Game of Life



**CS-2303, System Programming Concepts, C-term 2019**

Version 1.3

### Abstract

Write a *C* program that plays the *Game of Life*. Accept as arguments the size of the board, the initial configuration, and the number of generations to play. Play that number of generations and display the final configuration of the board.

### John Conway’s Game of Life

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 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 two configurations. In rare cases, a pattern repeats after an interval of more than two generations. In still other cases (some not so rare), a pattern replaces itself by a fixed offset in one or both dimensions, thereby “flying” off the screen. In this assignment, you will be responsible for terminating after a steady state is reached or an oscillation of two alternating patterns is reached.

In theory, the Game of Life is played on an infinite grid. In this assignment, your program will play on a finite grid. The same rules apply, but squares beyond the edge of the grid are assumed to be always unoccupied.

Note that, according to the rules, all changes for each generation are considered to take place simultaneously. Unfortunately, your program has to work through the board square-by-square. That is why you always need to have at least two arrays: One holding the state of the board at the beginning of the turn, and one which will be filled in with the state of the board at the end of the turn (which will be the state at the beginning of the next turn). It turns out that you will also need a third array; more about this later.

### Implementing your program

Your program needs to do several things:

* Read the arguments to the program from the command line.
* Read the initial configuration of the board from an input file.
* Allocate at least three arrays, each large enough to hold one generation of the game. Initialize the first generation with the initial configuration in the approximate center of the board.
* Play the game for as many generations as needed until one of the termination conditions above is met.
* Print out the final configuration, along with a message saying how many generations were played and under what condition the game terminated.

### Program Arguments and Input

The program should be invoked with the following ``command line’’ arguments:

*NR NC gens inputfile [print] [pause]*

where

* **NR** and **NC** are unsigned integers indicating the number of elements in the *y* and *x* directions of the grid, respectively. In other words, the number of rows and columns, respectively, in the grid
* **gens** is the number of generations to play. This value must be greater than zero. The program should halt after this number of generations. However, the program should halt prior to this number of generations if it determines that the game has reached a termination condition.
* **inputfile** is the name of a file containing a sequence of lines, each consisting of a sequence of **'x'** and **'o'** characters, indicating the occupied and unoccupied cells of the initial configuration. An **‘x’** indicates an occupied cell, an **‘o’** or a blank indicates an unoccupied cell. If a line is shorter than the width of the grid, cells to the right are considered unoccupied. If there are fewer lines in the file than the height of the grid, cells below are considered unoccupied. Note: The **‘o’** characters are included to make it easier to see the patterns in the file. When you print the grid, please only print blanks and **‘x’** characters.
* **print** is an optional argument with value of **'y'** or **'n'** indicating whether each generation (including generation 0) should be printed or displayed before proceeding to the next generation. If this item is missing, it defaults to **'n'.** Note: The user does not type the square brackets; they are used to denote that this argument is optional.
* **pause** is an optional argument with value of **'y'** or **'n'** indicating whether a keystroke is needed between generations. If this and/or the print item is missing, it defaults to **'n'**. It is not possible to specify a value for **pause** if you do not also specify a value for **print.**

After interpreting the program arguments, your program must open the input file, read its lines, and initialize the configuration in the approximate center of your board in the *x-* and *y-*dimensions. In other words, the pattern in the file starts in the upper-left corner of the grid; after you read it in, determine the number of lines and the greatest width, and shift the pattern down and to the right as needed to approximately center it.

#### Example patterns

Here is a simple pattern that happens to be a “still life” or steady state:

xx  
xx

That is, the next generation starting from this pattern produces exactly the same pattern. Here is another still life pattern:

oxo  
xox  
xox  
oxo

The following pattern produces an oscillation between a vertical line of three occupied cells and a horizontal line of three occupied cells

x  
x  
x

The following pattern is a well-studied one called the *R-Pentomino*.

oxx  
xxo  
oxo

It creates an interesting sequence of generations, including many sub-patterns that come, go, and/or fly off the edge of the board, until it finally reaches a steady state after 1176 generations.

If the pause argument is set to **'y'**, the program should wait for the user to input something from the keyboard between calls to PlayOne(). What the user inputs can be as short as the Enter key, but be sure it is clear to the user what should be entered.

### Testing

Since we are using test-driven development, you need to test all the various functions, including the helper functions. For example, test the function which tells if a cell is occupied by setting up a known pattern, then looking at several cells, both within the grid and off the edges.

You should test your complete Game of Life with several initial conditions, including patterns that you find on the web. When the graders test your program, they will use one or more standard input files containing with typical patterns. The program arguments will match the input files.