ESS201: Programming-II
Module: C++

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

Lab-03: Lab assignment on October 23, 2018
Submission Deadline: 11:59:59 pm IST, October 30, 2018 (Tuesday)

1 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) applications, 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, `game_of_life.cpp`, for:

1. reading an input ASCII file for initializing the state of the grid, and
2. generating the state of grid for Game of Life, after a given number of generations, given as input.

The format for running your executable:

```
$./game_of_life <input-file> <number-of-generations>
```

where `<input-file>` is the name of the file containing the initial grid, in the belowmentioned format, and `<number-of-generations>` is a positive integer, which is the number of generations the game of life is simulated for.

The format for the input file:

- The file contains 2+`<number-of-rows>` lines, one for the header, one for the end-line, and the rest of the files contains the state information per row.
- The header or the first line ([line-1]) gives the grid size and the generation:
  `<number-of-rows> <number-of-columns> <generation>`
- The information per row is the state of the grids in the specific row, hence contains `<number-of-columns>` characters, where 'o' represents 0-state and '1' represents 1-state of the corresponding grid.

  ```
  o o o o o ...  o
  o o o + o ...  o
  o o + + o ...  o
  ```

- The end-line contains the word `eof`.
- The file reads in the following ordering of the grid: top to bottom, and left to right, which means:

  line-2 contains grid cells [0, 0] to [0, `<number-of-columns>`-1] of the grid, and
  - line-[`<number-of-rows>`+1] contains grid cells [`<number-of-rows>`-1, 0] to [`<number-of-rows>`-1, `<number-of-columns>`-1], of the grid.
- For an input file, <generation> must be 0.
- If the <generation> is non-zero, the end-line is missing, or the grid size is zero, the input file must be considered *invalid*.

Example file:

```
3 4 0
o o o o
o + + o
o o o o
eof
```

The expected output:
The output is the grid after `<number-of-generations>` generations. The output must be printed in the same format as the input file.

- If the input file is invalid, then the output is a grid of size zero. The corresponding output file is as follows:

  ```
  0 0 0
  eof
  ```

Validity of the grid implies the following:
At any generation (including the initialization), the grid is considered *invalid* if the condition about 0-state boundary cells is violated.

- In case the grid becomes invalid at a generation after the initialization but has not reached generation `<number-of-generations>`, the simulation of game of life may stop and the last valid grid needs to be output with its appropriate generation in the header.

Notes:

- **This assignment is to be built on the code developed for Lab-02.**
- Write separate functions to check validity of input-file and grid, namely, `is_valid` functions.
- The simulations for generations of game of life must use the rules given in Section 1.
- Make a temporary data storage, namely a *proxy*, to store the values of the grid cells in the next generation. Only if the grid is valid, the data needs to be updated on the grid.
- `http://www.cplusplus.com/reference/fstream/ifstream/` is useful for the formats and a working example of `std::ifstream`.
- `http://www.cplusplus.com/reference/fstream/ofstream/ofstream/` is useful for the formats and a working example of `std::ofstream`.
- Optionally, write a *Makefile* for this assignment.