HW3b - Game of Life

Submit Assignment

Due Friday by 11:59pm **Points** 100 **Submitting** a file upload **File Types** cpp, cxx, and cc
Available until Sep 28 at 11:59pm

The game of life is a finite automata invented by Conway. See: https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)

Create a class containing

1, an array L (or life) which is 2-dimensional.

2. a number of generations to compute

3. how often to print the field

Create a method allowing you to load initial states from a file. For every asterisk, the cell is alive, and space represents dead:

```
* **  *
 *   *
   *
```

Internally, I would recommend that either L be bool (true/false) or int. The reason int might be an easier representation is that you must count how many of your neighbors are alive, and the easy way to see how many neighbors you have for an element i,j would be:

L[i][j-1] + L[i][j+1] + L[i-1][j] + ...

Anyway, the code is a bit messy because at the edge, you must be careful not to ask for an element that is outside the array. If you do, you will get into memory that is not in the array and either crash (if you are lucky) or just get wrong results.

So the clever way to make your "life" easy is to create an array that is bigger than the grid you intend to use. If you want a grid that is 10x10, instead create an array L[12][12].

If you only process and print elements from 1 to 10, you will never go out of bounds.

Until we learn dynamic memory, we will need a fixed size, so the size of the grid will be 10x10. The input to this program should be a file called "hw3b.dat" (spelled exactly that way, with no path name) containing a number of generations to process, and how often to print them, followed by the initial array. Every cell that is live at the beginning is written with an asterisk "*" and every cell that is not alive is a space. For example:

```
2    1

***

**
**
```

should compute 2 generations, printing out every 1. If you take a look at the wikipedia article you can see that three cells in a row form an oscillator:

```
***
```

turns into:

```
 *
 *
 *
```

four cells like this:

```
**
**
```

are stable and stay the same because each one has exactly three live neighbors.

given the two numbers 100 10

you should compute 100 generations, printing every 10th one. so starting with generation 0

print 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100