ESS201: Programming-II
Module: C++

Jaya Sreevalsan Nair
jnair@iiitb.ac.in
International Institute of Information Technology, Bangalore

Lab-02: Take-home assignment on October 11, 2018
Submission Deadline: 11:59:59 pm IST, October 23, 2018 (Tuesday)

Premise: Conway's Game of Life

Conway's Game of Life is the simplest two-dimensional cellular automaton, devised by John Horton Conway, in 1970. Cellular automaton, a discrete model studied in varied STEM (science, technology, engineering, mathematics), is a regular grid of cells, where each cell is in one of the finite number of states, e.g. on or off. The grid changes state based on a *fixed* rule applied to its neighborhood.

In Conway's life of game, each cell has eight neighbors (Moore neighborhood of range-1 cell) and two states ("live" and "dead"). The rules applied in Conway's Game of Life allow transitions between live and dead states of the cell (i.e. between on and off states). At each step in time, the following transitions occur:

1. Any live cell with fewer than two live neighbors dies, as if by underpopulation.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by starvation or overpopulation.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Assignment

The objective of this assignment is to write a C++ program, `initialize_game_of_life.cpp`, for generating a state of the initial grid for Game of Life. The aspect of your coding exercise that you must consider is using C++ classes.

- Define grid cell, `gridcell` to be a `class`, with the following data members: `int x, y; uchar state; uchar neighborhood[8];` where `(x,y)` specifies the location of the grid cell in the grid, and `state` is either '0' or '1' depending on the state of the cell being dead or alive, respectively. The `neighborhood` value is generated as per ordering of indices given in Figure 1. The value of each array element in `neighborhood` is the `state` of the corresponding neighbor.
- Create `grid` as a `class` with a data member of dynamically allocated pointer of pointer to grid cell: `gridcell **gridcells;`
- Inputs to the program named `gameoflife` are number of rows of the grid, number of columns of the grid, and the number of live cells in the grid.
  `./gameoflife X Y L`

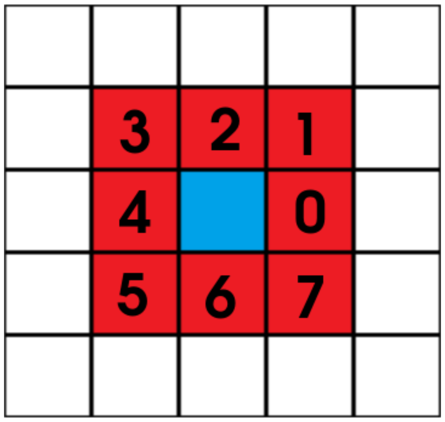

Figure 1: Ordering of indices of the neighbors of the Moore neighborhood (red cells) of a given grid cell (blue cell), which must be used for index ordering in the `neighborhood` array value of the concerned `gridcell`.

- Some details on the following functions:
  - `int main()`: This function must include checking the validity of inputs, i.e., if `X`, `Y`, and `L` are positive integers and `L <= (X-2)*(Y-2)`; dynamically allocating memory of `(X*Y)` gridcells in `grid`, initializing `grid`, invoking the functions given below, and printing the output, as given below.
  - `void generate_initial_state`: This is a class method in `grid` to generate the initial state of the grid. This function must randomly assign '0' or '1' as state of each of the grid cells, which must be done by invoking the `set_state` class method of `gridcell`. This function must, however, follow two conditions:
    - All boundary grid cells (i.e. grid cells in the first and last, row as well as column) will be '0'.
    - The total number of grid cells in the grid with value '1' will not exceed `L`.
  - `void update_neighborhood`: This function must update the value of `neighborhood` of each grid cell. Again, this must be done by invoking the class method `set_neighborhood` of class `gridcell`, for each grid cell, i.e., instance of the class `gridcell`.
  - `int count_live_neighbors`: For a grid cell, this function gives the number of live neighbors. This must be a class method of `gridcell`.
  - `void print_grid`: In order to test the correctness of the state of the grid, the grid may be printed row-wise, with a row in a line, single space between each element, 'o' to represent grid cells with state '0' and '+' to represent those with state '1'. This function is to facilitate your visual debugging, going forward. There must be friend methods using output stream for both `gridcell` and `grid`, which must be used. Hence, the function `print_grid` need not be written as such, but must be subsumed by the `friend` methods.
  - You may optionally write functions to generate the consecutive generations of the grid, based on the rules of Conway's Game of Life. Care must be taken the updates are done in two phases: first one to update `state` of all grid cells, and then the second subsequent one to update the `neighborhood` of all grid cells. They must appropriately be class methods of `gridcell` and `grid`.
- Output: the sum of live neighbors of all the grid cells in the grid. If any of the inputs are invalid, the output must return "0".