CS 2213-001 Advanced Programming

Instructor Dr. Turgay Korkmaz

Homework 2
Due date: check BB
!!!! NO LATE HOMEWORK WILL BE ACCEPTED !!!
Total 5 points

Write a C program that plays the Game of Life (a simplified version, no graphics etc.)

Background from web

The Game of Life was invented by the mathematician John Conway and was originally described in the April 1970 issue of *Scientific American* (page 120). The Game of Life has since become an interesting object of mathematical study and amusement, and it is the subject of many websites.

The game is played on a rectangular board/grid of cells, so that each cell has eight neighbors (adjacent cells). Each cell is either occupied by an organism or not. A pattern of occupied and unoccupied cells in the grid is called a *generation*. The rules for deriving a new generation from the previous generation are these:–

1. *Death*. If an *occupied* cell has 0, 1, 4, 5, 6, 7, or 8 occupied neighbors, the organism dies (0 or 1 of loneliness; 4 thru 8 of overcrowding).
2. *Survival*. If an occupied cell has two or three neighbors, the organism survives to the next generation.
3. *Birth*. If an unoccupied cell has precisely three occupied neighbors, it becomes occupied by a new organism.

Examples can be found at http://www.math.com/students/wonders/life/life.html.

Once started with an initial configuration of organisms (Generation 0), the game continues from one generation to the next until one of three conditions is met for termination:

1. all organisms die, or
2. the pattern of organisms repeats itself from a previous generation, or
3. a predefined number of generations is reached.

Note that for some patterns, a new generation is identical to the previous one — i.e., a steady state. When this occurs, termination under condition #2 occurs. In some other common cases, a new generation is identical to the second previous generation; that is, the board oscillates back and forth between to configurations. In rare cases, a pattern repeats after an interval of two or more generations. **In this assignment**, you will be responsible for terminating after **a steady state is reached** or **an oscillation of two alternating patterns is reached** (no need to consider repetitions after more than two generations)**.**

In theory, the Game of Life is played on an infinite grid. In this assignment, your program will play on a finite grid (board). **Suppose the size of the board will be 50 by 30 cells**. The same rules apply, but squares beyond the edge of the grid are assumed to be always unoccupied.

Implementing your program

You are not expected to develop a graphical user interface as in the above web page. You will simply interact with your program on a command line (i.e., input the initial configuration, and the number of generations to play), play the game, and finally print the final configuration on the screen by putting ' ' for no organism and 'x' for an organism.

- You need to declare at least three 2D arrays of 50 by 30 in your program. For example, `int A[50][30], B[50][30], C[50][30];`
- Initialize one of your arrays (e.g., A) as the generation 0. First, set each cell to 0, indicating that there is no organism. Then, ask user to enter which cells have an organism in the initial configuration. In a loop, ask user to enter i and j and set A[i][j] to 1, indicating that there is an organism in A[i][j]. You can stop this loop when user enters -1 -1.
- After getting the initial configuration, ask user to enter the number of generations to play.
- Then your program plays the game for as many generations as needed until one of the three termination conditions above is met.

  Why do we need three arrays?: The number of live neighbors is always based on the cells before the rule was applied. In other words, you must first find all of the cells that change before changing any of them. That is, you need to use generation 0 saved in A to compute generation 1 and save it in B without changing A. Then you can use B to compute generation 2 and save it in C. For the other generation you can reuse arrays in a circular manner and re-use A to save next generation computed based on C etc. Two arrays would be enough for just finding generations, these three arrays are necessary to be able check the above stopping condition #2 if **a steady state is reached** or **an oscillation of two alternating patterns is reached**! Sounds like a job for a computer!
- Print out the final configuration, along with a message saying how many generations were played and under what condition the game terminated.

What to return: !!!! NO LATE HOMEWORK WILL BE ACCEPTED !!!

1. Follow the problem solving methodology, and solve the problem(s). Then convert your solution(s) to a C program. Include /* **comments** */ so that we can understand your solution(s). You can name your program as hw02.c
2. Compile and run it. You should test your Game of Life with several initial conditions, including patterns that you find on the web and compare your final configuration to the one on the above web page.
3. Copy/paste the result in an output file, say out2.txt.
   (or you can use **script** command in linux as follows)
   > script out02.txt # saves everything into out02.txt until you hit **ctrl-d**
   > hw02
   …
4. Zip your files hw02.c and out02.txt as a .zip file
5. Go to BB Learn, and submit your .zip file as an attachment before the deadline.

/* Don't forget to include comments about the problem, yourself and each major step in your program! */

You must submit your work using Blackboard Learn and respect the following rules:

1) All assignments must be submitted as either a zip or tar archive file unless it is a single pdf file.
2) Assignments must include all source code.
3) Assignments must include an output.txt file which demonstrates the final test output run by the student.
4) If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the underlying cause of the error.
