Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
    - Stack
    - The Prutor Editor
    - Finding your identity
    - Queue
    - The Prutor Editor Part II
    - Only Ones
    - Graphs
    - How Mr C actually does Math
    - The Hidden Positives and Negatives
    - How Prutor Manages Memory
    - Message in the Matrix
    - The Hidden Key
- PRACTICE-11_FUN-PTR
- LAB-PRAC-11_FUN-PTR
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC

How Mr C actually does Math

LAB-PRAC-10_MAT-FUN

How Mr C actually does Math [20 marks]

Problem Statement
When you ask Mr C to evaluate a certain math expression with lots and lots of brackets and operators, he actually does not evaluate that expression as is. He first converts that expression into a format known as the *postfix* format and then evaluates the expression. An example of a postfix expression is

2 3 + 7 * #

The above expression is actually equivalent to the usual expression (2 + 3) * 7 and both expressions evaluate to 35. If you are wondering how the expression 2 + (3 * 7) (which is just 2 + 3 * 7 due to BODMAS rules) is represented in postfix, it is represented as 2 3 7 * +. The rules on how bracketed expressions are converted to postfix expressions will be taught to you in a more advanced course like ESO207 or CS345. However, below are given the rules of how postfix expressions are evaluated. Note that postfix expressions we looked at do not have any brackets. This is true of all postfix expressions - none of them need brackets.

Your post fix expression will be given in a single line in the input and will only consist of single digit non-negative integers, the characters + (addition), * (multiplication), and the termination symbol #. There will be a single space between every two characters in the input. You have to maintain an integer array of length 100 and follow the rules given below

1. Initially the array will be empty
2. If you encounter a digit symbol, simply insert that integer in the next available position in the array. Number of elements stored in the array goes up by one). Print the entire array on a single line of your output with a single space between two elements of the array.
3. If you encounter a math operator symbol i.e. + or *, take the integers in the last two positions in the array, erase them from the array (number of elements stored in the array down by two), perform the addition/multiplication operation with those two numbers, and then insert the result back into the next available position in the array (number of elements in the array goes up by one). Print the entire array on a single line of your output with a single space between two elements of the array. However, if there are less than 2 elements in the array to begin with, print "ERROR" and do not change the array at all.
4. If you encounter the # character, if there is exactly one element in the array, print that element. If there are no elements in the array or else if there are more than one elements left in the array, print "ERROR" (without quotes).

The postfix format may seem a bit funny to you but it comes with a huge advantage that no brackets are necessary in math expression which are in postfix format. You may also find it interesting that some languages like Sanskrit also operate in postfix format at times. The sentence "Ram and Sita went to the garden" translates to "Ram Sita cha udyaanam gachchatah". Note that the "and" operator appears in postfix notation in Sanskrit!

Caution

1. All your error messages or array element prints should be on separate lines. There should be no trailing spaces at the end of any line.
2. There will be no spaces after the # character. However, there will be a single space after every other character in the expression.
3. The results of the computations you do (addition/multiplication) may be numbers with more than one digit. However, the input will only contain single digit non-negative numbers.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

HINTS:

1. Use a variable to keep track of how many elements are present in the array at any given point of time. Although your array is of size 100, most of the time you will have less than 100 elements in the array.

EXAMPLE 1:
INPUT
```
2 3 + 7 * #
```

OUTPUT:
```
2
2 3
5
5 7
35
35
```

Explanation Let us look at the characters and the state of the array after each operation

1. Initially array is empty []
2. Character 2 : Put 2 at index 0 since that is the first empty location. Array is now [2]
3. Character 3 : Put 3 at index 1 since that is the next empty location. Array is now [2 3]
4. Character + : Take out the last two elements in the array - array is now empty []. Put the sum of those two elements into the next available position in the array which is index 0 since array is now empty - array is now [5]
5. Character 7 : Put 7 at index 1 since that is the next empty location. Array is now [5 7]
6. Character * : Take out the last two elements in the array - array is now empty []. Put the product of those two elements into the next available position in the array which is index 0 since array is now empty - array is now [35]
7. Character # : There is only one element in the array so things are nice. Print that element 35 and exit

EXAMPLE 2:
INPUT
```
#
```

OUTPUT:

ERROR

EXAMPLE 3:
INPUT
1 5 + 3 #

OUTPUT:
1
1 5
6
6 3
ERROR

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

Start Solving! (/editor/practice/6204)