CSE2122 (Spring 2020)
Homework 2, due 1/31/2019 11:59pm

Before you start, please login your stdlinux account and download the solution file and the template program from the class directory to your stdlinux directory. You can use the command "**`cp /class/cse2122/hw2/* ./.`**".

Submit your code to the designated dropbox on Carmen under the Assignments tab. Late submissions receive a 10% penalty for each day. No late submissions after the second day.

Skills needed to complete this assignment: functions, dynamic arrays.

The mathematician John Horton Conway invented the "Game of Life." Though not a "game" in any traditional sense, it provides interesting behavior that is specified with only a few rules. This project asks you to write a program that allows you to specify an initial configuration. The program follows the rules of LIFE to show the continuing behavior of the configuration.

LIFE is an organism that lives in a discrete, two-dimensional world. This world is an array with each cell capable of holding one LIFE cell. Generations mark the passing of time. Each generation brings births and deaths to the LIFE community. The births and deaths follow the following set of rules.

- We define each cell to have eight neighbor cells. The neighbors of a cell are the cells directly above, below, to the right, to the left, diagonally above to the right and left, and diagonally below to the right and left. Be careful when checking for neighbors on the edges.
- If an occupied cell has zero or one neighbors, it dies of loneliness. If an occupied cell has more than three neighbors, it dies of overcrowding.
- If an empty cell has exactly three occupied neighbor cells, there is a birth of a new cell to replace the empty cell.
- Births and deaths are instantaneous and occur at the changes of generation. A cell dying for whatever reason may help cause birth, but a newborn cell cannot resurrect a cell that is dying, nor will a cell's death prevent the death of another, say, by reducing the local population.

Please check here for some example patterns. You can also test your own patterns with the provided solution file.

In this assignment, you will implement several functions to complete a program that simulates this game. The program will first ask the user for the world's dimensions (rows and columns), the number of initial alive cells, and the initial alive cells' coordinates. After that, the program will display the initial world with spaces for dead cells, and asterisks for alive cells. The user can type in 'G' for the next generation, or type 'Q' to quit.

Prototypes of the following functions are provided in the template, you must implement and use the following functions in your program.

- `void initialization(bool **world, int nrows, int ncols)`
  This function takes a 2-dimensional Boolean array we call world, and the size of this array in rows and columns. It prompts the user input for the number of initial alive cells and their coordinates, and sets the initial world according to the inputs.

- `int neighbor_count(bool **world, int nrows, int ncols, int i, int j)`
  This function takes a 2-dimensional Boolean array we call world, and the size of this array in rows and columns, and one cell's coordinates (indices for rows and columns). It counts how many neighbor cells are occupied for the input cell, and returns the count.

- `void generation(bool **world, bool **copy, int nrows, int ncols)`
  This function takes the world array, a copy of the world array (it can be used to help count neighbors while cells are turned on or off in the original vector), and the size of the world. The function scans the array and modifies the cells, marks the cells with births and deaths in accord with the rules listed earlier. This involves examining each cell in turn, either killing the cell, letting it live, or, if the cell is empty, deciding whether a cell should be born.

- `void display(bool **world, int nrows, int ncols)`
  This function accepts the array world and displays the array on the screen. For better visualization, we also display a border of the world.

You are free to add necessary helper functions to this program. If you have any dynamic arrays in your program, make sure they are deleted after use.

Please do NOT:

1. modify the main function except at the places commented with /* your code here */ (for allocating/deallocating dynamic memory).
2. modify the prototypes (including data types of input parameters and the returned values) of the above four functions.
3. use C/C++ libraries other than `iostream`.
4. use any global non-constant variables in your program.

Test cases

Four test cases (beehive, blinker, glider, and pulsar, please check here for visualization) are provided on Carmen (in hw2_testcase.txt).

Hints

Note that not all cells have eight neighbors. For example, a cell at the top left corner does not have any neighbors above it, nor any neighbors on its left side.

To count the occupied neighbors of one cell, an intuitive idea would be having multiple nested if-else structures to check all possible nine cases (four corners, four borders, and cells not at corners/borders), and count the neighbors in each branch. This works, but you will need to deal with a lot of details and write a very long program for it.

To do the same task with a shorter program, we can first assume that all cells have eight neighbors, get the coordinates of these neighbors, and only exclude the invalid neighbors (whose coordinates go out of the map's range) during the check. The following pseudocode describes how to implement this idea with for loops.

```
Initialize a variable count to be 0
for (int di=-1; di<=1; ++di)
   for (int dj=-1; dj<=1; ++dj)
      if di≠i or dj≠j: // to skip the input cell itself
         initialize the neighbor's coordinates as (di+i, dj+j)
         if (di+i, dj+j) is within the map range and it is occupied:
            count += 1
return count's value as the result
```

Program Submission

Compile your code with the following command. Submissions do not compile with the following command will receive zeros.

```
g++ <your file name> -std=c++11 -pedantic-errors
```

Make sure your code is formatted properly. Up to 5% of the assignment credit can be deducted if your program does not have appropriate indention and necessary comments. See textbook on section 2.5 or any code example in textbook/lecture slides.

Submit your code (the .cpp file) to the designated dropbox on Carmen. You can find the dropbox under the Assignments tab.