stack) and end with (**
- **+ , - , * , \ , % they can sit with each other in Stack.**
- **^, $, ↑ they cannot sit with each other in Stack.**
- **Reverse the answer of Nearly Postfix [That is our Final Answer Postfix answer will not have any ()]**

Example 15 – Convert the Following String from Infix to Prefix: a + b * e + f / g

Step 1: Reverse the given string

Infix Expression: a + b * e + f / g
Reversed Expression: g / f + e * b + a

Step 2: Find Nearly Postfix

Following the table format:

Character Scanned	Content of Stack	Postfix	Rank
#	#		
g	#g		
/	#/	g	1
f	#/f	g	1
+	#+	gf/	1
e	#+e	gf/	1
*	#+*	gf/e	2
b	#+*b	gf/e	2
+	#++	gf/eb*	2
a	#++a	gf/eb*	2
#	#	gf/eb* a++	1

Step 3: Reverse the Postfix Expression

Postfix obtained: gf/eb* a++
Reverse this to get the **Prefix Expression**: ++a * eb / gf

Thus, the **Prefix Expression** for a + b * e + f / g is: **++a*eb/gf**

Example 16 - Convert the Following String from Infix to Prefix: (a + b) * (c - d) / e^f

Step 1: Reverse the given String

Infix Expression: (a + b) * (c - d) / e^f
Reversed Expression: f^e / d - c * b + a

Step 2: Find Nearly Postfix

Character Scanned	Content of Stack	Postfix	Rank
#	#		
f	#f		

^	#^	f	1
e	#^e	f	1
/	#/	fe^	1
d	#/d	fe^	1
-	#/-	fe^d	1
c	#/-c	fe^d	1
*	#/*	fe^dc-	2
b	#/*b	fe^dc-	2
+	#/+	fe^dc-b	2
a	#/+a	fe^dc-b	2
#	#	fe^dc-ba+*/	1

Step 3: Reverse the Final Answer

Postfix obtained: **fe^dc-ba+*/**
Reversing this gives the **Prefix Expression**:
/*+ab-cd^ef

Thus, the **Prefix Expression** for (a + b) * (c - d) / e^f is: **/*+ab-cd^ef**

Example 16 – Convert the Following String from Infix to Prefix: a^b^c/d*e-f

Step 1: Reverse the given String

Reversed Expression: **f-e*d/c^b^a**

Step 2: Find Nearly Postfix

Character Scanned	Content of Stack	Postfix	Rank
#	#		
f	#f		
-	#-	f	1
e	#-e	f	1
*	#-*	fe	2
d	#-*d	fe	2
/	#-*/	fed	3
c	#-*/c	fed	3
^	#-*/^	fedc	4
b	#-*/^b	fedc	4
^	#-*/^	Fedcb^	5
a	#-*/^a	Fedcb^	5
#	#	**fedcb^a^/*-**	1

Step 3: Reverse the Final Answer

Postfix obtained: **fedcb^a^/*-**
Reversing this gives the **Prefix Expression**:
-*/^a^bcdef

Thus, the **Prefix Expression** for a^b^c/d*e-f is: **-*/^a^bcdef**

EXAMPLES – Infix Expression to Prefix Conversion With Parentheses

Example 17 – Convert the Following String from Infix to Prefix:

((a + b) / d)

Step 1: Reverse the given String

Reversed Expression: **) d /) b + a ((**

Step 2: Find Nearly Postfix

Character Scanned	Content of Stack	Postfix	Rank
)	)		
d	)d		
/	)/	d	1
)	)/)	d	1
b	)/)b	d	1
+	)/)+	d	2
a	)/)+a	db	2
(	)/	dba+	2
(		dba+/	1

Step 3: Reverse the Final Answer

Final Prefix Expression: **/ + a b d**

Example 18 – Convert the Following String from Infix to Prefix: (A * B) + (C + D)

Step 1: Reverse the given String

Reversed Expression: **) D + C (+) B * A (**

Step 2: Find Nearly Postfix

Character Scanned	Content of Stack	Postfix	Rank
)	)		
)	))		
D	))D		
+	))+	D	1
C	))+C	D	1
(	)	DC+	1
+	)+	DC+	1
)	)+)	DC+	1
B	)+)B	DC+	1
*	)+)*	DC+B	2
A	)+)*A	DC+B	2
(	)+	DC+BA*	2
(		DC+BA*+	1

Step 3: Reverse the Final Answer

Final Prefix Expression: **+ * A B + C D**

Postfix to Infix Conversion

- **Always Start with Empty Stack Frame.**
- **Scan the given expression from Left to Right.**
- **While Scanning, If operand gets scanned then PUSH into the stack.**
- **If operator gets scanned then POP TWO recent elements from the TOP of the stack.**
- **The 1st element which get pop is operand 2 (OP2) .**
- **The 2nd element which get pop is operand 1 (OP1) .**
- **Perform the operation – OP1 <Operator> OP2**

- **Now, Push the Answer with () into the stack.**
- **Repeat the process until you finish the given expression.**
- **Perform only 1 operation at a time.**

Example 19 – Given Postfix Expression: ab+c*

Scanned Char	Op1	Op2	Stack
a	-	-	a
b	-	-	a, b
+	a	b	(a + b)
c	-	-	(a + b), c
*	(a + b)	c	((a + b) * c)

Example 20– Given Postfix Expression: AB+EFG/+*

Scanned Char	Op1	Op2	Stack
A	-	-	A
B	-	-	A, B
+	A	B	(A + B)
E	-	-	(A + B), E
F	-	-	(A + B), E, F
G	-	-	(A + B), E, F, G
/	F	G	(A + B), E, (F / G)
+	E	(F / G)	(A + B), (E + (F / G))
*	(A + B)	(E + (F / G))	((A + B) * (E + (F / G)))

Prefix to Infix Conversion

- **Always Start with Empty Stack Frame.**
- **Scan the given expression from Right to Left.**
- **While Scanning, If operand gets scanned then PUSH into the stack.**
- **If operator gets scanned then POPTWO recent elements from the TOP of the stack.**
- **The 1st element which get pop is operand 1 (OP1)**
- **The 2nd element which get pop is operand 2 (OP2)**
- **Perform the operation – OP1 <Operator> OP2**
- **Now, Push the Answer with () into the stack.**
- **Repeat the process until you finish the given expression.**
- **Perform only 1 operation at a time.**

Example 21– Given Prefix Expression: +−+AB/CDE

Character Scanned	Op1	Op2	Stack Content
E	-	-	E
D	-	-	E, D
C	-	-	E, D, C
/	C	D	E, (C / D)
+	(C / D)	E	((C / D) + E)
B	-	-	((C / D) + E), B
A	-	-	((C / D) + E), B, A
-	A	B	((C / D) + E), (A - B)
+	(A - B)	((C / D) + E)	((A - B) + ((C / D) + E))

Final Infix Expression:

(A−B)+((C/D)+E)

Example 22– Given Prefix Expression: * −A/BC−/ADE

Character Scanned	Op1	Op2	Stack Content
E	-	-	E
D	-	-	E, D
A	-	-	E, D, A
/	A	D	E, (A / D)
-	(A / D)	E	((A / D) - E)
C	-	-	((A / D) - E), C
B	-	-	((A / D) - E), C, B
/	B	C	((A / D) - E), (B / C)
A	-	-	((A / D) - E), (B / C), A
-	A	(B / C)	((A / D) - E), (A - (B / C))
*	(A - (B / C))	((A / D) - E)	((A - (B / C)) * ((A / D) - E))

Final Infix Expression:

(A−(B/C))∗((A/D)−E)

Example 23– Given Prefix Expression: ** + 1 2 / 4 2 + 3 5

Character Scanned	Op1	Op2	Stack Content
5	-	-	5
3	-	-	5, 3
+	3	5	8
2	-	-	8, 2
4	-	-	8, 2, 4
/	4	2	8, (4 / 2) = 2
2	-	-	8, 2, 2
1	-	-	8, 2, 2, 1
+	1	2	8, 2, (1 + 2) = 3
*	3	2	8, (3 * 2) = 6
*	6	8	(6 * 8) = 48

Final Evaluated Result: 48

Prefix to Postfix Conversion Using Stack

- **Always Start with Empty Stack Frame.**
- **Scan the given expression from Right to Left.**
- **While Scanning, If operand gets scanned then PUSH into the stack.**
- **If operator gets scanned then POPTWO recent elements from the TOP of the stack.**
- **The 1st element which get pop is operand 1 (OP1)**
- **The 2nd element which get pop is operand 2 (OP2)**
- **Perform the operation – OP1 OP2 <Operator>**
- **Now, Push the Answer into the stack.**
- **Repeat the process until you finish the given expression.**
- **Perform only 1 operation at a time.**

Example 24– Given Prefix Expression: +∗+ab/cd∗gh+ * + a b / c d * g h

Character Scanned	Op1	Op2	Stack Content
h	-	-	h
g	-	-	h, g
*	g	h	g h *
d	-	-	g h *, d
c	-	-	g h *, d, c
/	c	d	g h *, c d /

b	-	-	g h *, c d /, b
a	-	-	g h *, c d /, b, a
+	a	b	g h *, c d /, a b +
*	+ a b	/ c d	g h *, a b + c d / *
+	* + a b / c d	* g h	a b + c d / * g h * +

Final Postfix Expression:

a b + c d / ∗ g h ∗ +

Postfix to Prefix Conversion Using Stack

- **Always Start with Empty Stack Frame.**
- **Scan the given expression from Left to Right.**
- **While Scanning,If operand gets scanned then PUSH into the stack.**
- **If operator gets scanned then POPTWO recent elements from the TOP of the stack.**
- **The 1st element which get pop is operand 2 (OP2)**
- **The 2nd element which get pop is operand 1 (OP1)**
- **Perform the operation – <Operator> OP1 OP2**
- **Now, Push the Answer into the stack.**
- **Repeat the process until you finish the given expression.**
- **Perform only 1 operation at a time.**

Example 25– Given Postfix Expression:

a b + c d / ∗ g h ∗ +a \ b \ + \ c \ d \ / \ * \ g \ h \ * \ +

Character Scanned	Op1	Op2	Stack Content
a	-	-	a
b	-	-	a, b
+	a	b	+ a b
c	-	-	+ a b, c
d	-	-	+ a b, c, d
/	c	d	+ a b, / c d
*	+ a b	/ c d	* + a b / c d
g	-	-	* + a b / c d, g
h	-	-	* + a b / c d, g, h

*	g	h	* + a b / c d, * g h
+	* + a b / c d	* g h	+ * + a b / c d * g h

Final Prefix Expression:

+ ∗ + a b / c d ∗ g h