C Programming Lab
Assignment#3

Revision Date: November 7, 2014

This is your fourth assignment.

Life!

> "Life! Don't talk to me about life." – *Marvin the Paranoid Android*, **The Hitchhiker's Guide to the Galaxy**

Your task is to program the classic Computer Science demonstration of cellular automata, *The Game of Life*.

You will read about Conway's Game of Life on the Web to understand the strategy involved in this game. The strategy involved will help you in developing the program.

Creating a two-dimensional list

The Game of Life is most easily implemented using a two-dimensional array. Suppose you wish to create a two-dimensional array, which is an array of arrays, with r rows and c columns.

To create a backbone of size of rows and columns, we do the following:

```
backbone = (int**) malloc(sizeof(int *) * rows);
for(i=0;i<rows;i++)
    backbone[i] = (int*) malloc(sizeof(int)*columns);
```

Now, the variable backbone points to the two-dimensional array. Pictorially, the array looks as shown in Figure 1. This 2-D array has five rows and eight columns. The backbone array is shown in yellow. Note that it has five slots since it is a holder for the five rows. The row arrays are shown in red. Each of the row arrays have eight slots since there are eight columns.

To access the i^{th} row and the j^{th} column, one uses the notation:

```
storedValue = backbone[i][j];
```

To set the value at the i^{th} row and the j^{th} column, one uses this notation:

```
backbone[i][j] = newValue;
```

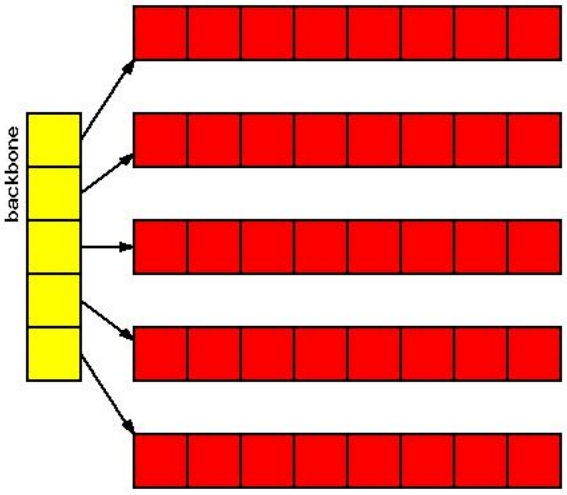

Figure 1: Schematic illustration of 2-D arrays

Remember, arrays use zero-based counting!

Input / output

The name of file containing the first generation and the number of generations in the simulation are to be passed as command-line arguments, as in:

```
./life xxxx yyyy
```

where *xxxx* is the name of the file containing the initial generation and *yyyy* is the number of generations to simulate.

Your program should output a series of generation representations preceded by the generation number, starting with generation zero. See the compliance instructions for what a generation representation looks like. If n generations are desired, then n generation representations should be output by the program, each preceded by its generation number.

To read a character at a time, use the *fscanf* function and *%c* format specifier.

The initial generation

The initial generation is to be stored in a file and will look similar to:

```
........................xx........
..x.............xx........xxx.......
..x...x.........x..x........xxx.....
................xx..........xx......
......x.....................x.......
```

```
x....x.........x.........xxxx.......
...............xx.............x...
.x...x..............x..xx......x...
................................xx..
..x.......x...........x...........x..
```

The file will be composed of only dots and x's with no blank lines. To determine the dimensions of the grid, read the first line and count the number of the characters in the line using *strlen* function (remember to take into account the newline character at the end of the line - you don't want to count that). This step gives you the number of columns. Then keep reading lines to determine the number of rows. Once the number of rows and columns have been determined. close the file. Always remember to close the file handler when you are finished reading.

Stepwise refinement

You may use stepwise refinement in implementing this assignment.

Level 0 Write a program that prints out the name of the file containing the initial population and the number of generations to simulate.

Level 1 Modify the program to print out each line of the initial population.

Level 2 Modify the program to print the number of rows and the number of columns in the initial population file.

Level 3 Modify the program that reads each character of the initial population file, placing that character into its proper place in a two-dimensional list. Print out the two-dimensional list in a pleasing format when finished.

Level 4 Write a program that prints out the initial generation (unmodified each time) for the desired number of generations.

Level 5 Write a program that, for each generational step, copies the current generation to another two-dimensional list, replacing each non-dot character with 'x'. This new two-dimensional list becomes the current generation.

Level 6 Like Level 5, but replaces each non-dot characters with the number of non-dot characters in the neighboorhood. Note, the only change you will see is from the first generation to the second. After that, there should be no changes.

Level 7 Like Level 6, but replaces each non-dot characters with a dot or an x, depending on the number of non-dot neighbors.

Level 8 Like Level 7, but also replaces each dot characters with a dot or an x, depending on the number of non-dot neighbors.

Create a directory called *life* which hangs off *clab* directory. Write your main function in *life.c* file. All other functions should be written in *util.c* file. The function declarations should be specified in *util.h* file. You need to have *makefile* and *README* file in the directory. Note that *life.c* should have a main function and it should only have a series of actions to the generations via function calls.

Compliance Instructions

To make sure that you have implemented your program correctly, copy the initial generation into a *test.dat* file in your project directory. You should then be able to run the following command:

```
./life test.dat 2
```

and see the exact output:

```
Generation 0
........................xx........
..x.............xx........xxx.....
..x...x.........x..x........xxx....
................xx..........xx....
......x.................x.........
x....x..........x..........xxxx....
..............xx..............x...
.x...x................x..xx......x...
................................xx..
..x.......x............x...........x..
Generation 1
........................w.x.......
................ww...........x.....
..............w..w.......x.........
................ww...........w.....
................x.........w.........
..............wx.........wwww......
..............ww........x..x.......
................................xw...
................................ww..
................................xw..
Generation 2
....................................
................vv.........x.........
..............v..v...................
..............xvv...................
........................v.x.........
................x.......xv.v........
..............vv.........x.w........
................................wvx..
....................................
................................wv..
```

If your code does not produce this output or fails with a runtime error while running this test, then you will receive a zero for this assignment. Also you need to compile your program using *makefile*. Remember to include *README* file in the directory.

Submission Instructions

Change to the directory containing your assignment.

```
submit clab mr life <your-iiitb.org-email-address>
```

Due Date

The project is due by December 1 by 11:59:59 AM.