CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

BIG DATA PROCESSING.

Lab01: Sequential-solving Programming Exercise: Minesweeper

BACKGROUND.

The Irish Collegiate Programming Contest (IrlCPC) is a programming competition organised by the UCC's ACM Student Chapter and sponsored by Google, Microsoft, Workday, The Insight Centre for Data Analytics and the School of Computer Science and Information Technology (UCC).

This contest is open to student teams with 2-3 members from Irish third level institutions and will test the participating teams on their combined knowledge of algorithms, programming and problem solving abilities.

For more information in the event:
https://www.ucc.ie/en/compsci/events/irish-collegiate-programming-competition-irlcpc.html

In this first lab we are going to solve one of the problems proposed in the 2017 edition. The problem was proposed by my former colleague Milan de Cauwer, is based on the classical game 'Minesweeper' and is presented in the next section.

You can see the whole set of problems proposed on that edition in the following link:
http://multimedia.ucc.ie/Public/training/cycle2/IrlCPC-ProblemSet2017.pdf

1 Corcaigh-xit

The day has arrived for the people of the Republic of Cork to declare their independence. Politicians have decided that the border of the county should be protected from the rest of Ireland. Because neighbouring counties stubbornly refuse to pay for the construction of a wall, a decision has been made to plant mines along the county border. To implement this, local geographers have marked rectangular zones of height N and width M in which they defined cells of unit size. Each of these $N \times M$ cells may contain a landmine. They carefully recorded whether or not cells contained a mine and the total number of mines in the zone.

Although the location of the mines must remain secret, local authorities want to create maps from which their location can be inferred. Dónal, a UCC student, was asked to produce these maps so that each cell contains the number of mines buried in the vicinity of that cell or the character 'x' if the cell itself is mined. Each cell has between 3 and 8 neighbours, depending on its location in the grid.

Not Cork

o	x	o	x	o
o	o	o	x	x
o	o	o	o	o

Cork

Not Cork

1	x	3	x	3
1	1	3	x	x
0	0	1	2	2

Cork

Original maps on the left. On the right, the expected completed map.

Can you help Dónal to produce these maps ?

Input The first line contains two integers N and M, $1 \leq N, M \leq 1000$, corresponding to the the height and width of the map respecitvely. The next N lines contain M space-separated characters. Each character is either an *x* if this cell contains a mine, or *o* to represent an empty cell.

Output The output consists of N lines of M space-separated characters. Each character either encodes for the number of adjacent cells containing mines, or the character 'x' if the cell itself contains a mine.

Sample Input 1

```
3 5
o x o x o
o o o x x
o o o o o
```

Sample Output 1

```
1 x 3 x 3
1 1 3 x x
0 0 1 2 2
```

Sample Input 2

```
5 5
o o o o o
o o x o o
o o x o o
o o x o o
o o o o o
```

Sample Output 2

```
0 1 1 1 0
0 2 x 2 0
0 3 x 3 0
0 2 x 2 0
0 1 1 1 0
```

LAB EXERCISE.

The folder Lab01 contains 3 sub-folders:

- code:
  - Contains the Python file **my_main.py** to be completed in the lab exercise.
- input_files:
  - Contains the input file with the instance to be passed to the program my_main.py. The folder contains two instance examples: **input_1.txt** and **input_2.txt**, but you can generate your own instance if you wish.
- results:
  - Contains the output file with the instance solution generated by the program my_main.py. The folder contains an instance solution example: **output.txt**.

Complete the following functions in the Python file **my_main.py** to come up with a solution for the problem being proposed.

- parse_in
  - The function receives as input the name of the file to be parsed (e.g., “input_1.txt”) and returns as output the tuple *(num_rows, num_columns, matrix)*. For example, if input_1.txt is the instance used, then it returns:
    - num_rows => 3
    - num_columns => 5
    - matrix => [['o', 'x', 'o', 'x', 'o'], ['o', 'o', 'o', 'x', 'x'], ['o', 'o', 'o', 'o', 'o']]

2. solve
   - The function receives as input the tuple *(num_rows, num_columns, matrix)* and returns as output the result matrix *sol_matrix*.

     For example, if input_1.txt is the instance used, then it returns:
     - sol_matrix => [['1', 'x', '3', 'x', '3'], ['1', '1', '3', 'x', 'x'], ['0', '0', '1', '2', '2']]

3. parse_out
   - The function receives as input the tuple *(num_rows, num_columns, sol_matrix)* and generates the output file (e.g., output.txt).