150 points. Individual Work Only. Due Sep 12, 2024 before 12:15 PM.

Objectives:

To design and implement the "Game of Life" program.

Problem Statement:

The "Game of Life" is a well-known cellular automaton devised by John Horton Conway, a Cambridge mathematician. The Game of Life is a board game that consists of a two-dimensional array of cells. Each cell can hold an organism and has eight neighboring cells (left, right, top, bottom, top-left, bottom-right, top-right, and bottom-left). Each cell can be in two states: alive or dead. The game starts with an initial state and cells either live, die or multiply in the next iteration (generation) according to the following rules:

1. If a cell is “alive” in the current generation, then depending on its neighbor’s state, in the next generation the cell will either live or die based on the following conditions:
    - Each cell with one or no neighbor dies, as if by loneliness.
    - Each cell with four or more neighbors dies, as if by overpopulation.
    - Each cell with two or three neighbors survives.
2. If a cell is “dead” in the current generation, then if there are exactly three neighbors "alive" then it will change to the "alive" state in the next generation, as if the neighboring cells gave birth to a new organism.

The above rules apply at each iteration (generation) so that the cells evolve, or change state from generation to generation. Also, all cells are affected simultaneously in a generation (*i.e.,* for each cell you need to use the value of the neighbors in the current iteration to compute the values for the next generation).

The objectives of this homework are:

1. design and implement a program in C/C++ to simulate the "Game of Life"
2. test the program for functionality and correctness
3. measure the performance of the program and optimize the program to improve performance

The program must take the size of the board and the maximum number of generations as command-line arguments. The game will end when there is no change in the board between two generations or when the maximum generation count is reached.

Guidelines and Hints:

1. Instead of allocation a two-dimensional array of size N X N, where N is the size of the board, allocate a two-dimensional array of size (N+2) X (N+2). This will create an extra set of cells around the board, commonly referred to as ghost cells. Initialize these ghost cells to be "dead" (by following the steps shown in the diagram). Having these ghost cells avoids the need to check if a cell is along the boundary and makes the computation of number of neighboring cells "alive" simpler. Also, note that you will require two arrays, one for the values from the previous generation and one for the new values computed in the current generation.

Note: The figure depicted below is a sample example. You can create your own board and initialize with arbitrary "alive" and "dead" cells (e.g., using a random number generator).

	1	1	1	
	1	1	1	
	1	1	0	

(a) Initial state of the board before the ghost cells are filled in **Generation 0**

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	0	0
0	0	0	0	0

(b) State of the board after all the ghost cells are filled in **Generation 0**

0	0	0	0	0
0	1	0	1	0
0	0	0	0	0
0	1	0	1	0
0	0	0	0	0

(c) State of the board in **Generation 1** and ghost cells remain unchanged

2. If you are programming in C/C++ for the first time, use static arrays to initially design and test the program. Then you can change the program to use dynamic arrays using the sample program provided to dynamically allocate the array.
3. Test the program for small problem sizes (say, 3X3, 5x5 or 10x10) first. After you are convinced that your program is working correctly, execute your program for the following input parameters and complete the table below:

Test Case #	Problem Size	Max. Generations	Time Taken
1	1000x1000	1000	
2	1000x1000	5000	
3	5000x5000	1000	
4	5000x5000	5000	
5	10000x10000	1000	
6	10000x10000	5000	

Make sure you comment out any print statements you might have to print the board when you execute with larger problem sizes. Also, execute the program three times and use the average time taken.

4. Analyze the performance results and include the analysis in the report. Make sure you include details about the machine where you executed the program. Specifically, include the machine name, OS name and version, processor name, clock speed, amount of memory in the system, and compiler name and version as well as any compiler flags used to compile the program.

Program Documentation:

1. Use appropriate function name and variables names.
2. Include meaningful comments to indicate various operations performed by the program and instructions to compile and run the program.
3. Programs must include the following header information within comments:

```
/*
  Name:
  Email:
  Course Section: CS 481 or CS 581
  Homework #:
  Instructions to compile the program: (for example: gcc -Wall -O -o hw1 hw1.c)
  Instructions to run the program: (for example: ./hw1 1000 1000)
*/
```

Report:

Follow the guidelines provided in Blackboard to write the report. Submit the report as a Word or PDF file. The report will have 10% of the points for this homework.

Submission:

Upload the source files and report (.doc or .docx or .pdf file) to Blackboard in the assignment submission section for this homework. You can create a zip file with all the source files and report and upload the zip file to Blackboard.

Grading:

Rubrics for grading the programming assignment and format of the report is provided in Blackboard.