**Exercises: Multidimensional Arrays**

This document defines the exercises for ["Java Advanced" course @ Software University](https://softuni.bg/modules/59/java-advanced). Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/Contests/1460/Multidimensional-Arrays-Exercises).

* **Fill the Matrix**

Filling a matrix in the regular way (**top to bottom** and **left to right**) is boring. Write two **methods** that **fill** a **matrix** of size **N x N** in **two** different **patterns.** Both patterns are described below:

|  |  |
| --- | --- |
| **Pattern A** | **Pattern B** |
|  |  |

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3, A | 1 4 7  2 5 8  3 6 9 |
| 3, B | 1 6 7  2 5 8  3 4 9 |

**Hints**

* Make a different method for each pattern.
* Make a method for printing the matrix.
* **Matrix of Palindromes**

Write a program to generate the following **matrix of palindromes** of **3** letters with **r** rows and **c** columns like the one in 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**

* The numbers **r** and **c** stay at the first line at the input.
* **r** and **c** are integers in the range **[1…26]**.
* **r** **+** **c** ≤ 27

**Examples**

|  |  |
| --- | --- |
| **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**

* Use two nested loops to generate the matrix.
* Print the matrix row by row in a loop.
* Don’t forget to pack everything in methods.
* **Diagonal Difference**

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



**Input**

* The **first line** holds a number **n** – the size of matrix.
* The next **n lines** hold the **values for every row** – **n** numbers separated by a space.

**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 |
| 4  -7 14 9 -20  3 4 9 21  -14 6 8 44  30 9 7 -14 | 34 |  |

**Hints**

* Use a **single** loop **i** = **[1** **…** **n]** to sum the diagonals.
* The **primary diagonal** holds all cells {**row**, **col**} where **row** == **col** == **i**.
* The **secondary diagonal** holds all cells {**row**, **col**} where **row** == **i** and **col** == **n-1-i**.
* **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 a 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 elements**.

Print the **elements** of the 3 x 3 square as a matrix, along with their **sum**. See the format of the output below.

**Examples**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 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 |  |
| 5 6  1 0 4 3 1 1  1 3 1 3 0 4  6 4 1 2 5 6  2 2 1 5 4 1  3 3 3 6 0 5 | Sum = 34  2 5 6  5 4 1  6 0 5 |  |

* **Matrix Shuffling**

Write a program, which reads a string matrix from the console and performs certain operations with its elements. User input is provided in a similar way as in the problems above – first, you read the **dimensions** and then the **data**.

Your program should then receive commands in the format: "**swap row1 col1 row2c col2**" where row1, row2, col1, col2 are **coordinates** in the matrix. For a command to be valid, it should start with the "**swap**" keyword along with **four valid coordinates** (no more, no less). You should **swap the values** at the given coordinates (cell [row1, col1] with cell [row2, col2]) **and print the matrix at each step** (this you'll be able to check if the operation was performed correctly).

If the **command is not valid** (doesn't contain the keyword "**swap**", has fewer or more coordinates entered or the given coordinates do not exist), print "**Invalid input!**" and move on to the next command. Your program should finish when the string "**END**" is entered.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 3  1 2 3  4 5 6  swap 0 0 1 1  swap 10 9 8 7  swap 0 1 1 0  END | 5 2 3  4 1 6  Invalid input!  5 4 3  2 1 6 |
| 1 2  Hello World  0 0 0 1  swap 0 0 0 1  swap 0 1 0 0  END | Invalid input!  World Hello  Hello World |

**Hints**

* Think about **Exception Handling**.