CS 420
PROGRAM 6P

Purpose

The purpose of this assignment is to give you exposure to a simulation with a 2-D array of structures.

Scenario

The Game of Life, invented by John H. Conway, is supposed to model the genetic laws for birth, survival, and death. We will play the game on a board that consists of a maximum of 25 squares in the horizontal and vertical directions (a total maximum of 625 squares). Each square can be empty, or it can contain an X indicating the presence of an organism. Each square (except the border squares) has eight neighbors.

The next generation of organisms is determined according to the following criteria:

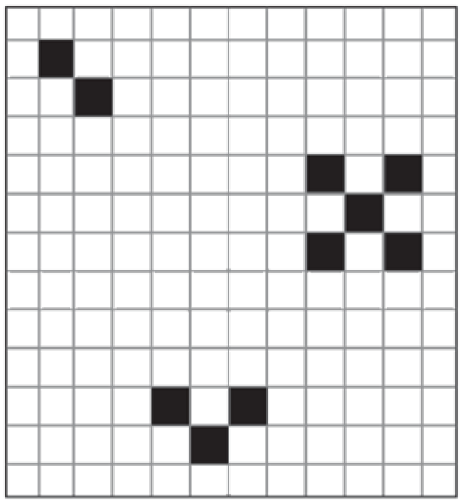
1. Birth – an organism will be born in each empty location that has exactly three neighbors.
2. Death – an organism with four or more organisms as neighbors will die from overcrowding. An organism with fewer than two neighbors will die from loneliness.
3. Survival – an organism with two or three neighbors will survive to the next generation.

Take an initial configuration of organisms as input data. Display the original game array, calculate the next generation of organisms in a new array, copy the new array into the original game array, and repeat the cycle for as many generations as you wish. Assume that the borders of the game array are infertile regions where organisms can neither survive nor be born; you will not have to process the border squares.

To illustrate an example of how the game works let us look at two successive generations of the game. The squares filled in black indicates the existence of an organism in that generation.

Let us say our first generation looked like this:

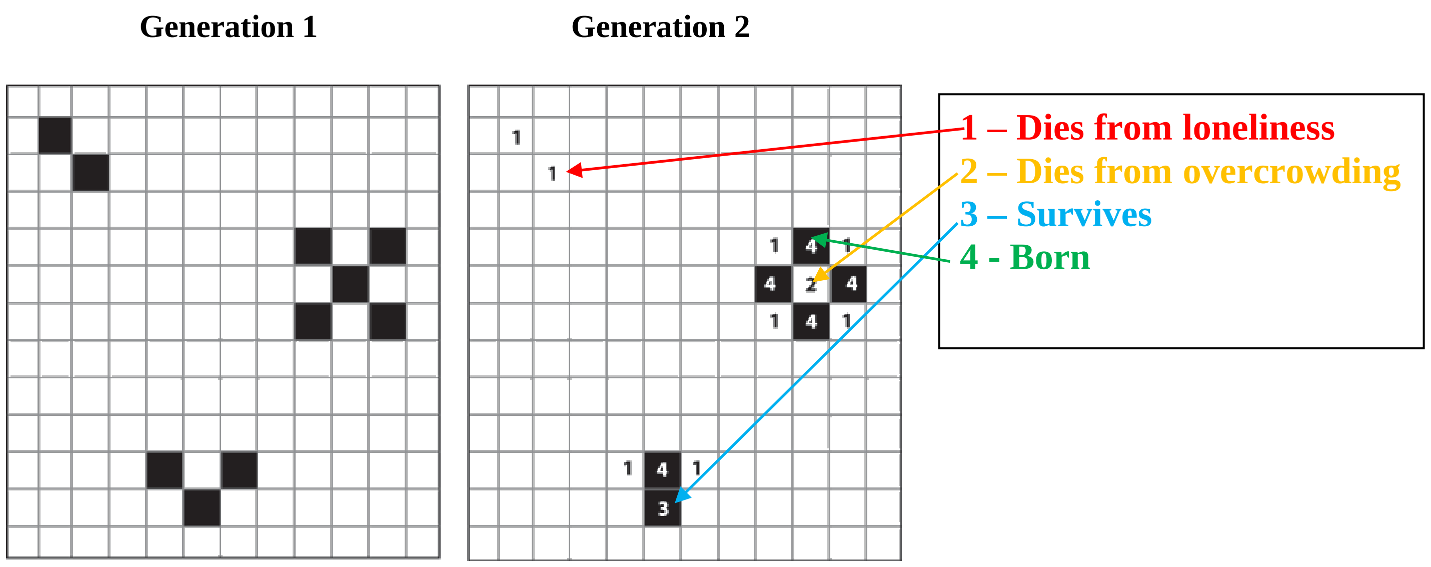
Generation 1



Here are the rules for every cell/parcel:

- *For a populated parcel:*
    - ***Rule (1)*** *Each organism with one or no neighbor organism dies from loneliness.*
    - ***Rule (2)*** *Each organism with four or more neighbors perishes due to overspill overcrowding.*
    - ***Rule (3)*** *Each organism with two or three neighbors survives because of the balance of the social structure.*
- *For an unpopulated parcel:*
    - ***Rule (4)*** *A new organism arises from procreation on a parcel with exactly three neighbors. (It is insignificant which of the three neighbor organisms are involved in the procreation and what their sex is).*

The numbers in the cell for Generation 2 shows the Rule number used (above) to indicate whether an organism survived, died or was reborn in any cell. Once again only the squares in black show cells that have an organism in that generation.



- We would also like to maintain some additional information about each parcel/cell, listed as follows:
    - How many deaths occurred in a cell?
    - How many births occurred in a cell?
    - How many iterations did the cell have an organism in it?

    (initial birth and death counts in a cell are zero, initial state of the game will determine whether the iterations for a cell having an organism is 0 or 1)

Details

- Your program should read data from an input file for the initial configuration and display it. Then you need to create the next generation of organisms by applying the above rules. You display the next generation of organisms and continue to generate them until one of the following conditions are reached:
    a. There are no more organisms alive, so we can stop creating generations.

b. The total number of organisms do not change for 5 consecutive generations. If this happens we can assume that the configuration of the generation is such that it will maintain the same number of organisms forever and therefore we can stop.

- Your output does not have to match mine exactly but should closely resemble it.
- You are required to implement your solution using a 2-D array of structures where each element of the array maintains information about the state of a cell

Structure to be defined:

```
typedef struct {
          ………..
          ………..
} GridCell;
```

Constants to be defined:

```
/* The following constants are used for statistics */
#define DEATHS 1
#define BIRTHS 2
#define ITERATIONS 3
```

Functions needed:

(note that array content can be changed within a function by using it like a typical input argument i.e. without the "&" to pass the address, since the address of the first element of the array is what gets passed in)

int initializeGrid(char [], GridCell [][25], int *);
(input arguments: filename, array of structures, output argument: size of row/column)
Function return value: count of organisms alive

void printGrid(GridCell [][25], int, int, int);
(input arguments: array of structures, size of row/column, generation count, total organisms alive)

int computeNextGeneration(GridCell [][25] , GridCell [][25], int);
(input arguments: array of structures for current generation, array of structures for next generation, size of row/column)
This function should also copy over the new generation (array) into the old generation (array) in preparation to compute another generation. This can be done either in the beginning or at the end of the function. *This copying of the array should not be done in your main() function.*
Function return value: count of organisms alive

void getCellStatistics(GridCell [][25], int, int, int *, int *, int *);
(input arguments: array of structures, row number, column number, output arguments: death count, birth count, iteration count)

void printStatistics(GridCell [][25], int, int);
(input arguments: array of structures, size of row/column, statistics to print: 1 for deaths, 2 for births, 3 for iterations)

Files to be submitted:

golFunctions.c (should contain function definitions)
gol.h (should contain constants, structure definition, function prototype declarations)
gol.c (should contain main() function)
makefile

The makefile should have the following targets:
gol – to build your test driver gol.c with the functions defined in golFunctions.c

gol.o – to build and create just an object file (.o file) after compiling your gol.c module

golFunctions.o – to build and create just an object file (.o file) after compiling your golFunctions.c module

clean – delete all object and executable files

Algorithm for main() function

- Read command line argument
- Process input file and initialize grid
- Print initial state
- Loop
    - Compute the next state of the game
    - Print state of the game
    - Keep track of total organisms for the last 5 states
- Until no organisms are alive or last 5 states yielded the same number of organisms
- Print statistics

Input

The input data file name is supplied as a command line argument. The first line of the input file will have a number indicating how many squares exists in one row and one column (it's a square array i.e. same number of rows and columns). You can assume that the maximum size of the array is going to be 25 x 25 although some of the input files have data that fit into smaller arrays. The input files contain just the initial configuration of organisms.

Sample Output

Sample input and output can be found in the public folder for 6P. Here's an example of how a generation looks like with a 15x15 grid:

```
    #### GENERATION 3 #### Total Organisms: 62 ####
------------------------------------------------------------
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
------------------------------------------------------------
|   |   | x | x |   | x | x | x | x | x | x | x | x |   |   |
------------------------------------------------------------
|   | x |   |   |   | x | x | x |   |   |   |   |   | x |   |
------------------------------------------------------------
|   | x | x |   |   |   |   |   |   | x | x |   |   |   |   |
------------------------------------------------------------
|   |   | x |   |   |   |   |   |   | x |   |   |   |   |   |
------------------------------------------------------------
|   | x | x |   |   |   |   |   | x |   |   | x |   |   |   |
------------------------------------------------------------
|   | x | x |   |   |   |   | x | x |   |   |   |   | x |   |
------------------------------------------------------------
|   |   |   |   |   |   |   |   | x |   | x | x |   | x |   |
------------------------------------------------------------
|   |   |   |   |   |   |   | x |   | x | x |   |   |   |   |
------------------------------------------------------------
|   |   |   |   |   | x | x | x |   | x |   |   | x |   |   |
------------------------------------------------------------
|   |   |   |   |   | x |   | x |   | x | x | x | x |   |   |
------------------------------------------------------------
|   |   |   |   |   | x | x |   | x | x |   | x |   | x |   |
------------------------------------------------------------
|   |   |   |   |   | x | x |   | x |   | x | x | x |   |   |
------------------------------------------------------------
|   |   |   |   |   |   |   |   | x | x |   |   |   |   |   |
------------------------------------------------------------
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
------------------------------------------------------------
```