**Final Project**
**Cellular Simulation**

Develop the Python source code required to solve the following problem.

## Problem

You are tasked with creating a program capable of executing the first 100 steps of a modified cellular life simulator. This simulator will receive the path to the input file as an argument containing the starting cellular matrix. The program must then simulate the next 100 time-steps based on the algorithm discussed on the next few pages. The simulation is guided by a handful of simplistic rules that will result in a seemingly complex simulation of cellular organisms.

Below is an example starting cellular matrix consisting of 10 rows and 20 columns:

```
.....O..O.O....O....
...OO.....O.....O...
.......OO...O..OO...
O........OOOO...OOOO
O....OO..O..OO..OOOO
O......O....O..O...O
O.............OO...O
O..........O..O....O
.O..............O...
....O.OO.......O...O
```

# Directions

Develop a program capable of accepting the following command line arguments:

- -i <path_to_input_file>
  - Purpose: This option retrieves the file path to the starting cellular matrix.
  - Input Type: String
  - Validation: Entire file path must exist, otherwise error.
  - Required: Yes
- -o <path_to_output_file>
  - Purpose: This option retrieves the file path for the final output file.
  - Input Type: String
  - Validation: The directories in the file path must exist, otherwise error.
  - Required: Yes
- -t <int>
  - Purpose: This option retrieves the number of threads to spawn.
  - Input Type: Unsigned Integer
  - Validation: Must be a positive integer > 0, otherwise error.
  - Required: No (Default Value of 1 if not provided)

### *Example executions:*

- *python3 Eric_Rees_R123456_final_project.py -i inputFile.txt -o timeStep100.txt -t 36*
  - Sets input file to "inputFile.txt"
  - Sets output file to "timeStep100.txt"
  - Sets thread count to 36
- *python3 Eric_Rees_R123456_final_project.py -i myInput.txt -o myOutput.txt*
  - Sets input file to "myInput.txt"
  - Sets output file to "myOutput.txt"
  - Sets thread count to 1 (default when not specified)

## Input/Output Files

Valid input and output files must abide by the following rules:

1) The matrix may only contain the following symbols:
   a. Periods '.' to signify currently "dead" cells.
   b. Capital O's to signify currently "alive" cells.
   c. End of Line Characters marking the end of each row.
2) The matrix may not contain any spaces, commas, or other delimiters between symbols.
3) The matrix must separate rows with a line break.
4) Files containing any other symbols beyond periods, capital O's, and newline characters are considered invalid.

## Processing the Matrix

Using this starting cellular matrix, your program should then simulate the next 100 steps of a simulation that uses the following rules to dictate what occurs during each time step:

1) Any position in the matrix with a period '.' is considered "dead" during the current time step.
2) Any position in the matrix with a capital 'O' is considered "alive" during the current time step.
3) If an "alive" square has exactly two, three, or four living neighbors, then it continues to be "alive" in the next time step.
4) If a "dead" square has an even number greater than 0 living neighbors, then it will be "alive" in the next time step.
5) Every other square dies or remains dead, causing it to be "dead" in the next time step.

For this program, a neighbor is defined as any cell that touches the current cell, meaning each current cell, regardless of position, has 8 neighbor cells. Cells located at the edge should "wrap around" the matrix to find their other neighbors. The two examples below show the neighbors (**N**) for a given cell (**C**) with example #1 showing a cell in the middle of a matrix and example #2 showing a cell on an edge to demonstrate the "wrap around" effect.

**Neighbor Example #1**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | | | | | | |
| 1 | | N | N | N | | |
| 2 | | N | C | N | | |
| 3 | | N | N | N | | |
| 4 | | | | | | |
| 5 | | | | | | |

**Neighbor Example #2**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | N | | | | N | N |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | N | | | | N | N |
| 5 | N | | | | N | C |

Your solution will accept the starting matrix (**Time Step 0**) as a file from the command line and then simulate steps 1 through 100. The final matrix (**Time Step 100**) will then be written to an output file whose name and path is dictated by a separate command line argument. These files must contain a copy of your matrix with each row of the matrix printed on separate lines and no characters in between the columns. Example files are provided as part of the project.

### *Example time steps for a 6 x 6 matrix*

**Starting Matrix (Time Step 0)**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | . | O | . | . | . | . |
| 1 | . | . | . | . | . | . |
| 2 | . | . | . | . | . | . |
| 3 | O | . | . | . | . | . |
| 4 | . | . | O | . | O | . |
| 5 | . | . | . | . | O | . |

→

**Time Step 1**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | . | . | . | . | . | . |
| 1 | . | . | . | . | . | . |
| 2 | . | . | . | . | . | . |
| 3 | . | O | . | O | . | O |
| 4 | . | O | . | . | . | . |
| 5 | . | O | O | . | . | O |

→

**Time Step 2**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | O | O | O | . | . | . |
| 1 | . | . | . | . | . | . |
| 2 | O | . | O | . | O | . |
| 3 | . | . | . | . | O | . |
| 4 | . | O | . | O | . | O |
| 5 | . | O | O | . | . | . |

### Multithreading

Your solution must also make use of multithreading and spawn a total number of threads equivalent to the number specified by the user in the -t option. Keep in mind that your program running normally would be considered running serially, in other words on a single thread.

## Additional Rules and Hints

### Program Output

Your program may print as much information as you wish to the screen during execution. The first line of output printed to the screen during execution **must** be "Project :: R#" where R# is your specific TTU R#. That is the only **required** item it must print and is the only line the grading script cares about. Your program correctness grade will be determined by the output found in the file specified by the -o argument.

### Hints for getting started

1) Focus on the utility functions first:
    a. Reading a matrix from a file.
    b. Writing a matrix to a file.
    c. Validating user arguments.
2) Next, focus on writing a fully functional serial solution to the problem. Don't lose sight of the fact this eventually must be multithreaded, so try to pick ways of solving this that you believe can be adapted to run in parallel.
3) Once the serial version is functional, then work to convert your program into a multithreaded application. Keep in mind, this may require re-structuring or re-writing some sections of the code.
4) Generate more matrices on your own and make sure you test, test and test some more. **Do not** rely purely on the provided test case as the only test case you run.

### Python Information

1) Warning:
    a. Your solution must be written to be compatible with Python version 3.x, specifically clean installs of Python 3.8.
    b. Python version 2.x was ubiquitous and heavily documented online for over a decade.
    c. Python version 2.x code is often not compatible with Python version 3.x.
2) Python 3.x Tutorial / Refresher Links:
    a. https://www.w3schools.com/python/default.asp
    b. https://www.tutorialspoint.com/python3/index.htm
3) Python 3.x Tutorial for handling command line arguments:
    a. https://docs.python.org/3.8/howto/argparse.html
    b. https://pymotw.com/3/argparse/
4) Additional information and tutorials for threading:
    a. Links below cover either the multiprocessing and multithreading libraries. For reasons discussed in class it is recommended you use the multiprocessing library.
    b. Multiprocessing - http://zetcode.com/python/multiprocessing/
    c. Multithreading - https://www.tutorialspoint.com/python3/python_multithreading.htm

## What to turn in

A zip archive (.zip) whose name does not matter which contains your source code files such that:

- The primary python file for your project is named <FirstName>_<LastName>_<R#>_final_project.py.
    - Example: Eric_Rees_R123456_final_project.py
- Any additional files necessary for your program to execute are included.

Keep in mind the grading program will demand your python file be named correctly.

## Example Final Output

Below is an example execution of a known correct simulator. Keep in mind, the <u>only</u> line that the grader cares about is the "Project :: R#" line. The rest is ignored by the grader as it cares about what is written in the file specified by -o.

*Execution:*

```
(base) quanah:/CS1411/summer20/project$ python3 Eric_Rees_R123456_final_project.py -i time_step_0.dat -o time_step_100.dat
Project :: R123456

Reading input from file time_step_0.dat.

Simulating...

Time Step #0
.......0..00...0.................0.00...
..0............0..0.0..00...............
.......0.......0.....0...........0...00.
....0..0.........0......0.............0.
........0.....0.0......0.......0........
.......0.....0..........................
.0...........0............0.......00.0..
..0....0...0....00......................
.0.0...........0................0.......
0..........0.........0.............0....
......0........0.......0..........0.....
..00..............0.........0...0.0...0.
.....00.0..0.0.....0.0..................
0........0....00.....0.....00.0.........
0............0............0..0......0...
...00.......0...............0.........0.
0.............0............0.........0..
.........0.....0..............0...0.....
........................................
.0........0...............0......0.0....




Time Step #1
.00......000..0.0..0...000......0.000...
......000.00...0...0.00.........00.0..0.
...0..0.0.....0..0000.0..0...........000
......00......0.......00........0.....00
......0.0....0.0.0......0...............
........0...00.0..................000...
..0...000.....0.00..................0...
0.00........0.000................0000...
00........000....0......................
.00...........000.....0..........00.....
.000..................0...........0....0
....0.......0.0....00.0.............0...
.0000..0.00.0..0..0...0........0.0.....0
.0...0000.0...0.......0...000..........0
.0.00.......00.0...........0000......0..
00............0...........0.00......0...
...00........0.0............00........00
..............0.........................
.........00......................0.0....
.........00.....................0.00....
```

Time Steps 2-98 hidden for space.

```
Time Step #99
...0..0..0......0.0.000.0...000000...0.0
0.0.00...0.00...00..00.00......00......0
00.00..0.00..00.0.00000..0.......0....00
000.0...00..0..000000...0...0...000...00
...00000.00...0.000.0...00.0.....0.0.000
...00.....00000.0.0.....000.....000..000
000.00000.0.00...000.....0.....0.00..0.0
..0..000000.0......000000..0000000......
0000.0..0.000....000.0000.0000.00000000.
...0.0....0.0.00...00.0.......0...0.0000
00.....0....0..0...00.0.......0.00.0.000
.0...000....0..0.00.000..0......00.0.00.
00..0..0.000...00.000000.0....0.0000....
.0..00....0........00...0....0.0000..0..
.0..00..0000...0..0.0...0....0.00.....00
....0.000.00..00...0..000.00..000...0000
..0000.0....0.00...000.00..000000..0000.
..0..000...0.000....000...000000...0.000
0.000.0....0.0...0000000.00.0000..00.000
.....0........0000..00...00..00.00.0.000




Time Step #100
0000000.0..0000..000..00000000...0...000
.00.00.000.00.000.00..000...0.00.00..0..
....000..000000.......0000.......0...00.
.000..00.....00.......0.00.0....000000..
..0..000000.0.00...000..0.000..0..00.0..
..0...0.0.0.0.000...0..00.00...00...00..
000.0...000....000000..0.00...00..0000.0
..........0.00..00...0..0000.0....000000
0000.0.0000.00000.0.....0.00..0...000.00
0..00..00.000000.0...00.000..00..0.0....
00.....00..00000000...0......0.000.00...
00..000000.000.0.00....0..0..00....0.00.
00.00..0.0000.00000...00000..00....00..0
.0..00.000..0.0...0....00...00....0.0...
0.0.0.000..00..00.0.0.0.0000.00...00...0
.0...0000.00..00...0.00.0000........0..0
00000...00.00.....00..0.00......0000....
.00.0..00.00.0.0.000....000.....0000....
0.000.0...0.000..000...0.0.0....0.000...
00.000.0..0.00000..0..000000.....0.0.0..
Simulation complete. Final result stored in the output file time_step_100.dat.

(base) quanah:/CS1411/summer20/project$
```