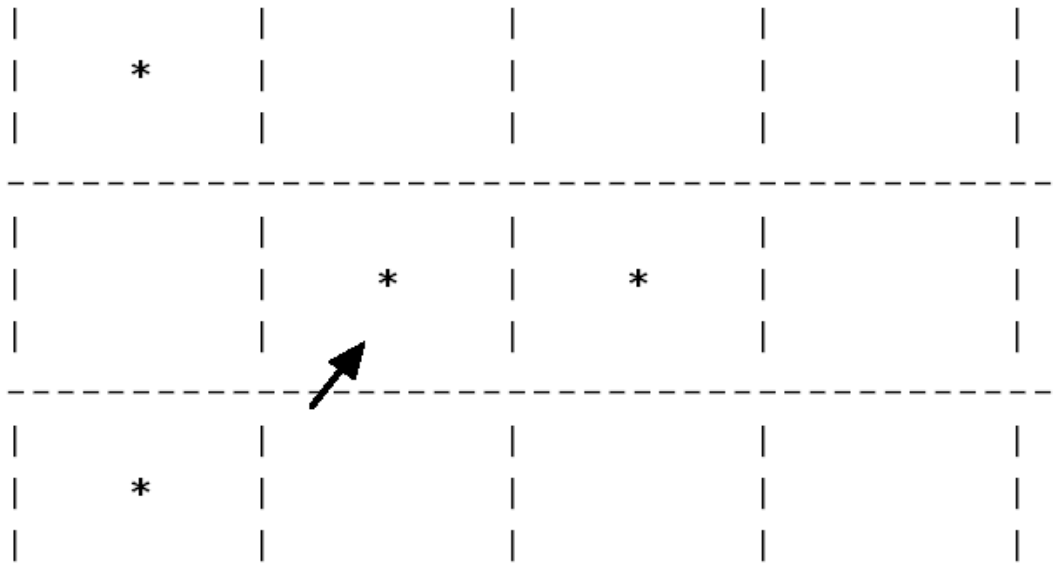
CPS212 - COMPUTATIONAL STRUCTURES AND ALGORITHMS

Project #1 - Due Wednesday, February 3, at the start of class

Purpose: To give you experience using C++ and working with some very basic computational structures.

Background: The game of LIFE, invented by mathematician John H. Conway, is intended to model life in a society of organisms. That is, the game follows the development of an initial pattern of organisms as new organisms are born and old organisms die. Using simple rules for birth, death, and survival we model the growth of the "society" of organisms. Some initial patterns of organisms rapidly die off; others result in new patterns that repeat in a cyclic manner; others change shape and size and may move as a society through their world; others may produce "gliders" that detach themselves from the society and move off on their own. (See Scientific American, October, 1970, p. 120 if you are interested in a discussion of the simulation.) We will play the game on a rectangular board consisting of 900 locations (18 rows by 50 columns). Each location can either be empty (displayed as blank) or contain an organism (displayed as some symbol such as an asterisk) in some stage of life (i.e. surviving, dying, being born). Each location, except those along the edges, has 8 neighboring locations.



For example, the location in the second row and second column of the partial playing board above (the one pointed to by the arrow) has eight neighboring locations. Three of the eight neighboring locations contain organisms and the other five are empty. Births and deaths occur in the game according to the following rules.

1. During each cycle of play, an organism is born into any empty location that has exactly three neighboring locations containing organisms.
2. All organisms with fewer than two neighboring organisms die of loneliness.
3. All organisms with more than three neighboring organisms die of overcrowding.
4. All other organisms (those with two or three neighboring organisms) survive into the

next generation.

Note: when determining the number of neighboring organisms, a dying organism is still considered a neighboring organism; however, a gestating organism is not (i.e. count it if it's marked as dying, don't if it's about to be born).

Requirements

Write a C++ program which implements the game of Life on at least a 18 x 50 board, making good use of the object-oriented features of C++.

<u>If you decide to do a console solution:</u>

Your program should ask the user for the number of organisms to be initially placed upon the game board - e.g.:

How many organisms initially? **6**

It should then read in the location of all organisms (given as row/column pairs - e.g. the following represents organisms at (1,1), (1, 3), (2,2) etc.):

Locations? 1 1 1 3 2 2 3 2 4 1 4 3

The program should then ask for the number of generations to be considered - e.g.:

Generations? 2

The program should then display the initial game board, prompt to continue, then clear the screen (see discussion below), update the board, and display the updated board. This will continue for each generation. Thus, for the data above we would have something like the following output:

```
      Initial:
      +------------------------------------------------+
|* *                                                   |
| *                                                    |
| *                                                    |
|* *                                                   |
|                                                      |
 etc...
+------------------------------------------------------+

Generation 1:
      +------------------------------------------------+
| *                                                    |
|***                                                   |
|***                                                   |
| *                                                    |
|                                                      |
 etc...
+------------------------------------------------------+

Generation 2:
      +------------------------------------------------+
|***                                                   |
|                                                      |
|                                                      |
|***                                                   |
|                                                      |
 etc...
+------------------------------------------------------+
```

Note: the above format for the display of the board is for example only. Feel free to be

creative as you design the output for your program. Your grade will partially be based upon the aesthetic appeal of the output.

Test your program with the data given above for 10 generations. Try the same pattern somewhere near the center of the world. Then, experiment with some additional initial configurations. Turn in the test cases you used (just the starting coordinates - not reams of output!) and the resulting configuration.

One demonstration version of this project is available for your use as you compare the output of your program to the "expected" output. To run the demonstration – ssh onto the Joshua server and then type the following at the terminal prompt:

/gc/cs212/p1_demo (command prompt demo program)

Implementation Notes

1. It will be convenient for the array to have an extra row at the top (row 0) and at the bottom (row 19), together with an extra column at the left (column 0) and at the right (column 51), to represent the border. (These squares will never contain organisms, but will be considered as neighboring squares when testing a cell to see whether it gestates or survives. This avoids having to write special case code for the cells next to the border) The "real" board itself will therefore consist of rows 1 .. 18 and columns 1 .. 50. You might include statements in your program like these:

```
static const int activeRows = 18;
static const int activeCols = 50;
static const int totalRows = activeRows + 2;
static const int totalCols = activeCols + 2;
```

2. Utilize an enumerated data type (enum) to represent the status of any location on the board. That is, you might include statements in your program like these:

```
enum Organism { NONE, GESTATING, LIVING, DYING };
Organism _board[totalRows][totalCols];
```

Note: see www.cprogramming.com/tutorial/**enum**.html for more information on enumeration.

3. For each iteration of the game, you will need to make two "sweeps" of the board - first to set the state of each cell to NONE, GESTATING, LIVING, or DYING based on its current state and number of neighboring organisms, and the second sweep to set all GESTATING cells to LIVING and DYING cells to NONE.

4. The following code should be used to flush the input buffer after reading input

from the user. " Immediately after reading all the input values from the user at startup:

```
while (cin.get() != '\n') ; // NOTE THE SEMICOLON!
```

(This keeps reading characters from standard input until the newline that the user typed to enter the line of input has been read. Failure to do this will result in this newline being interpreted as the first signal to continue after the initial screen is displayed, causing it to disappear immediately.)

5. The following code can be used to clear the screen before the initial board is displayed: "

At toplevel near the start of the file: **static const char ESC = 27;**

Just before displaying the initial board: **cout << ESC << "[H" << ESC << "[J" << "Initial:" << endl;**

6. The following code can be used to position the cursor to the top of the screen, so that each board overwrites the previous one, giving the appearance of animation. " Just before starting to display the board each time after the first: **cout << ESC << "[H" << "Generation " << some variable << ":" << endl;**

7. The following code can be used to prompt the user to press Return after each generation and wait for the user to do so. "

After each board has been displayed:

```
cout << ESC << "[23;1H" << ESC << "[K" << "Press RETURN to continue";

while (cin.get() != '\n') ; // NOTE THE SEMICOLON! (This displays the message "Press RETURN to continue" at the bottom of the screen, and then waits for the newline that the user types.)
```

Turn in (electronically – via cps212Dropbox)

1. **Coversheet and writeup**
2. **Softcopy of code file(s) and executable**

(see the CPS212 Submission Policy document on our website)

<u>Extra credit opportunity (+10)</u>

Use GUI programming (i.e. QT library) instead of using console programming.