Cairo University

Faculty of Computers & Artificial Intelligence

Structured Programming (2020/2021)

Assignment 3

Delivery Notes:

- This is a group assignment of 2 members (at most) and the members should be from the same group/lab.
- Both students should work and fully understand everything in the code.
- Due date is on Tuesday Jun 15th, until 11:55 pm
- No late submission is allowed.
- Submission will be on blackboard. It is your duty to ensure that your submission was properly uploaded to blackboard after you finish submitting it. If your submission was not uploaded properly while marking, you will not receive a grade for the assignment.
- No submission through e-mails.

- You will develop a .cpp file that should include a block comment containing students' IDs and name this file task1.cpp, then put that cpp file in a folder named GroupNum_firstStudentID_SecondStudentID and compress them to a .zip file with the same folder name. The compressed file would be the file to be delivered.

- The allowed values for group numbers in the zip file name is **S1 till S44**, and for the **old ByLaw G1 till G4**. Please check your group number using the student names list uploaded on blackboard because writing it wrong will not allow your TA to receive your assignment and you might lose its grade.

- **Failing to abide by the naming conventions of the file or failing to submit the files as per the requested .cpp extension, would result in a ZERO for both team members.**

- **Do not send your code** to anyone**,** so that no other student would take your files and submit it under their names.

- **In case of Cheating you will get a negative grade whether you give the code to someone, take the code from someone/internet, or even send it to someone for any reason.**

- You have to write clean code and follow a good coding style including choosing meaningful variable names.
- You have to use functions and not write all of the code in the main.
- If the input is not valid the program should not close. The program should notify the user that the input was invalid and allow the user to re-enter the input until a valid input is given

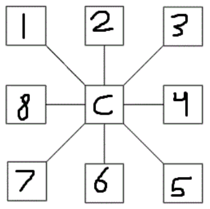
Microscopic Multiply

We want to simulate a small square surface of an area with micro-organisms (like bacteria) and how they spread every cycle of life (iteration) on this surface. This surface is 2-D, its size is n x n and each 1x1 space forms a block. Each can be empty or hold a micro-organism. Every cycle of life (iteration) the micro-organism can either leave the block (this block on the surface will become empty) or stay on the block. An empty surface can attract new micro-organisms or can stay empty. NOTE: When a micro-organism leaves the surface, it just flies into the air and never returns. It does **NOT** move to any other block (this is to avoid confusion).

The rules that govern whether a micro-organism stays on its block on the surface or leaves it and that govern whether an empty block on the surface attracts a new micro-organism is as follows:

- a- If a micro-organism is close to 4 or more micro-organisms it will leave the block making it empty.
- b- If a micro-organism on a given block is alone or close to only 1 micro-organism the micro-organism will leave the block making it empty.
- c- If a micro-organism is close to 2 or 3 micro-organisms it stays.
- d- An empty space with 3 and only 3 close micro-organisms attracts a new micro-organism and it stops being empty.

Now a micro-organism is close to another micro-organism if it is within the 8-connect direction of it. The 8 directions are up↑, upper-right↗, right→, lower-right↘, down↓, lower-left↙, left ←, upper-left↖. The following illustrations shows the close blocks to the block labelled '**C**'. The blocks around the '**C**' block are the 8-connect directions that you should check.



An example of the surface, with 2 iterations and 'M' being a micro-organism and '-' being an empty space and '[' ']' being a boundary (this example will be in 7x7, you are required to simulate a n x n) is as follows:

*We will also use the following colors to point your attention to the rule that applied on the block in its current iteration and made it change. The rules we are mentioning are the four mentioned above. The colors used will be: **a** = leaves, **b** = leaves, **c** = stays, **d** = attracts new one.

```
Start iteration (0)               iteration 1                     iteration 2

[--MMM--]          →→→            [--M-M--]          →→→          [----MM-]
[--M-M--]          →→→            [--M-MM-]          →→→          [--M-MM-]
[----M--]          →→→            [---M---]          →→→          [---MM--]
[-------]          →→→            [-------]          →→→          [-------]
[-------]          →→→            [-------]          →→→          [-------]
[-------]          →→→            [-------]          →→→          [-------]
[-------]          →→→            [-------]          →→→          [-------]
```

In this task you are required to simulate multiple cycles of life (iterations) as the example above with the rules stated before. Your input will be the number 'N' of iterations to draw (taken as input from the user) and the initial state of the surface (You can have it fixed somehow in the code but note that you might need dynamic arrays to re-allocate. Being fixed means that it will be fixed after compiling, but you can still edit the code to change how the initial state looks and recompile).

You are required to calculate and get the state of the board for the next 'N' iterations/cycles of life, starting with the given initial state. For every iteration (including the initial one) you will draw how the surface will look.

For this task you are required to write some helper functions to make the process easier, you can add more functions if you need to. You will have to figure out what parameters to pass to the functions and what to return. But a hint will be supplied for some of them. (Please note that a cycle is an iteration).

The functions required will be:

A function which should return the number of close micro-organisms in a given block. *This function will need access to the surface, and the location of the block you want to know how many close micro-organisms it has.*

A function which should return whether in the next iteration, this block will be empty or have a micro-organism. *This function will at-least need access to the block's state and its location.*

A function which should prepare the next state of the surface and return it. *This function will require at-least the current surface.*

A function which will apply the new states from the next state surface to the current state surface. *This function will be called in the latter part of the code.*

A function which will take the current surface states and print them in a way similar to the example shown above (choose distinguishable characters and make sure you can separate each iteration). The input number of iterations will typically be 12 or more iterations.

Note that surface state means all the 2-D blocks on the surface. Also, the print in the example shows them side by side, **BUT** you are required to print each surface state separately as follows (do not draw the arrows, they are just for illustration). The dimension of the surface is n x n meaning n x n blocks/elements.

```
[--MMM--]
[--M-M--]
[----M--]
[-------]
[-------]
[-------]
[-------]

↓↓↓↓↓

[--M-M--]
[--M-MM-]
[---M---]
[-------]
[-------]
[-------]
[-------]

↓↓↓↓↓

[----MM-]
[--M-MM-]
[---MM--]
[-------]
[-------]
[-------]
[-------]
```