Stack Data Structure

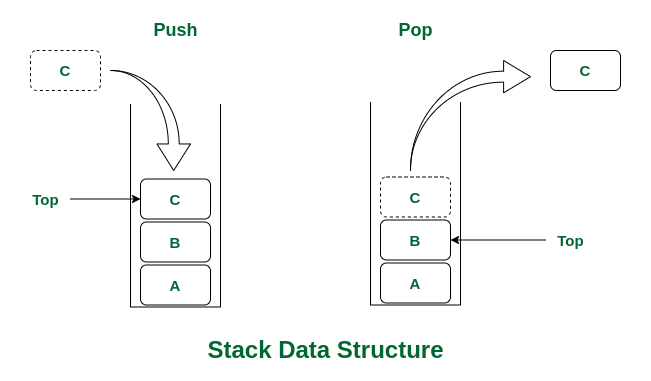
[**Data Structure and Algorithms Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=gfg&utm_medium=header+link+click&utm_campaign=dsa+course+tracker&utm_term=dsa+course+promo&utm_content=stack-lp)

[**Practice Problems on Stack**](https://practice.geeksforgeeks.org/explore/?category%5B%5D=Stack&page=1&category%5B%5D=Stack&utm_source=gfg&utm_medium=header+link+click&utm_campaign=practice+tracker&utm_term=practice+promo&utm_content=stack-lp)

[**Recent articles on Stack**](https://www.geeksforgeeks.org/category/stack/?utm_source=gfg&utm_medium=header+link+click&utm_campaign=recent+article+tracker&utm_term=recent+article+tracker&utm_content=stack-lp)

[**What is Stack?**](https://www.geeksforgeeks.org/introduction-to-stack-data-structure-and-algorithm-tutorials/)

Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).



*Stack Data Structure*

There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO(Last In First Out)/FILO(First In Last Out) order.

**Topics :**

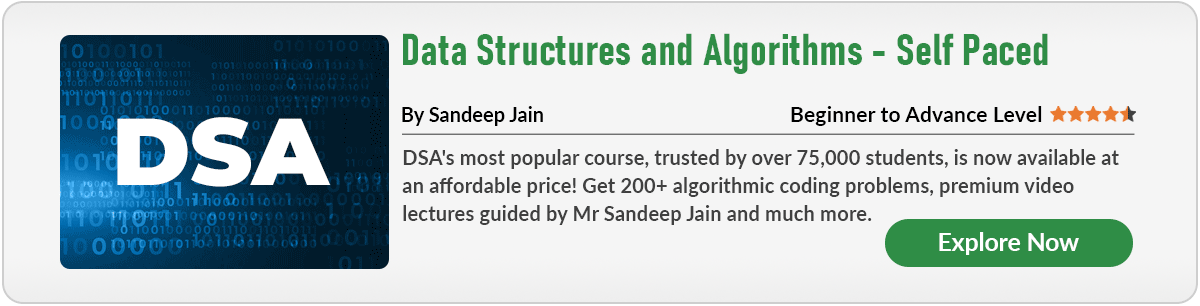
* [Introduction](https://www.geeksforgeeks.org/stack-data-structure/?ref=ghm#intro)
* [Implementation of Stack in different language](https://www.geeksforgeeks.org/stack-data-structure/?ref=ghm#difflang)
* [Some other implementation of Stack](https://www.geeksforgeeks.org/stack-data-structure/?ref=ghm#operations)
* [Standard problems on stack](https://www.geeksforgeeks.org/stack-data-structure/?ref=ghm#standard)
* [Quick Links](https://www.geeksforgeeks.org/stack-data-structure/?ref=ghm#quick)

**Introduction:**

1. [Introduction to Stack – Data Structure and Algorithm Tutorials](https://www.geeksforgeeks.org/introduction-to-stack-data-structure-and-algorithm-tutorials/)
2. [Applications, Advantages and Disadvantages of Stack](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-stack/)
3. [Implement a stack using singly linked list](https://www.geeksforgeeks.org/implement-a-stack-using-singly-linked-list/)
4. [Introduction to Monotonic Stack](https://www.geeksforgeeks.org/introduction-to-monotonic-stack/)

**Implementation of Stack in different languages:**

1. [Stack in C++ STL](https://www.geeksforgeeks.org/stack-in-cpp-stl/)
2. [Stack Class in Java](https://www.geeksforgeeks.org/stack-class-in-java/)
3. [Stack in Python](https://www.geeksforgeeks.org/stack-in-python/)
4. [Stack in C#](https://www.geeksforgeeks.org/c-sharp-stack-with-examples/)
5. [Implementation of Stack in JavaScript](https://www.geeksforgeeks.org/implementation-stack-javascript/)
6. [Stack in Scala](https://www.geeksforgeeks.org/stack-in-scala/)

[](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=page&utm_medium=page&utm_campaign=dsa-self-paced)

**Some other Implementation of Stack:**

1. [Implement Queue using Stacks](https://www.geeksforgeeks.org/queue-using-stacks/)
2. [Design and Implement Special Stack Data Structure | Added Space Optimized Version](https://www.geeksforgeeks.org/design-and-implement-special-stack-data-structure/)
3. [Implement two stacks in an array](https://www.geeksforgeeks.org/implement-two-stacks-in-an-array/)
4. [Implement Stack using Queues](https://www.geeksforgeeks.org/implement-stack-using-queue/)
5. [How to efficiently implement k stacks in a single array?](https://www.geeksforgeeks.org/efficiently-implement-k-stacks-single-array/)
6. [Design a stack that supports getMin() in O(1) time and O(1) extra space](https://www.geeksforgeeks.org/design-a-stack-that-supports-getmin-in-o1-time-and-o1-extra-space/)
7. [Implement a stack using single queue](https://www.geeksforgeeks.org/implement-a-stack-using-single-queue/)
8. [How to implement stack using priority queue or heap?](https://www.geeksforgeeks.org/implement-stack-using-priority-queue-or-heap/)
9. [Create a customized data structure which evaluates functions in O(1)](https://www.geeksforgeeks.org/create-customized-data-structure-evaluates-functions-o1/)
10. [Implement Stack and Queue using Deque](https://www.geeksforgeeks.org/implement-stack-queue-using-deque/)

**Standard problems on Stack:**

* **Easy:**
  1. [Infix to Postfix Conversion using Stack](https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/)
  2. [Prefix to Infix Conversion](https://www.geeksforgeeks.org/prefix-infix-conversion/)
  3. [Prefix to Postfix Conversion](https://www.geeksforgeeks.org/prefix-postfix-conversion/)
  4. [Postfix to Prefix Conversion](https://www.geeksforgeeks.org/postfix-prefix-conversion/)
  5. [Postfix to Infix](https://www.geeksforgeeks.org/postfix-to-infix/)
  6. [Convert Infix To Prefix Notation](https://www.geeksforgeeks.org/convert-infix-prefix-notation/)
  7. [Check for balanced parentheses in an expression](https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/)
  8. [Arithmetic Expression Evalution](https://www.geeksforgeeks.org/arithmetic-expression-evalution/)
  9. [Evaluation of Postfix Expression](https://www.geeksforgeeks.org/stack-set-4-evaluation-postfix-expression/)
  10. [Reverse a stack using recursion](https://www.geeksforgeeks.org/reverse-a-stack-using-recursion/)
  11. [Reverse individual words](https://www.geeksforgeeks.org/reverse-individual-words/)
  12. [Reverse a string using stack](https://www.geeksforgeeks.org/stack-set-3-reverse-string-using-stack/)
  13. [Reversing a Queue](https://www.geeksforgeeks.org/reversing-a-queue/)
* **Medium:**
  1. [How to create mergable stack?](https://www.geeksforgeeks.org/create-mergable-stack/)
  2. [The Stock Span Problem](https://www.geeksforgeeks.org/the-stock-span-problem/)
  3. [Next Greater Element](https://www.geeksforgeeks.org/next-greater-element/)
  4. [Next Greater Frequency Element](https://www.geeksforgeeks.org/next-greater-frequency-element/)
  5. [Maximum product of indexes of next greater on left and right](https://www.geeksforgeeks.org/maximum-product-of-indexes-of-next-greater-on-left-and-right/)
  6. [Iterative Tower of Hanoi](https://www.geeksforgeeks.org/iterative-tower-of-hanoi/)
  7. [Sort a stack using a temporary stack](https://www.geeksforgeeks.org/sort-stack-using-temporary-stack/)
  8. [Reverse a stack without using extra space in O(n)](https://www.geeksforgeeks.org/reverse-stack-without-using-extra-space/)
  9. [Delete middle element of a stack](https://www.geeksforgeeks.org/delete-middle-element-stack/)
  10. [Check if a queue can be sorted into another queue using a stack](https://www.geeksforgeeks.org/check-queue-can-sorted-another-queue-using-stack/)
  11. [Check if an array is stack sortable](https://www.geeksforgeeks.org/check-array-stack-sortable/)
  12. [Iterative Postorder Traversal | Set 1 (Using Two Stacks)](https://www.geeksforgeeks.org/iterative-postorder-traversal/)
  13. [Largest Rectangular Area in a Histogram | Set 2](https://www.geeksforgeeks.org/largest-rectangle-under-histogram/)
  14. [Find maximum of minimum for every window size in a given array](https://www.geeksforgeeks.org/find-the-maximum-of-minimums-for-every-window-size-in-a-given-array/)
  15. [Find index of closing bracket for a given opening bracket in an expression](https://www.geeksforgeeks.org/find-index-closing-bracket-given-opening-bracket-expression/)
  16. [Find maximum difference between nearest left and right smaller elements](https://www.geeksforgeeks.org/find-maximum-difference-between-nearest-left-and-right-smaller-elements/)
  17. [Delete consecutive same words in a sequence](https://www.geeksforgeeks.org/delete-consecutive-words-sequence/)
  18. [Check mirror in n-ary tree](https://www.geeksforgeeks.org/check-mirror-n-ary-tree/)
  19. [Reverse a number using stack](https://www.geeksforgeeks.org/reverse-number-using-stack/)
  20. [Reversing the first K elements of a Queue](https://www.geeksforgeeks.org/reversing-first-k-elements-queue/)
* **Hard:**
  1. [The Celebrity Problem](https://www.geeksforgeeks.org/the-celebrity-problem/)
  2. [Print next greater number of Q queries](https://www.geeksforgeeks.org/print-next-greater-number-q-queries/)
  3. [Iterative Postorder Traversal | Set 2 (Using One Stack)](https://www.geeksforgeeks.org/iterative-postorder-traversal-using-stack/)
  4. [Print ancestors of a given binary tree node without recursion](https://www.geeksforgeeks.org/print-ancestors-of-a-given-binary-tree-node-without-recursion/)
  5. [Length of the longest valid substring](https://www.geeksforgeeks.org/length-of-the-longest-valid-substring/)
  6. [Expression contains redundant bracket or not](https://www.geeksforgeeks.org/expression-contains-redundant-bracket-not/)
  7. [Find index of closing bracket for a given opening bracket in an expression](https://www.geeksforgeeks.org/find-index-closing-bracket-given-opening-bracket-expression/)
  8. [Find if an expression has duplicate parenthesis or not](https://www.geeksforgeeks.org/find-expression-duplicate-parenthesis-not/)
  9. [Find next Smaller of next Greater in an array](https://www.geeksforgeeks.org/find-next-smaller-next-greater-array/)
  10. [Iterative method to find ancestors of a given binary tree](https://www.geeksforgeeks.org/iterative-method-to-find-ancestors-of-a-given-binary-tree/)
  11. [Stack Permutations (Check if an array is stack permutation of other)](https://www.geeksforgeeks.org/stack-permutations-check-if-an-array-is-stack-permutation-of-other/)
  12. [Spaghetti Stack](https://www.geeksforgeeks.org/g-fact-87/)
  13. [Remove brackets from an algebraic string containing + and – operators](https://www.geeksforgeeks.org/remove-brackets-algebraic-string-containing-operators/)
  14. [Range Queries for Longest Correct Bracket Subsequence](https://www.geeksforgeeks.org/range-queries-longest-correct-bracket-subsequence-set-2/)

*From <*[*https://www.geeksforgeeks.org/stack-data-structure/?ref=ghm*](https://www.geeksforgeeks.org/stack-data-structure/?ref=ghm)*>*

**Easy Questions:**

**Convert Infix expression to Postfix expression**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 26 Dec, 2022
* Read
* Discuss(193)
* Courses
* Practice
* Video

Write a program to Convert Infix expression to Postfix.

***Infix expression:****The expression of the form a operator b (a + b). When an operator is in-between every pair of operands.*

***Postfix expression:****The expression of the form a b operator (ab+). When an operator is followed by every pair of operands.*

**Examples:**

***Input:****A + B \* C + D*

***Output:****ABC\*+D+*

***Input:****((A + B) – C \* (D / E)) + F*

***Output:****AB+CDE/\*-F+*

Recommended Problem

Infix to Postfix

[Stack](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Stack&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Paytm](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Paytm&sortBy=submissions)

[VMWare](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=VMWare&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/infix-to-postfix-1587115620/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 51.9K

**Why postfix representation of the expression?**

The compiler scans the expression either from left to right or from right to left.

Consider the expression: **a + b \* c + d**

The compiler first scans the expression to evaluate the expression b \* c, then again scans the expression to add a to it. The result is then added to d after another scan. The repeated scanning makes it very inefficient and Infix expressions are easily readable and solvable by humans whereas the computer cannot differentiate the operators and parenthesis easily so, it is better to convert the expression to postfix(or prefix) form before evaluation.

The corresponding expression in postfix form is **abc\*+d+**. The postfix expressions can be evaluated easily using a stack.

[Recommended PracticeInfix to PostfixTry It!](https://practice.geeksforgeeks.org/problems/infix-to-postfix-1587115620/1/)

[](https://practice.geeksforgeeks.org/courses/dsa-self-paced)

*DSA Self-Paced course*

**Steps to convert Infix expression to Postfix expression using Stack:**

* Scan the infix expression from left to right.
* If the scanned character is an operand, output it.
* Else,
* If the precedence and associativity of the scanned operator are greater than the precedence and associativity of the operator in the stack(or the stack is empty or the stack contains a ‘(‘ ), then push it.
* ‘^’ operator is right associative and other operators like ‘+’,’-‘,’\*’ and ‘/’ are left-associative. Check especially for a condition when both,  operator at the top of the stack and the scanned operator are ‘^’. In this condition, the precedence of the scanned operator is higher due to its right associativity. So it will be pushed into the operator stack. In all the other cases when the top of the operator stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.
* Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)
* If the scanned character is an ‘(‘, push it to the stack.
* If the scanned character is an ‘)’, pop the stack and output it until a ‘(‘ is encountered, and discard both the parenthesis.
* Repeat steps 2-6 until the infix expression is scanned.
* Print the output
* Pop and output from the stack until it is not empty.

Below is the implementation of the above algorithm:

* C++14
* C
* Java
* Python
* C#
* Javascript

# Python program to convert infix expression to postfix

# Class to convert the expression

**class** Conversion:

    # Constructor to initialize the class variables

**def** \_\_init\_\_(self, capacity):

        self.top **= -**1

        self.capacity **=** capacity

        # This array is used a stack

        self.array **=** []

        # Precedence setting

        self.output **=** []

        self.precedence **=** {'+': 1, '-': 1, '\*': 2, '/': 2, '^': 3}

    # check if the stack is empty

**def** isEmpty(self):

**return** True **if** self.top **== -**1 **else** False

    # Return the value of the top of the stack

**def** peek(self):

**return** self.array[**-**1]

    # Pop the element from the stack

**def** pop(self):

**if not** self.isEmpty():

            self.top **-=** 1

**return** self.array.pop()

**else**:

**return** "$"

    # Push the element to the stack

**def** push(self, op):

        self.top **+=** 1

        self.array.append(op)

    # A utility function to check is the given character

    # is operand

**def** isOperand(self, ch):

**return** ch.isalpha()

    # Check if the precedence of operator is strictly

    # less than top of stack or not

**def** notGreater(self, i):

**try**:

            a **=** self.precedence[i]

            b **=** self.precedence[self.peek()]

**return** True **if** a <**=** b **else** False

**except** KeyError:

**return** False

    # The main function that

    # converts given infix expression

    # to postfix expression

**def** infixToPostfix(self, exp):

        # Iterate over the expression for conversion

**for** i **in** exp:

            # If the character is an operand,

            # add it to output

**if** self.isOperand(i):

                self.output.append(i)

            # If the character is an '(', push it to stack

**elif** i **==** '(':

                self.push(i)

            # If the scanned character is an ')', pop and

            # output from the stack until and '(' is found

**elif** i **==** ')':

**while**((**not** self.isEmpty()) **and**

                      self.peek() !**=** '('):

                    a **=** self.pop()

                    self.output.append(a)

**if** (**not** self.isEmpty() **and** self.peek() !**=** '('):

**return -**1

**else**:

                    self.pop()

            # An operator is encountered

**else**:

**while**(**not** self.isEmpty() **and** self.notGreater(i)):

                    self.output.append(self.pop())

                self.push(i)

        # pop all the operator from the stack

**while not** self.isEmpty():

            self.output.append(self.pop())

        print "".join(self.output)

# Driver's code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    exp **=** "a+b\*(c^d-e)^(f+g\*h)-i"

    obj **=** Conversion(len(exp))

    # Function call

    obj.infixToPostfix(exp)

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

abcd^e-fgh\*+^\*+i-

**Time Complexity:**O(N), where N is the size of the infix expression

**Auxiliary Space:**O(N)

*From <*[*https://www.geeksforgeeks.org/convert-infix-expression-to-postfix-expression/*](https://www.geeksforgeeks.org/convert-infix-expression-to-postfix-expression/)*>*

**Prefix to Infix Conversion**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 03 Aug, 2022
* Read
* Discuss(31)
* Courses
* Practice
* Video

**Infix** : An expression is called the Infix expression if the operator appears in between the operands in the expression. Simply of the form (operand1 operator operand2).

**Example :**(A+B) \* (C-D)

**Prefix** : An expression is called the prefix expression if the operator appears in the expression before the operands. Simply of the form (operator operand1 operand2).

**Example :** \*+AB-CD (Infix : (A+B) \* (C-D) )

Given a Prefix expression, convert it into a Infix expression.

Computers usually does the computation in either prefix or postfix (usually postfix). But for humans, its easier to understand an Infix expression rather than a prefix. Hence conversion is need for human understanding.

**Examples:**

**Input :** Prefix : \*+AB-CD  
**Output :** Infix : ((A+B)\*(C-D))

**Input :** Prefix : \*-A/BC-/AKL  
**Output :** Infix : ((A-(B/C))\*((A/K)-L))

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Algorithm for Prefix to Infix**:

* Read the Prefix expression in reverse order (from right to left)
* If the symbol is an operand, then push it onto the Stack
* If the symbol is an operator, then pop two operands from the Stack   
  Create a string by concatenating the two operands and the operator between them.   
  **string = (operand1 + operator + operand2)**   
  And push the resultant string back to Stack
* Repeat the above steps until the end of Prefix expression.
* At the end stack will have only 1 string i.e resultant string

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python Program to convert prefix to Infix

**def** prefixToInfix(prefix):

    stack **=** []

    # read prefix in reverse order

    i **=** len(prefix) **-** 1

**while** i >**=** 0:

**if not** isOperator(prefix[i]):

            # symbol is operand

            stack.append(prefix[i])

            i **-=** 1

**else**:

            # symbol is operator

            str **=** "(" **+** stack.pop() **+** prefix[i] **+** stack.pop() **+** ")"

            stack.append(str)

            i **-=** 1

**return** stack.pop()

**def** isOperator(c):

**if** c **==** "\*" **or** c **==** "+" **or** c **==** "-" **or** c **==** "/" **or** c **==** "^" **or** c **==** "(" **or** c **==** ")":

**return** True

**else**:

**return** False

# Driver code

**if** \_\_name\_\_**==**"\_\_main\_\_":

    str **=** "\*-A/BC-/AKL"

**print**(prefixToInfix(str))

# This code is contributed by avishekarora

**Output**

Infix : ((A-(B/C))\*((A/K)-L))

**Time Complexity:** **O(n)**

**Auxiliary Space: O(n)**

*From <*[*https://www.geeksforgeeks.org/prefix-infix-conversion/*](https://www.geeksforgeeks.org/prefix-infix-conversion/)*>*

**Prefix to Postfix Conversion**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 24 May, 2022
* Read
* Discuss(15)
* Courses
* Practice
* Video

**Prefix**: An expression is called the prefix expression if the operator appears in the expression before the operands. Simply of the form (operator operand1 operand2).

Example : \*+AB-CD (Infix : (A+B) \* (C-D) )

**Postfix**: An expression is called the postfix expression if the operator appears in the expression after the operands. Simply of the form (operand1 operand2 operator).

Example : AB+CD-\* (Infix : (A+B \* (C-D) )

Given a Prefix expression, convert it into a Postfix expression.

Conversion of Prefix expression directly to Postfix without going through the process of converting them first to Infix and then to Postfix is much better in terms of computation and better understanding the expression (Computers evaluate using Postfix expression).

**Examples:**

Input : Prefix : \*+AB-CD  
Output : Postfix : AB+CD-\*  
Explanation : Prefix to Infix : (A+B) \* (C-D)  
 Infix to Postfix : AB+CD-\*

Input : Prefix : \*-A/BC-/AKL  
Output : Postfix : ABC/-AK/L-\*  
Explanation : Prefix to Infix : (A-(B/C))\*((A/K)-L)  
 Infix to Postfix : ABC/-AK/L-\*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Algorithm for Prefix to Postfix**:

* Read the Prefix expression in reverse order (from right to left)
* If the symbol is an operand, then push it onto the Stack
* If the symbol is an operator, then pop two operands from the Stack   
  Create a string by concatenating the two operands and the operator after them.   
  **string = operand1 + operand2 + operator**   
  And push the resultant string back to Stack
* Repeat the above steps until end of Prefix expression.
* C++
* Java
* Python 3
* C#
* Javascript

# Write Python3 code here

# -\*- coding: utf-8 -\*-

# Example Input

s **=** "\*-A/BC-/AKL"

# Stack for storing operands

stack **=** []

operators **=** set(['+', '-', '\*', '/', '^'])

# Reversing the order

s **=** s[::**-**1]

# iterating through individual tokens

**for** i **in** s:

    # if token is operator

**if** i **in** operators:

        # pop 2 elements from stack

        a **=** stack.pop()

        b **=** stack.pop()

        # concatenate them as operand1 +

        # operand2 + operator

        temp **=** a**+**b**+**i

        stack.append(temp)

    # else if operand

**else**:

        stack.append(i)

# printing final output

print(**\***stack)

**Output**

Postfix : ABC/-AK/L-\*

**Time Complexity:** O(N), as we are using a loop for traversing the expression.

**Auxiliary Space:**O(N), as we are using stack for extra space.

*From <*[*https://www.geeksforgeeks.org/prefix-postfix-conversion/*](https://www.geeksforgeeks.org/prefix-postfix-conversion/)*>*

**Postfix to Prefix Conversion**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 03 Aug, 2022
* Read
* Discuss(9)
* Courses
* Practice
* Video

**Postfix**: An expression is called the postfix expression if the operator appears in the expression after the operands. Simply of the form (operand1 operand2 operator).

**Example :** AB+CD-\* (Infix : (A+B) \* (C-D) )

**Prefix** : An expression is called the prefix expression if the operator appears in the expression before the operands. Simply of the form (operator operand1 operand2).

**Example :** \*+AB-CD (Infix : (A+B) \* (C-D) )

Given a Postfix expression, convert it into a Prefix expression.

Conversion of Postfix expression directly to Prefix without going through the process of converting them first to Infix and then to Prefix is much better in terms of computation and better understanding the expression (Computers evaluate using Postfix expression).

**Examples:**

**Input :** Postfix : AB+CD-\*  
**Output :** Prefix : \*+AB-CD  
**Explanation :** Postfix to Infix : (A+B) \* (C-D)  
 Infix to Prefix : \*+AB-CD

**Input :** Postfix : ABC/-AK/L-\*  
**Output :** Prefix : \*-A/BC-/AKL  
**Explanation :** Postfix to Infix : ((A-(B/C))\*((A/K)-L))  
 Infix to Prefix : \*-A/BC-/AKL

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Algorithm for Postfix to Prefix**:

* *Read the Postfix expression from left to right*
* *If the symbol is an operand, then push it onto the Stack*
* *If the symbol is an operator, then pop two operands from the Stack   
  Create a string by concatenating the two operands and the operator before them.****string = operator + operand2 + operand1****And push the resultant string back to Stack*
* *Repeat the above steps until end of Postfix expression.*

 Below is the implementation of the above idea:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 Program to convert postfix to prefix

# function to check if

# character is operator or not

**def** isOperator(x):

**if** x **==** "+":

**return** True

**if** x **==** "-":

**return** True

**if** x **==** "/":

**return** True

**if** x **==** "\*":

**return** True

**return** False

# Convert postfix to Prefix expression

**def** postToPre(post\_exp):

    s **=** []

    # length of expression

    length **=** len(post\_exp)

    # reading from right to left

**for** i **in** range(length):

        # check if symbol is operator

**if** (isOperator(post\_exp[i])):

            # pop two operands from stack

            op1 **=** s[**-**1]

            s.pop()

            op2 **=** s[**-**1]

            s.pop()

            # concat the operands and operator

            temp **=** post\_exp[i] **+** op2 **+** op1

            # Push string temp back to stack

            s.append(temp)

        # if symbol is an operand

**else**:

            # push the operand to the stack

            s.append(post\_exp[i])

    ans **=** ""

**for** i **in** s:

        ans **+=** i

**return** ans

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    post\_exp **=** "AB+CD-"

    # Function call

    print("Prefix : ", postToPre(post\_exp))

# This code is contributed by AnkitRai01

**Output**

**Prefix : \*-A/BC-/AKL**

**Time Complexity: O(N)** where N is the length of the string

**Auxiliary Space: O(N)** where N is the stack size.

*From <*[*https://www.geeksforgeeks.org/postfix-prefix-conversion/*](https://www.geeksforgeeks.org/postfix-prefix-conversion/)*>*

**Postfix to Infix**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 24 May, 2022
* Read
* Discuss(30)
* Courses
* Practice
* Video

**Infix expression**: The expression of the form a op b. When an operator is in-between every pair of operands.

**Postfix expression**: The expression of the form a b op. When an operator is followed for every pair of operands.

Postfix notation, also known as reverse Polish notation, is a syntax for mathematical expressions in which the mathematical operator is always placed after the operands. Though postfix expressions are easily and efficiently evaluated by computers, they can be difficult for humans to read. Complex expressions using standard parenthesized infix notation are often more readable than the corresponding postfix expressions. Consequently, we would sometimes like to allow end users to work with infix notation and then convert it to postfix notation for computer processing. Sometimes, moreover, expressions are stored or generated in postfix, and we would like to convert them to infix for the purpose of reading and editing

**Examples:**

Input : abc++  
Output : (a + (b + c))

Input : ab\*c+  
Output : ((a\*b)+c)

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have already discussed [Infix to Postfix](https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/). Below is algorithm for Postfix to Infix.

**Algorithm**

1.While there are input symbol left

…1.1 Read the next symbol from the input.

2.If the symbol is an operand

…2.1 Push it onto the stack.

3.Otherwise,

…3.1 the symbol is an operator.

…3.2 Pop the top 2 values from the stack.

…3.3 Put the operator, with the values as arguments and form a string.

…3.4 Push the resulted string back to stack.

4.If there is only one value in the stack

…4.1 That value in the stack is the desired infix string.

Below is the implementation of above approach:

* C++
* Java
* Python3
* C#
* PHP
* Javascript

# Python3 program to find infix for

# a given postfix.

**def** isOperand(x):

**return** ((x >**=** 'a' **and** x <**=** 'z') **or**

            (x >**=** 'A' **and** x <**=** 'Z'))

# Get Infix for a given postfix

# expression

**def** getInfix(exp) :

    s **=** []

**for** i **in** exp:

        # Push operands

**if** (isOperand(i)) :

            s.insert(0, i)

        # We assume that input is a

        # valid postfix and expect

        # an operator.

**else**:

            op1 **=** s[0]

            s.pop(0)

            op2 **=** s[0]

            s.pop(0)

            s.insert(0, "(" **+** op2 **+** i **+**

                             op1 **+** ")")

    # There must be a single element in

    # stack now which is the required

    # infix.

**return** s[0]

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    exp **=** "ab\*c+"

    print(getInfix(exp.strip()))

# This code is contributed by

# Shubham Singh(SHUBHAMSINGH10)

**Output:**

((a\*b)+c)

**Time Complexity:**O(N) where N is the length of the string

**Auxiliary Space:**O(N) where N is the stack size.

*From <*[*https://www.geeksforgeeks.org/postfix-to-infix/*](https://www.geeksforgeeks.org/postfix-to-infix/)*>*

**Convert Infix To Prefix Notation**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 12 Aug, 2022
* Read
* Discuss(57)
* Courses
* Practice
* Video

Humans use *infix* expressions on a day-to-day basis, but the nature of infix notation means that compound expressions are generally not evaluated in a linear fashion.  Conventions governing operator precedence—which includes the use of parentheses—are a means of removing the very likely possibility that a given combination of operators and operands would otherwise give rise to ambiguous expressions, such as 12 ÷ 6 × 3, which could evaluate to either 6 or ⅔.  Infix expressions are, therefore, not evaluated in a linear, left-to-right fashion, but in an order of priority determined by the laws of operator precedence.  This doesn’t lend itself to the way computers generally perform operations: sequentially, and one-at-a-time.  *Prefix* and *postfix* notations, on the other hand, build compound expressions that are never ambiguous, as they only ever yield a singular possible result that depends solely on the order in which the operators and operands appear in an expression when processed in a linear fashion.  Without the added complication of parentheses and other rules of precedence, computers can evaluate prefix and postfix expressions simply as a set of instructions for them to follow in the order they are given.

Given two operands

and

and an operator

, the infix notation implies that O will be placed in between a and b i.e

. When the operator is placed after both operands i.e

, it is called postfix notation. And when the operator is placed before the operands i.e

, the expression in prefix notation.

Given any infix expression, we can obtain the equivalent prefix and postfix format.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Examples:**

Input : A \* B + C / D  
Output : + \* A B/ C D

Input : (A - B/C) \* (A/K-L)  
Output : \*-A/BC-/AKL

To convert an infix to postfix expression refer to this article [Stack | Set 2 (Infix to Postfix)](https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/). We use the same to convert Infix to Prefix.

* 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 prefix is +A\*BC.

Note that for Step 2, we don’t use the postfix algorithm as it is. There is a minor change in the algorithm. As per <https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/> , we have to pop all the operators from the stack which are**greater than or equal to** in precedence than that of the scanned operator. But here, we have to pop all the operators from the stack which are **greater**in precedence than that of the scanned operator. Only in the case of “^” operator, we pop operators from the stack which are **greater than or equal to** in precedence.

*From <*[*https://www.geeksforgeeks.org/convert-infix-prefix-notation/*](https://www.geeksforgeeks.org/convert-infix-prefix-notation/)*>*

**Check for Balanced Brackets in an expression (well-formedness) using Stack**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 22 Sep, 2022
* Read
* Discuss(280)
* Courses
* Practice
* Video

Given an expression string **exp**, write a program to examine whether the pairs and the orders of “{“, “}”, “(“, “)”, “[“, “]” are correct in the given expression.

**Example**:

***Input****: exp = “[()]{}{[()()]()}”*

***Output****: Balanced*

***Explanation:****all the brackets are well-formed*

***Input****: exp = “[(])”*

***Output****: Not Balanced*

***Explanation: 1****and****4****brackets are not balanced because*

*there is a closing ‘]’ before the closing ‘(‘*

Recommended Problem

Parenthesis Checker

[Stack](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Stack&sortBy=submissions)

[STL](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=STL&sortBy=submissions)

+2 more

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+8 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/parenthesis-checker2744/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

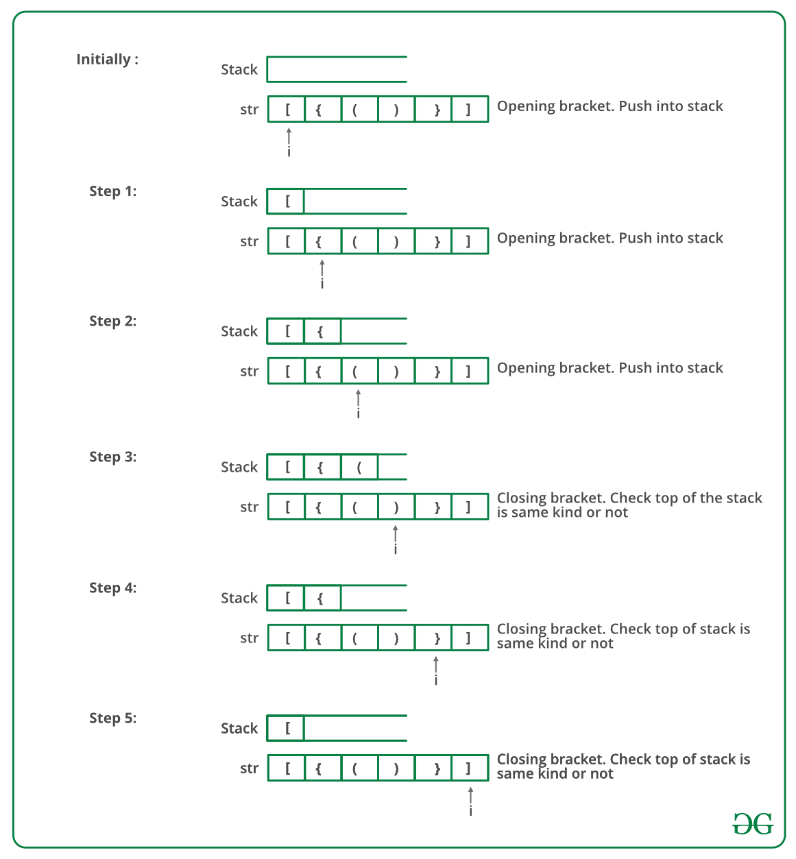
Submission count: 4L

**Check for Balanced Bracket expression using**[**Stack**](https://www.geeksforgeeks.org/introduction-to-stack-data-structure-and-algorithm-tutorials/)**:**

*The idea is to put all the opening brackets in the stack. Whenever you hit a closing bracket, search if the top of the stack is the opening bracket of the same nature. If this holds then pop the stack and continue the iteration, in the end if the stack is empty, it means all brackets are well-formed . Otherwise, they are not balanced.*

**Illustration:**

Below is the illustration of the above approach.



Follow the steps mentioned below to implement the idea:

* Declare a character [stack](https://www.geeksforgeeks.org/stack-data-structure/) (say **temp**).
* Now traverse the string exp.
* If the current character is a starting bracket ( **‘(‘ or ‘{‘  or ‘[‘**) then push it to stack.
* If the current character is a closing bracket ( **‘)’ or ‘}’ or ‘]’**) then pop from stack and if the popped character is the matching starting bracket then fine.
* Else brackets are**Not Balanced**.
* After complete traversal, if there is some starting bracket left in stack then **Not balanced**, else **Balanced**.

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 program to check for

# balanced brackets.

# function to check if

# brackets are balanced

**def** areBracketsBalanced(expr):

    stack **=** []

    # Traversing the Expression

**for** char **in** expr:

**if** char **in** ["(", "{", "["]:

            # Push the element in the stack

            stack.append(char)

**else**:

            # IF current character is not opening

            # bracket, then it must be closing.

            # So stack cannot be empty at this point.

**if not** stack:

**return** False

            current\_char **=** stack.pop()

**if** current\_char **==** '(':

**if** char !**=** ")":

**return** False

**if** current\_char **==** '{':

**if** char !**=** "}":

**return** False

**if** current\_char **==** '[':

**if** char !**=** "]":

**return** False

    # Check Empty Stack

**if** stack:

**return** False

**return** True

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    expr **=** "{()}[]"

    # Function call

**if** areBracketsBalanced(expr):

**print**("Balanced")

**else**:

**print**("Not Balanced")

# This code is contributed by AnkitRai01 and improved

# by Raju Pitta

**Output**

Balanced

**Time Complexity:** O(N), Iteration over the string of size N one time.

**Auxiliary Space:** O(N) for stack.

*From <*[*https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/*](https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/)*>*

**Arithmetic Expression Evaluation**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 03 May, 2021
* Read
* Discuss(1)
* Courses
* Practice
* Video

The stack organization is very effective in evaluating arithmetic expressions. Expressions are usually represented in what is known as **Infix notation**, in which each operator is written between two operands (i.e., A + B). With this notation, we must distinguish between ( A + B )\*C and A + ( B \* C ) by using either parentheses or some operator-precedence convention. Thus, the order of operators and operands in an arithmetic expression does not uniquely determine the order in which the operations are to be performed.

**1. Polish notation (prefix notation) –**

It refers to the notation in which the operator is placed before its two operands. Here no parentheses are required, i.e.,

+AB

**2. Reverse Polish notation(postfix notation) –**

It refers to the analogous notation in which the operator is placed after its two operands. Again, no parentheses is required in Reverse Polish notation, i.e.,

AB+

Stack-organized computers are better suited for post-fix notation than the traditional infix notation. Thus, the infix notation must be converted to the postfix notation. The conversion from infix notation to postfix notation must take into consideration the operational hierarchy.

There are 3 levels of precedence for 5 binary operators as given below:

Highest: Exponentiation (^)  
Next highest: Multiplication (\*) and division (/)  
Lowest: Addition (+) and Subtraction (-)

**For example –**

Infix notation: (A-B)\*[C/(D+E)+F]  
Post-fix notation: AB- CDE +/F +\*

Here, we first perform the arithmetic inside the parentheses (A-B) and (D+E). The division of C/(D+E) must be done prior to the addition with F. After that multiply the two terms inside the parentheses and bracket.

Now we need to calculate the value of these arithmetic operations by using a stack.

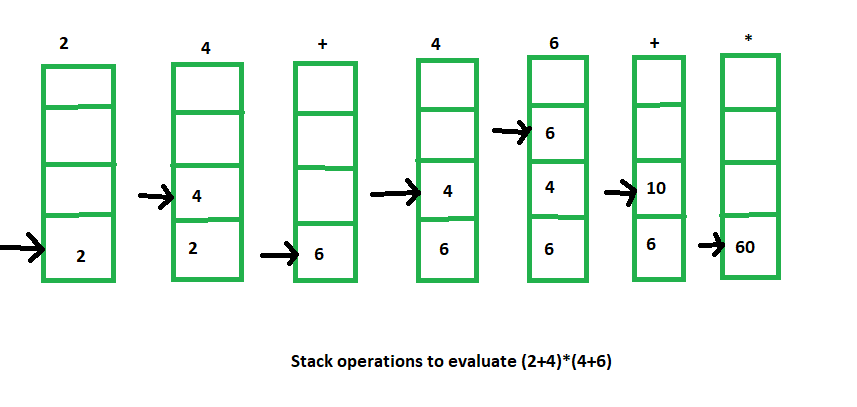
The procedure for getting the result is:

1. Convert the expression in Reverse Polish notation( post-fix notation).
2. Push the operands into the stack in the order they appear.
3. When any operator encounters then pop two topmost operands for executing the operation.
4. After execution push the result obtained into the stack.
5. After the complete execution of expression, the final result remains on the top of the stack.

**For example –**

Infix notation: (2+4) \* (4+6)  
Post-fix notation: 2 4 + 4 6 + \*  
Result: 60

The stack operations for this expression evaluation is shown below:



*From <*[*https://www.geeksforgeeks.org/arithmetic-expression-evalution/*](https://www.geeksforgeeks.org/arithmetic-expression-evalution/)*>*

**Stack | Set 4 (Evaluation of Postfix Expression)**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 22 Aug, 2022
* Read
* Discuss(43)
* Courses
* Practice
* Video

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis is not required in postfix. We have discussed [infix to postfix conversion](https://www.geeksforgeeks.org/stack-set-2-infix-to-postfix/). In this post, the evaluation of postfix expressions is discussed.

Recommended Problem

Evaluation of Postfix Expression

[Stack](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Stack&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/evaluation-of-postfix-expression1735/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 62.5K

**Example:**

***Input****: str = “2 3 1 \* + 9 -“*

***Output****: -4*

***Explanation:***

* *Scan 2, it’s a number, so push it to stack. Stack contains ‘2’*
* *Scan 3, again a number, push it to stack, stack now contains ‘2 3’ (from bottom to top)*
* *Scan 1, again a number, push it to stack, stack now contains ‘2 3 1’*
* *Scan \*, it’s an operator, pop two operands from stack, apply the \* operator on operands, we get 3\*1 which results in 3. We push the result 3 to stack. The stack now becomes ‘2 3’.*
* *Scan +, it’s an operator, pop two operands from stack, apply the + operator on operands, we get 3 + 2 which results in 5. We push the result 5 to stack. The stack now becomes ‘5’.*
* *Scan 9, it’s a number, so we push it to the stack. The stack now becomes ‘5 9’.*
* *Scan -, it’s an operator, pop two operands from stack, apply the – operator on operands, we get 5 – 9 which results in -4. We push the result -4 to the stack. The stack now becomes ‘-4’.*
* *There are no more elements to scan, we return the top element from the stack (which is the only element left in a stack).*

***Input****: str = “100 200 + 2 / 5 \* 7 +”*

***Output****: 757*

**Evaluation of Postfix Expression Using Stack:**

Follow the steps mentioned below to evaluate postfix expression using stack:

* Create a stack to store operands (or values).
* Scan the given expression from left to right and do the following for every scanned element.
* If the element is a number, push it into the stack
* If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack
* When the expression is ended, the number in the stack is the final answer

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to evaluate value of a postfix expression

# Class to convert the expression

**class** Evaluate:

    # Constructor to initialize the class variables

**def** \_\_init\_\_(self, capacity):

        self.top **= -**1

        self.capacity **=** capacity

        # This array is used a stack

        self.array **=** []

    # check if the stack is empty

**def** isEmpty(self):

**return** True **if** self.top **== -**1 **else** False

    # Return the value of the top of the stack

**def** peek(self):

**return** self.array[**-**1]

    # Pop the element from the stack

**def** pop(self):

**if not** self.isEmpty():

            self.top **-=** 1

**return** self.array.pop()

**else**:

**return** "$"

    # Push the element to the stack

**def** push(self, op):

        self.top **+=** 1

        self.array.append(op)

    # The main function that converts given infix expression

    # to postfix expression

**def** evaluatePostfix(self, exp):

        # Iterate over the expression for conversion

**for** i **in** exp:

            # If the scanned character is an operand

            # (number here) push it to the stack

**if** i.isdigit():

                self.push(i)

            # If the scanned character is an operator,

            # pop two elements from stack and apply it.

**else**:

                val1 **=** self.pop()

                val2 **=** self.pop()

                self.push(str(eval(val2 **+** i **+** val1)))

**return** int(self.pop())

# Driver program to test above function

exp **=** "231\*+9-"

obj **=** Evaluate(len(exp))

print ("postfix evaluation: %d"**%**(obj.evaluatePostfix(exp)))

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

postfix evaluation: -4

**Time Complexity:**O(N)

**Auxiliary Space:** O(N)

**There are the following limitations of the above implementation.**

* It supports only **4** binary operators**‘+’, ‘\*’, ‘-‘**and**‘/’**. It can be extended for more operators by adding more switch cases.
* The allowed operands are only single-digit operands. The program can be extended for multiple digits by adding a separator-like space between all elements **(operators and operands)** of the given expression.

Below given is the extended program which allows operands to have multiple digits.

* C++14
* C
* Java
* Python3
* C#
* Javascript

# Python program to evaluate value of a postfix

# expression with integers containing multiple digits

**class** evalpostfix:

**def** \_\_init\_\_(self):

        self.stack **=**[]

        self.top **=-**1

**def** pop(self):

**if** self.top **==-**1:

**return**

**else**:

            self.top**-=** 1

**return** self.stack.pop()

**def** push(self, i):

        self.top**+=** 1

        self.stack.append(i)

**def** centralfunc(self, ab):

**for** i **in** ab:

            # if the component of the list is an integer

**try**:

                self.push(int(i))

            # if the component of the list is not an integer,

            # it must be an operator. Using ValueError, we can

            # evaluate components of the list other than type int

**except** ValueError:

                val1 **=** self.pop()

                val2 **=** self.pop()

**if** i **==** '/':

                  self.push(val2 **/** val1)

**else**:

                # switch statement to perform operation

                  switcher **=**{'+':val2 **+** val1, '-':val2**-**val1, '\*':val2 **\*** val1, '^':val2**\*\***val1}

                  self.push(switcher.get(i))

**return** int(self.pop())

str **=**'100 200 + 2 / 5 \* 7 +'

# splitting the given string to obtain

# integers and operators into a list

strconv **=** str.split(' ')

obj **=** evalpostfix()

**print**(obj.centralfunc(strconv))

# This code is contributed by Amarnath Reddy

**Output**

757

**Time Complexity:**O(N)

**Auxiliary Space:** O(N)

*From <*[*https://www.geeksforgeeks.org/stack-set-4-evaluation-postfix-expression/*](https://www.geeksforgeeks.org/stack-set-4-evaluation-postfix-expression/)*>*

**How to Reverse a Stack using Recursion**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 02 Dec, 2022
* Read
* Discuss(235)
* Courses
* Practice
* Video

Write a program to reverse a stack using recursion, without using any loop.

**Example:**

***Input****: elements present in stack from top to bottom 1 2 3 4*

***Output****: 4 3 2 1*

***Input****: elements present in stack from top to bottom 1 2 3*

***Output****: 3 2 1*

Recommended Problem

Reverse a Stack

[Recursion](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Recursion&sortBy=submissions)

[Stack](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Stack&sortBy=submissions)

+2 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/reverse-a-stack/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 14.7K

**Reverse a stack using Recursion**

*The idea of the solution is to hold all values in Function Call Stack until the stack becomes empty. When the stack becomes empty, insert all held items one by one at the bottom of the stack.*

**Illustration:**

Below is the illustration of the above approach

* *Let given stack be*

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

* *After all calls of reverse,  4 will be passed to function insert at bottom, after that 4 will pushed to the stack when stack is empty*

        4

* *Then 3 will be passed to function insert at bottom , it will check if the stack is empty or not if not then pop all the elements back and insert 3 and then push other elements back.*

|  |
| --- |
| 4 |
| 3 |

* *Then 2 will be passed to function insert at bottom , it will check if the stack is empty or not if not then pop all the elements back and insert 2 and then push other elements back.*

|  |
| --- |
| 4 |
| 3 |
| 2 |

* *Then 1 will be passed to function insert at bottom , it will check if the stack is empty or not if not then pop all the elements back and insert 1 and then push other elements back.*

|  |
| --- |
| 4 |
| 3 |
| 2 |
| 1 |

 Follow the steps mentioned below to implement the idea:

* Create a **stack** and push all the elements in it.
* Call **reverse()**, which will pop all the elements from the stack and pass the popped element to function **insert\_at\_bottom()**
* Whenever **insert\_at\_bottom()** is called it will insert the passed element at the bottom of the stack.
* Print the **stack**

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to reverse a

# stack using recursion

# Below is a recursive function

# that inserts an element

# at the bottom of a stack.

**def** insertAtBottom(stack, item):

**if** isEmpty(stack):

        push(stack, item)

**else**:

        temp **=** pop(stack)

        insertAtBottom(stack, item)

        push(stack, temp)

# Below is the function that

# reverses the given stack

# using insertAtBottom()

**def** reverse(stack):

**if not** isEmpty(stack):

        temp **=** pop(stack)

        reverse(stack)

        insertAtBottom(stack, temp)

# Below is a complete running

# program for testing above

# functions.

# Function to create a stack.

# It initializes size of stack

# as 0

**def** createStack():

    stack **=** []

**return** stack

# Function to check if

# the stack is empty

**def** isEmpty(stack):

**return** len(stack) **==** 0

# Function to push an

# item to stack

**def** push(stack, item):

    stack.append(item)

# Function to pop an

# item from stack

**def** pop(stack):

    # If stack is empty

    # then error

**if**(isEmpty(stack)):

        print("Stack Underflow ")

        exit(1)

**return** stack.pop()

# Function to print the stack

**def** prints(stack):

**for** i **in** range(len(stack)**-**1, **-**1, **-**1):

        print(stack[i], end**=**' ')

    print()

# Driver Code

stack **=** createStack()

push(stack, str(4))

push(stack, str(3))

push(stack, str(2))

push(stack, str(1))

print("Original Stack ")

prints(stack)

reverse(stack)

print("Reversed Stack ")

prints(stack)

# This code is contributed by Sunny Karira

**Output**

Original Stack  
4 3 2 1   
Reversed Stack  
1 2 3 4

**Time Complexity:** O(N2).

**Auxiliary Space:**O(N) use of Stack

*From <*[*https://www.geeksforgeeks.org/reverse-a-stack-using-recursion/*](https://www.geeksforgeeks.org/reverse-a-stack-using-recursion/)*>*

**Reverse individual words**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 22 Dec, 2022
* Read
* Discuss(12)
* Courses
* Practice
* Video

 Output skeeG rof skeeG Given string str, we need to print the reverse of individual words.

**Examples:**

Input : Hello World  
Output : olleH dlroW  
   
Input : Geeks for Geeks  
Output : skeeG rof skeeG

Recommended Problem

Reverse each word in a given string

[Strings](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Strings&sortBy=submissions)

[Stack](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Stack&sortBy=submissions)

+1 more

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+6 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/reverse-each-word-in-a-given-string1001/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 32K

**Method 1 (Simple):** Generate all words separated by space. One by one reverse words and print them separated by space.

**Method 2 (Space Efficient):** We use a stack to push all words before space. As soon as we encounter a space, we empty the stack.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to reverse individual words

# in a given string using STL list

# reverses individual words of a string

**def** reverseWords(string):

    st **=** list()

    # Traverse given string and push all characters

    # to stack until we see a space.

**for** i **in** range(len(string)):

**if** string[i] !**=** " ":

            st.append(string[i])

        # When we see a space, we print

        # contents of stack.

**else**:

**while** len(st) > 0:

                print(st[**-**1], end**=** "")

                st.pop()

            print(end **=** " ")

    # Since there may not be space after

    # last word.

**while** len(st) > 0:

**print**(st[**-**1], end **=** "")

        st.pop()

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    string **=** "Geeks for Geeks"

    reverseWords(string)

# This code is contributed by

# sanjeev2552

**Output**

skeeG rof skeeG

**Time Complexity:**O(n), where n is the length of the string

**Auxiliary Space:** O(n), where n is the length of the string

*From <*[*https://www.geeksforgeeks.org/reverse-individual-words/*](https://www.geeksforgeeks.org/reverse-individual-words/)*>*

**How to Reverse a String using Stack**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 17 Aug, 2022
* Read
* Discuss(12)
* Courses
* Practice
* Video

Given a string, reverse it using stack.

**Example**:

***Input****: str = “GeeksQuiz”*

***Output****: ziuQskeeG*

***Input****: str = “abc”*

***Output****: cba*

Recommended Problem

Reverse a string using Stack

[Stack](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Stack&sortBy=submissions)

[Strings](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Strings&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/reverse-a-string-using-stack/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 70.8K

**Approach:**

*The idea is to create an empty*[*stack*](https://www.geeksforgeeks.org/stack-data-structure/)*and push all the characters from the string into it. Then pop each character one by one from the stack and put them back into the input string starting from the 0’th index. As we all know, stacks work on the principle of first in, last out. After popping all the elements and placing them back to string, the formed string would be reversed.*

Follow the steps given below to reverse a string using stack.

* Create an empty stack.
* One by one push all characters of string to stack.
* One by one pop all characters from stack and put them back to string.

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to reverse a string using stack

# Function to create an empty stack.

# It initializes size of stack as 0

**def** createStack():

    stack **=** []

**return** stack

# Function to determine the size of the stack

**def** size(stack):

**return** len(stack)

# Stack is empty if the size is 0

**def** isEmpty(stack):

**if** size(stack) **==** 0:

**return** true

# Function to add an item to stack .

# It increases size by 1

**def** push(stack, item):

    stack.append(item)

# Function to remove an item from stack.

# It decreases size by 1

**def** pop(stack):

**if** isEmpty(stack):

**return**

**return** stack.pop()

# A stack based function to reverse a string

**def** reverse(string):

    n **=** len(string)

    # Create a empty stack

    stack **=** createStack()

    # Push all characters of string to stack

**for** i **in** range(0, n, 1):

        push(stack, string[i])

    # Making the string empty since all

    # characters are saved in stack

    string **=** ""

    # Pop all characters of string and

    # put them back to string

**for** i **in** range(0, n, 1):

        string **+=** pop(stack)

**return** string

# Driver program to test above functions

string **=** "GeeksQuiz"

string **=** reverse(string)

print("Reversed string is " **+** string)

# This code is contributed by Sunny Karira

**Output**

Reversed string is ziuQskeeG

**Time Complexity:**O(N)

**Auxiliary Space:**O(N) for Stack.

*From <*[*https://www.geeksforgeeks.org/stack-set-3-reverse-string-using-stack/*](https://www.geeksforgeeks.org/stack-set-3-reverse-string-using-stack/)*>*

**Reversing a Queue**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 05 Dec, 2022
* Read
* Discuss(22)
* Courses
* Practice
* Video

Give an algorithm for reversing a queue **Q**. Only the following standard operations are allowed on queue.

1. **enqueue(x)**: Add an item x to the rear of the queue.
2. **dequeue()**: Remove an item from the front of the queue.
3. **empty()**: Checks if a queue is empty or not.

The task is to reverse the queue.

**Examples:**

***Input:****Q = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]*

***Output:****Q = [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]*

***Input:****[1, 2, 3, 4, 5]*

***Output:****[5, 4, 3, 2, 1]*

Recommended Problem

Queue Reversal

[Queue](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Queue&sortBy=submissions)

[Stack](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Stack&sortBy=submissions)

+1 more

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/queue-reversal/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 56K

**Reversing a Queue using**[stack](https://www.geeksforgeeks.org/stack-data-structure/)**:**

*For reversing the queue one approach could be to store the elements of the queue in a temporary data structure in a manner such that if we re-insert the elements in the queue they would get inserted in reverse order. So now our task is to choose such a data structure that can serve the purpose. According to the approach, the data structure should have the property of ‘LIFO’ as the last element to be inserted in the data structure should actually be the first element of the reversed queue.*

Follow the below steps to implement the idea:

* Pop the elements from the queue and insert into the stack now topmost element of the stack is the last element of the queue.
* Pop the elements of the stack to insert back into the queue the last element is the first one to be inserted into the queue.

Below is the implementation of above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to reverse a queue

**from** collections **import** deque

# Function to reverse the queue

**def** reversequeue(queue):

    Stack **=** []

**while** (queue):

        Stack.append(queue[0])

        queue.popleft()

**while** (len(Stack) !**=** 0):

        queue.append(Stack[**-**1])

        Stack.pop()

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    queue **=** deque([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

    reversequeue(queue)

    print(queue)

# This code is contributed by PranchalK

**Output**

100 90 80 70 60 50 40 30 20 10

**Time Complexity:** O(N),As we need to insert all the elements in the stack and later to the queue.

**Auxiliary Space:** O(N), Use of stack to store values.

**Reversing a Queue using**[recursion](https://www.geeksforgeeks.org/recursion/)**:**

*Instead of explicitly using stack goal can be achieved using recursion (recursion at backend will itself maintain stack).*

Follow the below steps to implement the idea:

* Recursively perform the following steps:
* If the queue size is **0**return.
* Else pop and store the front element and recur for remaining queue.
* push the current element in the queue.

Thank you [**Nakshatra Chhillar**](https://www.linkedin.com/in/nakshatrac/) for suggesting this approach and contributing the code

Below is the implementation of above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to reverse a Queue

# Utility function to print the queue

**def** Print(Queue):

**while** (len(Queue) > 0):

**print**(Queue[0],end **=** " ")

        Queue.pop(0)

# Function to reverse the queue

**def** reverseQueue(q):

    # base case

**if** (len(q) **==** 0):

**return**

    # storing front(first element) of queue

    fr **=** q[0]

    # removing front

    q.pop(0)

    # asking recursion to reverse the

    # leftover queue

    reverseQueue(q)

    # placing first element

    # at its correct position

    q.append(fr)

# Driver code

Queue **=** []

Queue.append(10)

Queue.append(20)

Queue.append(30)

Queue.append(40)

Queue.append(50)

Queue.append(60)

Queue.append(70)

Queue.append(80)

Queue.append(90)

Queue.append(100)

reverseQueue(Queue)

Print(Queue)

# This code is contributed by akashish\_\_

**Output**

100 90 80 70 60 50 40 30 20 10

**Time Complexity:** O(N).

**Auxiliary Space:** O(N). The recursion stack contains all elements of queue at a moment.