**Print Elements of a Matrix in Spiral Order**

Write a program that reads an **MxN** matrix **A** and prints its elements in spiral order.

You should start from the element in the 0th row and 0th column in the matrix and proceed in a spiral order as shown below.

1→ 2 → 3 → 4

                      ↓

5 → 6 → 78

↑             ↓↓

9    10←11 12

↑              ↓

13←14←15←16

Output for the above matrix: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

**INPUT:**

First line contains two integers **M** and **N** separated by whitespace. The next **M** lines contain elements of matrix A, starting with the topmost row. Within each row, the elements are given from left to right.

**OUTPUT:**

Elements of the matrix printed in a spiral order. All the elements should be separated by whitespace.

**CONSTRAINTS:**

1 <= M <= 5, 1 <= N <= 5.

Elements in the matrix will be in the range [-100,100]

Sample Test Cases

|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| Test Case 1 | 3 3  0 1 2  7 8 3  6 5 4 | 0 1 2 3 4 5 6 7 8 |
| Test Case 2 | 2 2  1 10  3 5 | 1 10 5 3 |
| Test Case 3 | 4 4  7 8 1 2  3 4 3 1  1 2 8 6  5 4 9 7 | 7 8 1 2 1 6 7 9 4 5 1 3 4 3 8 2 |
| Test Case 4 | 1 1  20 | 20 |
| Test Case 5 | 3 3  1 2 3  4 5 6  7 8 9 | 1 2 3 6 9 8 7 4 5 |
| Test Case 6 | 2 4  1 2 3 4  5 6 7 8 | 1 2 3 4 8 7 6 5 |
| Test Case 7 | 3 4  1 2 3 4  5 6 7 8  9 10 11 12 | 1 2 3 4 8 12 11 10 9 5 6 7 |
| Test Case 8 | 1 2  1 20 | 1 20 |
| Test Case 9 | 2 2  1 -1  2 3 | 1 -1 3 2 |
| Test Case 10 | 3 1  1 2 3 | 1 2 3 |
| Test Case 11 | 1 4  1  2  3  4 | 1 2 3 4 |
| Test Case 12 | 5 5  1 2 3 4 5  6 7 8 9 1  11 12 13 14 15  16 17 18 19 20  21 22 23 24 25 | 1 2 3 4 5 1 15 20 25 24 23 22 21 16 11 6 7 8 9 14 19 18 17 12 13 |

**Find whether two given strings are permutations of each other**

**Due on 2016-02-10, 23:55 IST**

Write a program to find whether two given strings are permutations of each other.  A string str1 is a permutation of str2 if all the characters in str1 appear the same number of times in str2 and str2 is of the same length as str1.

**Input:** Two strings S1 and S2

**Output:**   
yes - if they satisfy given criteria

no - otherwise

**Constraints:**

1 <= len(S1), len(S2) <= 100.

Characters from ASCII range 0 to 127.

White space will not be given in the string.

Sample Test Cases

|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| Test Case 1 | india  daini | yes |
| Test Case 2 | hellobye  hellobye! | no |
| Test Case 3 | iloveindia  loveindiai | yes |
| Test Case 4 | 12434  43214 | yes |
| Test Case 5 | aaa  aa | no |
| Test Case 6 | iitmadras.  madras.iit | yes |
| Test Case 7 | nptelisbest  ptenlisestb | yes |
| Test Case 8 | abcdefg  aabbccddeeffgg | no |
| Test Case 9 | 12345  1122334455 | no |
| Test Case 10 | #$%&  &%$# | yes |
| Test Case 11 | (abc)  (xyz) | no |
| Test Case 12 | "hellobye"  "byehello" | yes |

**Find the depth of letters in a string**

**Due on 2016-02-10, 23:55 IST**

A string is given which has letters from English alphabet and parentheses. The depth of a letter is defined as the number of balanced parentheses it is surrounded by. Write a  C program to find the depth of each letter in the input string.  
  
**Explanation:**  
  
(a(b)((cd)e)f)g  
  
g is at depth 0  
a and f are at depth 1  
b and e are at depth 2  
c and d are at depth 3  
  
**Input Constraints:**  
  
1) Input string can be of length 1 to 100.  
2) The input will have only ‘(‘ , ‘)’ and letters from English alphabet.  
3) There will be no repetition of letters.  
4) Only lowercase letters are used.  
5) The letters can be in any sequence.     
6) The parentheses are always well balanced. Every '(' has a matching ')' that follows it later in the string. Every ')' has a matching '(' before it in the input string.  
7) Notice that there are no space in the string.  
  
**Input:** An array of characters  
**Output:**  
1) The depth of each letter separated by a space.  
2) The order of the depth of the letters should be the same  order in which the letters appear in the input.  
3)To mark the end of the output it should end with a space and a ‘#’ character.  
  
**Example 1:**  
Input: (a(b)((cd)e)f)g  
Output: 1 2 3 3 2 1 0 #  
  
**Example 2:**  
Input: p(r(q))(s)  
Output: 0 1 2 1 #  
  
In this example, letters are appearing in the order p followed by r followed by q and s. They have depth of 0, 1, 2 and 1 respectively. Note that the depth is not printed in the order p,q,r,s (the alphabetical order) but p,r,q,s (the order in which they appear in the input string).

Sample Test Cases

Sample Test Cases

|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| Test Case 1 | (a(b)((cd)e)f)g | 1 2 3 3 2 1 0 # |
| Test Case 2 | a(b(c))(d) | 0 1 2 1 # |
| Test Case 3 | a(b(c))(d(fe)) | 0 1 2 1 2 2 # |
| Test Case 4 | a(b(c))(d(f()e)) | 0 1 2 1 2 2 # |
| Test Case 5 | () | # |
| Test Case 6 | ab()(c(d(e(f)()(g)h))) | 0 0 1 2 3 4 4 3 # |
| Test Case 7 | ((((a))b))cdegfhi(jklmnop) | 4 2 0 0 0 0 0 0 0 1 1 1 1 1 1 1 # |
| Test Case 8 | ((a))((b(c)((d(e(f(g(h(i(j(k(l)))))))))))) | 2 2 3 4 5 6 7 8 9 10 11 12 # |

**Largest sum of all contiguous subarrays**

Write an efficient C program to find the largest sum of contiguous subarray within an one-dimensional array of integers. A contiguous subarray of an array is defined as the sequence of elements that are in any continuous set of indices that are valid within an array.  
  
Lets take an example of an array {5,-3, 4}. Possible contiguous subarray combinations are {5}, {-3}, {4}, {5,-3}, {-3,4} and {5,-3,4}. Note that {5,4} is not a valid subarray as the indices of 5 and 4 are not continuous.

The contiguous subarray  {5,-3,4} has the largest sum 6.   
  
**Input Constraints**:

First line : array size (N), where 1<= N<=100

Second line : N integers separated by spaces

where each number Ni satisfies

-10000 <= Ni <=10000

**Output Constraints**:

Single integer SUM which is the largest sum of all possible contiguous subarrays.

Sample Test Cases

|  |  |  |
| --- | --- | --- |
|  | **Input** | **Output** |
| Test Case 1 | 3  5 -3 4 | 6 |
| Test Case 2 | 4  1 1 1 1 | 4 |
| Test Case 3 | 8  -2 -3 4 -1 -2 1 5 -3 | 7 |
| Test Case 4 | 5  -5 -2 4 5 1 | 10 |
| Test Case 5 | 7  -1 -2 -3 -4 -5 -6 -7 | -1 |
| Test Case 6 | 2  2000 2000 | 4000 |
| Test Case 7 | 10  -3 -4 1 2 3 -1 2 5 -8 8 | 12 |
| Test Case 8 | 5  -10000 -10000 -10000 -10000 -10000 | -10000 |
| Test Case 9 | 6  -1 2 -2 4 -3 6 | 7 |
| Test Case 10 | 6  1 -2 2 -4 3 -6 | 3 |
| Test Case 11 | 9  0 -1 0 0 0 0 -1 0 0 | 0 |
| Test Case 12 | 20  1 1 1 1 1 1 1 1 1 1 -2 -3 1 1 1 1 -1 1 1 1 | 11 |
| Test Case 13 | 1  5 | 5 |
| Test Case 14 | 2  0 -1 | 0 |