CAMBRIDGE INTERNATIONAL UNIVERSITY

COURSE OF COMPUTER SCIENCE
LABORATORY PRACTICE n. 7

Exercise 1 (1 points):

Write a Python program in order to:

- read an `R×C` matrix of integer values (`R` and `C` being predefined constants)
- print the same matrix with the sum of all elements of each row shown on the right and the sum of all values of each column displayed at the bottom.

Example: let `R` = 3 and `C` = 4. Then, assuming that the matrix specified by the used is the one reported below on the left, the output produced by the program is given on the right:

```
4  3  1  2
1  7  2  2
3  3  5  0
```

```
 4  3  1  2   10
 1  7  2  2   12
 3  3  5  0   11

 8 13  8  4
```

Exercise 2 (2 points):

Write a Python program which, once an $n \times n$ matrix of real values has been read in, generates a new matrix with size $(2n-1) \times n$, in which the $n-1$ additional rows are obtained by interpolating the original ones. More specifically, between any two consecutive rows of the loaded matrix, a new row must be inserted, whose elements are given by the arithmetic average of the corresponding values immediately above and below them.
The matrix obtained in this way should be printed on screen. Assume that the maxim value for n is equal to 50.

Example: given the matrix:

```
2.6  4.4  5.0
4.8  3.4  7.2
2.0  2.6  3.8
```

the matrix to be computed and displayed is then:

```
2.6  4.4  5.0
3.7  3.9  6.1
4.8  3.4  7.2
2.4  3.0  5.5
2.0  2.6  3.8
```

Exercise 3 (2 points):

Write down a Python program able to:

- declare two matrices of integer values named `m1` and `m2`, respectively, with size equal to `MAX` (`MAX` being a predefined constant).
- read the actual size (number of rows and columns) for both `m1` (r_1 and c_1) and `m2` (r_2 and c_2), verifying that such values allow to compute the product `m1·m2`.
- load from the keyboard the elements for both the matrices.
- compute the product matrix `m3`: `m3` = `m1·m2`
- display the computed matrix

Recall the product matrix is composted by r_1 rows and c_2 columns, and that the value of every its element in position `[i][j]` (denoted as $m3_{ij}$) is given by:

$$m3_{ij} = \sum_k m1_{ik} \cdot m2_{kj}$$

Exercise 4 (2 points):

Write down a Python program for:

- reading the dimensions *nr* and *nc* of a matrix `M` of integers (at most 20 for each side), then loading such a matrix.
- asking the user for an integer number *n*.
- looking for all the occurrences of *n* inside `M`. Every time such an occurrence is detected, the program should print the position of the matrix element (row and column) and all the sub-matrices obtained from `M` by removing the row and the column in which the occurrence has been found.

Example: let *nr* = *nc* = 4, *n* = 5 and the input matrix be:

```
1 -2  0  3
3  1  5  4
7  2  3  1
4  6 -1  5
```

Then, the program must produce the following output messages:

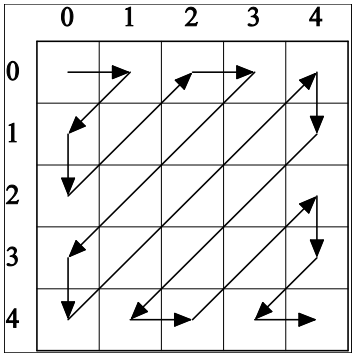
```
Row = 1, Column = 2
Top-left sub-matrix:
 1 -2
Top-right sub-matrix:
 3
Bottom-left sub-matrix:
 7  2
 4  6
Bottom-right sub-matrix:
 1
 5
Row = 3, Column = 3
Top-left sub-matrix:
 1 -2  0
 3  1  5
 7  2  3
```

Exercise 5 (2 points):

Write a Python program which, given an integer number *n*, displays the sequence of cells (their indices) of an $n \times n$ square in the order resulting from the application of the following algorithm:

1. start from position (0, 0)
2. move one step on the right if possible, otherwise move one step below
3. move all the possible steps in diagonal, bottom-left direction
4. move one step below if possible, otherwise move one step on the right
5. move all the possible steps in diagonal, top-right direction
6. if not all the cells have been visited, go back to point 2

The following figure graphically shows the application of this algorithm (and a possible program output) for the case *n* = 5.




```
(0,0) (0,1) (1,0) (2,0) (1,1)
(0,2) (0,3) (1,2) (2,1) (3,0)
(4,0) (3,1) (2,2) (1,3) (0,4)
(1,4) (2,3) (3,2) (4,1) (4,2)
(3,3) (2,4) (3,4) (4,3) (4,4)
```