# Exercises: Multidimensional Arrays

Problems for exercises and homework for the ["CSharp Advanced" course @ Software University](https://softuni.bg/courses/csharp-advanced).

You can check your solutions here: <https://judge.softuni.bg/Contests/573/Multidimensional-Arrays-Exercise>

## Matrix of Palindromes

Write a program to generate and print the following **matrix of palindromes** of **3** letters with **r** rows and **c** columns like at the examples below.

* **Rows** define the first and the last letter: row 0 🡪 ‘a’, row 1 🡪 ‘b’, row 2 🡪 ‘c’, …
* **Columns + rows** define the middle letter:
  + column 0, row 0 🡪 ‘a’, column 1, row 0 🡪 ‘b’, column 2, row 0 🡪 ‘c’, …
  + column 0, row 1 🡪 ‘b’, column 1, row 1 🡪 ‘c’, column 2, row 1 🡪 ‘d’, …

### Input

* On the **first line**, you are given the integers **r** and **c**, separated by a space.

### Output

* On **r** \***c** number of lines, print the matrix of palindromes as shown in the example.

### Constraints

* Constraints: r and c are integers in the range [1…26]; r + c ≤ 27.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 4 6 | aaa aba aca ada aea afa  bbb bcb bdb beb bfb bgb  ccc cdc cec cfc cgc chc  ddd ded dfd dgd dhd did | 3 2 | aaa aba  bbb bcb  ccc cdc |

### Hints

* char[] alphabet = "**abcdefghijklmnopqrstuvwxyz**".ToCharArray();

## Diagonal Difference

Write a program that finds the **difference between the sums of the square matrix diagonals** (absolute value).



### Input

* On the **first line**, you are given the integer **N** – the size of the square matrix
* The next N **lines** holds the values for **every row** – N numbers separated by a space

### Output

* Print **the absolute** difference between **the sums** of the primary and the secondary diagonal

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3  11 2 4  4 5 6  10 8 -12 | 15 | **Primary diagonal:** sum = 11 + 5 + (-12) = 4  **Secondary diagonal:** sum = 4 + 5 + 10 = 19  **Difference:** |4 - 19| = 15 |

## 2x2 Squares in Matrix

Find the count of **2 x 2 squares of equal chars** in a matrix.

### Input

* On the **first line**, you are given the integers **rows** and **cols –** the matrix’s dimensions
* Matrix characters come at the next **rows** lines (space separated)

### Output

* Print the number of all the squares matrixes you have found

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 3 4  A B B D  E B B B  I J B B | 2 | Two 2 x 2 squares of equal cells:  A **B B** D A B B D  E **B B** B E B **B B**  I J B B I J **B B** |
| 2 2  a b  c d | 0 | No 2 x 2 squares of equal cells exist. |

## Maximal Sum

Write a program that reads a rectangular integer matrix of size **N x M** and finds in it the square **3 x 3** that **has maximal sum of its elements**.

### Input

* On the first line, you will receive the rows **N** and columns **M**. On the next **N lines** you will receive **each row with its columns**

### Output

* Print the **elements** of the 3 x 3 square as a matrix, along with their **sum**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Matrix** | **Output** |
| 4 5  1 5 5 2 4  2 1 4 14 3  3 7 11 2 8  4 8 12 16 4 |  | Sum = 75  1 4 14  7 11 2  8 12 16 |

## Rubik’s Matrix

Rubik’s cube – everyone’s favorite head-scratcher. Writing a program to solve it will be quite a difficult task for an exam, though. Instead, we have a Rubik’s matrix prepared for you. You will be given a pair of dimensions: **R** and **C.** To prepare the matrix, fill each row with increasing integers, starting from one. For example, **2 x 4** matrix must look like this:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |

Next, you will receive series of commands, indicating which row or column you must move, in which direction, and how many times. For example, **1 up 1** means: column 1, direction: up, 1 move. After executing it, the matrix should look like:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 6 | 3 | 4 |
| 5 | 2 | 7 | 8 |

Directions **left** and **right** means you must move the corresponding **row**, while **up** and **down** are related to the **columns.** After shuffling the Rubik’s matrix, you have to **rearrange** it (meaning that the **values in each cell** should be in **increasing order**, such as the ones in the original matrix). The rearranging process should start at **top-left** and end at **bottom-right**. Find the **position** of the value you need, and print the **swap** **command** on the console, in the format described below.

### Input

* On the first line, you are given the integers **R** and **C**, separated by a space
* On the second line, you are given an integer **N**, indicating the number of commands to follow
* On the next N lines, you are given commands in format **{row/col} {direction} {moves}**

### Output

* On **R** \* **C** number of lines, print the **swap commands** needed to rearrange the matrix, either:
  + **“Swap ({row}, {col}) with ({row}, {col})”** or
  + **“No swap required”**

### Constraints

* **R, C, N** are integers in range [1 … 100]
* {**row**} and {**col**} will always be inside the matrix
* {**moves**} is in range [0 … 231-1]
* Allowed time and memory: 0.25s / 16 MB

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| 3 3  2  1 down 1  1 left 1 | No swap required  Swap (0, 1) with (1, 0)  No swap required  Swap (1, 0) with (1, 2)  Swap (1, 1) with (2, 1)  Swap (1, 2) with (2, 1)  No swap required  No swap required  No swap required | 3 3  2  0 down 3  0 left 3 | No swap required  No swap required  No swap required  No swap required  No swap required  No swap required  No swap required  No swap required  No swap required |

## Target Practice

Cotton-eye Gosho has a problem. **Snakes**! An infestation of snakes! Gosho is a red-neck which means he doesn’t really care about animal rights, so he bought some ammo, loaded his shotgun and started shooting at the poor creatures. He has a favorite spot – rectangular stairs which the snakes like to climb in order to reach Gosho’s stash of whiskey in the attic. You’re tasked with the horrible cleanup of the aftermath.

A **snake** is represented by **a string**. The **stairs** are a **rectangular matrix of size NxM**. A snake starts climbing the stairs from the **bottom-right corner** and slithers its way up in a **zigzag** – first it moves left until it reaches the left wall, it climbs on the next row and starts moving right, then on the third row, moving left again and so on. The first cell (bottom-right corner) is filled with the first symbol of the snake, the second cell (to the left of the first) is filled with the second symbol, etc. The snake is as long as it takes in order to **fill the stairs completely** – if you reach the end of the string representing the snake, start again at the beginning. Gosho is patient and a sadist, he’ll wait until the stairs are completely covered with snakes and then will fire a shot.

The shot has three parameters – **impact row, impact column and radius**. When the projectile lands on the specified coordinates in the matrix it **destroys all symbols in the given radius (turns them into a space)**. You can check whether a cell is inside the blast radius using the Pythagorean Theorem.

The symbols above the impact area start **falling down until they land on other symbols (meaning a symbol moves down a row as long as there is a space below)**. When the horror ends, print on the console the **resulting staircase, each row on a new line**. You should really check out the examples at this point.

### Input

* The input data should be read from the console. It consists of exactly three lines
* On the first line, you’ll receive the **dimensions** of the stairs in format: **"N M"**, where **N** is the number of **rows**, and **M** is the number of **columns**. They’ll be separated by a single space
* On the second line you’ll receive the string representing the **snake**
* On the third line, you’ll receive the **shot parameters (impact row, impact column and radius)**, all separated by a **single space**
* The input data will always be valid and in the format described. There is no need to check it explicitly

### Output

* The output should be printed on the console. It should consist of **N lines**
* Each line should contain a string representing the respective row of the final matrix

### Constraints

* The **dimensions** N and M of the matrix will be integers in the range [1 … 12]
* The **snake** will be a string with length in the range [1 … 20] and **will not contain any whitespace characters**
* The shot’s **impact row and column** will always be **valid coordinates** in the matrix – they will be integers in the range [0 … N – 1] and [0 … M – 1] respectively
* The shot’s **radius** will be an integer in the range [0 … 4]
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5 6  SoftUni  2 3 2 | O  US t  tn f  iSi UU  nUt oS | The matrix has 5 rows and 6 columns. Fill it in the described pattern:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | o | S | i | n | U | t | | U | n | i | S | o | f | | t | f | o | S | i | n | | i | S | o | f | t | U | | n | U | t | f | o | S |   The shot lands on cell (2,3). It has a radius of 2 cells. The impact cell is shaded black and the other cells within the shot radius are shaded grey.   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | o | S | i | n | U | t | | U | n | i | S | o | f | | t | f | o | S | i | n | | i | S | o | f | t | U | | n | U | t | f | o | S |   Replace all characters in the blast area with a space:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | o | S | i |  | U | t | | U | n |  |  |  | f | | t |  |  |  |  |  | | i | S |  |  |  | U | | n | U | t |  | o | S |   All characters start falling down until they land on other characters:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | o | S | i |  | U | t | | U | n |  |  |  | f | | t |  |  |  |  |  | | i | S |  |  |  | U | | n | U | t |  | o | S |   The resulting matrix is:   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | o |  |  |  |  |  | | U | S |  |  |  | t | | t | n |  |  |  | f | | i | S | i |  | U | U | | n | U | t |  | o | S | |