Lesson:

Stacks 3

PW SKILLS

Pre-Requisites

- Basics of stacks

List of Concepts Involved

- Problems related to Stacks
- Understanding infix, postfix and prefix expressions
- Evaluation of an infix expressions
- Evaluation of an postfix expressions
- Evaluation of an prefix expressions
- Conversion of a prefix expression to postfix expression
- Conversion of a infix expression to postfix expression

Topic 1 : Problems related to Stacks

Q1. Find the minimum number of brackets that we need to remove to make the given bracket sequence balanced.

Input

)()(

Output

2

Input

()(())

Output

0

Code

Explanation

To make the string valid with minimum removals, we need to get rid of all parentheses that do not have a matching pair.

- Traverse through the input string.
- **At each iteration-**
  a. If the current character is an opening bracket, push it into the stack.
  b. If the current character is a closing bracket, see if there exist a opening bracket in the stack that corresponds to this bracket-
    i. If the stack is not empty and the top of the stack is an opening bracket, remove it.
    ii. Else we have not found a matching pair so add the closing bracket into the stack.
- At this point the stack contains all the unresolved brackets. Return its size.

Q2. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time. Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.

- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

Assumption: Methods pop, top and getMin operations will always be called on non-empty stacks.

Input

["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Code

Explanation

You just need a stack in which each element will contain 2 values at any given time. One will represent the top of the stack and the second min element so far in the stack.

Topic 2: Understanding infix, postfix and prefix expressions

Infix: This is the type of notation where we write the operation in between the 2 operands. In our daily life, we use this kind of notation. We use "BODMAS" to evaluate such expressions.
For example:
A+B
A+B-C

Prefix: This is the type of notation where we write the operation before the 2 operands. This notation is also known as "Polish notation".
For example:
+AB
-+ABC

Postfix: This is the type of notation where we write the operation after the 2 operands. This notation is also known as "Reverse Polish notation".
For example:
AB+
AB+C-

Note: In postfix and prefix expressions whichever operator comes before will be evaluated first, irrespective of its priority. Also, there are no brackets in these expressions. As long as we can guarantee that a valid prefix or postfix expression is used, it can be evaluated with correctness.

Topic 3: Evaluation of an infix expressions

Steps

- Create 2 stacks: one for the operands and the other for the operators.

- If the character is an operand, push it into the operand stack.
- If the character is an opening bracket, push it into the operator stack.
- If the character is a closing bracket-
  a. Loop till you find the corresponding opening bracket in the operator stack.
  b. At each iteration, take out the top operation from the operator stack and apply it on the top 2 values of the operand stack.
- **Else -**
  a. Loop till the operation on the top of the operator stack has its precedence greater than or equal to the current operator.
  b. At each iteration, take out the top operator from the operator stack and apply it on the top 2 values of the operand stack.
- **After the traversal-**
  a. Loop till you have operators left to perform.
  b. At each iteration, take out the top operator from the operator stack and apply it on the top 2 values of the operand stack.
- Return the top value from the operand stack.

Code

Topic 4: Evaluation of a postfix expressions

Steps

- Create a stack for the operands and traverse the expression:
  a. If the character is an operand, push it into the operand stack.
  b. Else, take out the top 2 values of the operand stack and apply the current operator on them. Push the result into the operand stack.
- Return the top value from the operand stack.

Code

Topic 5: Evaluation of a prefix expressions

Steps

- Create a stack for the operands and traverse the expression in reverse order.
  a. If the character is an operand, push it into the operand stack.
  b. Else, take out the top 2 values of the operand stack and apply the current operator on them. Push the result into the operand stack.
- Return the top value from the operand stack.

Code

Topic 6: Conversion of a prefix expression to postfix expression

Steps

- Reverse the given expression.
- Create a stack.

- Traverse through the expression.
  a. If the character is an operand, push it into the operand stack.
  b. Else take out the top 2 expressions from the stack and make the postfix expression of the form- expression1 + expression2 + current_operator. Push the new expression into the stack.
- Return the top value of the stack.

Code

Topic 7: Conversion of an infix expression to a postfix expression

Steps

- Create a stack to store the operators of the infix expression and traverse through the expression.
  a. If the character is a digit, add it to the postfix expression.
  b. If the character is an opening bracket, add it to the stack.
  c. If the character is a closing bracket, add the top character of the stack into the postfix expression till the corresponding opening bracket is encountered
  d. Else loop till the operator on top of operation stack has a higher or equal precedence to the current operator. At each iteration add the top character of the stack into the postfix expression.
- Add the remaining operators of the stack to the postfix expression

Code