

# Evaluation of Postfix Expression - GeeksforGeeks

**Source:** <https://www.geeksforgeeks.org/evaluation-of-postfix-expression/>

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Evaluation of Postfix Expression Last Updated : 15 Sep, 2025 Given an array of strings `arr[]` representing a postfix expression, evaluate it. A postfix expression is of the form operand1 operand2 operator (e.g., "a b +"), where two operands are followed by an operator. Note: The operators can include +, -, \*, /, and ^ (where ^ denotes exponentiation, i.e., power). Division / uses floor division. Examples: Input: `arr[] = ["2", "3", "1", "*", "+", "9", "-"]` Output: -4 Explanation: The expression in infix form is:  $2 + (3 * 1) - 9$ . Now, evaluate step by step:  $3 * 1 = 3$ ,  $2 + 3 = 5$ ,  $5 - 9 = -4$  Final Answer: -4 Input : `arr[] = ["2", "3", "^", "1", "+"]` Output : 9 Explanation: If the expression is converted into an infix expression, it will be  $2 ^ 3 + 1 = 8 + 1 = 9$  Try it on GfG Practice [Approach] Using Stack -  $O(n)$  Time and  $O(n)$  Space The idea is to use the property of postfix notation, where two operands are always followed by an operator. We iterate through the expression from left to right, and whenever we encounter an operand, we push it onto the stack. When we encounter an operator, we pop the top two elements from the stack, apply the operator on them, and then push the result back into the stack. Illustration: C++ `#include <iostream> #include <stack> #include <vector> #include <cmath> using namespace std; int floorDiv ( int a , int b ) { if ( a * b < 0 && a % b != 0 ) return ( a / b ) - 1 ; return a / b ; }` `int evaluatePostfix ( vector < string > arr ) { stack < int > st ; for ( string token : arr ) { // If it's an operand (number), push it onto the stack if ( isdigit ( token [ 0 ] ) || ( token . size () > 1 && token [ 0 ] == '-' ) ) { st . push ( stoi ( token ) ); } // Otherwise, it must be an operator else { int val1 = st . top () ; st . pop () ; int val2 = st . top () ; st . pop () ; if ( token == "+" ) st . push ( val2 + val1 ) ; else if ( token == "-" ) st . push ( val2 - val1 ) ; else if ( token == "*" ) st . push ( val2 * val1 ) ; else if ( token == "/" ) st . push ( floorDiv ( val2 , val1 ) ); else if ( token == "^" ) st . push ( ( int ) pow ( val2 , val1 ) ); } } return st . top () ; }` `int main () { vector < string > arr = { "2", "3", "1", "*", "+", "9", "-" }; cout << evaluatePostfix ( arr ) << endl ; return 0 ; }` C `#include <stdio.h> #include <stdlib.h> #include <ctype.h> #include <math.h> #include <string.h> #include <stdbool.h> #include <limits.h> #define MAX 100 typedef struct { int top ; int arr [ MAX ] ; } Stack ; void push ( Stack * s , int val ) { if ( s -> top == MAX - 1 ) { printf ( "Stack Overflow \n " ); return ; } s -> arr [ ++ s -> top ] = val ; } int pop ( Stack * s ) { if ( s -> top == -1 ) { printf ( "Stack Underflow \n " ); return INT_MIN ; } return s -> arr [ s -> top -- ] ; } // Check if token is an operator bool isOperator ( char * token ) { return strcmp ( token , "+" ) == 0 || strcmp ( token , "-" ) == 0 || strcmp ( token , "*" ) == 0 || strcmp ( token , "/" ) == 0 || strcmp ( token , "^" ) == 0 ; } // Floor division for negative numbers int floorDiv ( int a , int b ) { if ( a * b < 0 && a % b != 0 ) return ( a / b ) - 1 ; return a / b ; } // Evaluate postfix expression int evaluatePostfix ( char ** arr , int n ) { Stack s = { -1 } ; for ( int i = 0 ; i < n ; i ++ ) { // If it's an operand (number), push it onto the stack if ( isdigit ( arr [ i ][ 0 ] ) || ( strlen ( arr [ i ] ) > 1 && arr [ i ][ 0 ] == '-' ) ) { push ( & s , atoi ( arr [ i ] ) ); } // Otherwise, it must be an operator else if ( isOperator ( arr [ i ] ) ) { int val1 = pop ( & s ) ; int val2 = pop ( & s ) ; if ( strcmp ( arr [ i ] , "+" ) == 0 ) push ( & s , val2 + val1 ) ; else if ( strcmp ( arr [ i ] , "-" ) == 0 ) push ( & s , val2 - val1 ) ; else if ( strcmp ( arr [ i ] , "*" ) == 0 ) push ( & s , val2 * val1 ) ; else if ( strcmp ( arr [ i ] , "/" ) == 0 ) push ( & s , floorDiv ( val2 , val1 ) ); else if ( strcmp ( arr [ i ] , "^" ) == 0 ) push ( & s , ( int ) pow ( val2 , val1 ) ); } } return pop ( & s ) ; } int main () { char * arr [] = { "2", "3", "1", "*", "+", "9", "-" }; int n = sizeof ( arr ) / sizeof ( arr [ 0 ] ); printf ( "%d \n " , evaluatePostfix ( arr , n ) ); return 0 ; }` Java `import java.util.Stack ; public class GfG { static int floorDiv ( int a , int b ) { if ( a * b < 0 && a % b != 0 ) return ( a / b ) - 1 ; return a / b ; } public static int evaluatePostfix ( String [] arr ) { Stack < Integer > st = new Stack <> () ; for ( String token : arr ) { // If it's an operand (number), push it onto the stack if ( Character . isDigit ( token . charAt ( 0 ) ) || ( token . length () > 1 && token . charAt ( 0 )`

```

== '-' ) { st . push ( Integer . parseInt ( token )); } // Otherwise, it must be an operator else { int val1 = st .
pop (); int val2 = st . pop (); if ( token . equals ( "+" ) ) st . push ( val2 + val1 ); else if ( token . equals ( "-"
) ) st . push ( val2 - val1 ); else if ( token . equals ( "*" ) ) st . push ( val2 * val1 ); else if ( token . equals (
"/" ) ) st . push ( floorDiv ( val2 , val1 )); else if ( token . equals ( "^" ) ) st . push ( ( int ) Math . pow ( val2 ,
val1 )); } } return st . pop (); } public static void main ( String [] args ) { String [] arr = { "2" , "3" , "1" , "*" ,
"+" , "9" , "-" }; System . out . println ( evaluatePostfix ( arr )); } } Python import math def evaluatePostfix
( arr ): st = [] for token in arr : # If it's an operand (number), push it onto the stack if token [ 0 ] . isdigit ()
or ( len ( token ) > 1 and token [ 0 ] == '-' ): st . append ( int ( token )) # Otherwise, it must be an
operator else : val1 = st . pop () val2 = st . pop () if token == '+' : st . append ( val2 + val1 ) elif token ==
'-' : st . append ( val2 - val1 ) elif token == '*' : st . append ( val2 * val1 ) elif token == '/' : st . append (
val2 // val1 ) elif token == '^' : st . append ( int ( math . pow ( val2 , val1 ))) return st . pop () if __name__
== '__main__' : arr = [ "2" , "3" , "1" , "*" , "+" , "9" , "-" ] print ( evaluatePostfix ( arr )) C# using System ;
using System.Collections.Generic ; public class GfG { static int floorDiv ( int a , int b ) { if ( a * b < 0 && a
% b != 0 ) return ( a / b ) - 1 ; return a / b ; } public static int evaluatePostfix ( string [] arr ) { Stack < int >
st = new Stack < int > (); foreach ( string token in arr ) { // If it's an operand (number), push it onto the
stack if ( char . IsDigit ( token [ 0 ]) || ( token . Length > 1 && token [ 0 ] == '-' ) ) { st . Push ( int . Parse (
token )); } // Otherwise, it must be an operator else { int val1 = st . Pop (); int val2 = st . Pop (); if ( token
== "+" ) st . Push ( val2 + val1 ); else if ( token == "-" ) st . Push ( val2 - val1 ); else if ( token == "*" ) st .
Push ( val2 * val1 ); else if ( token == "/" ) st . Push ( floorDiv ( val2 , val1 )); else if ( token == "^" ) st .
Push ( ( int ) Math . Pow ( val2 , val1 )); } } return st . Pop (); } public static void Main () { string [] arr = {
"2" , "3" , "1" , "*" , "+" , "9" , "-" }; Console . WriteLine ( evaluatePostfix ( arr )); } } JavaScript function
evaluatePostfix ( arr ) { let st = []; for ( let token of arr ) { // If token is a number if ( ! isNaN ( token )) { st .
push ( parseInt ( token )); } // Otherwise, it's an operator else { let val1 = st . pop (); let val2 = st . pop ();
if ( token === '+' ) st . push ( val2 + val1 ); else if ( token === '-' ) st . push ( val2 - val1 ); else if ( token
=== '*' ) st . push ( val2 * val1 ); else if ( token === '/' ) st . push ( Math . floor ( val2 / val1 )); else if (
token === '^' ) st . push ( Math . pow ( val2 , val1 )); } } return st . pop (); } // Driver Code let arr = [ "2" ,
"3" , "1" , "*" , "+" , "9" , "-" ]; console . log ( evaluatePostfix ( arr )); Output -4 Comment Article Tags:
Article Tags: Stack DSA Amazon expression-evaluation

```