

Programming Project 1

Conway's Game of Life

Organize the code:

Following what we discussed on multiple file etiquette take the single source file provided, and divide it into appropriate header files and implementation files, one pair of files for each class. Place the main routine in its own file named `main.cpp`. Make sure each file `#includes` the headers it needs. Each header file must have include guards. Only include header files when they are actually needed.

Now what about the global constants? Place them in their own header file named `globals.h`. And what about utility functions like `delay` or `clearScreen`? Place them in their own implementation file named `utils.cpp`, and place their prototype declarations in `globals.h`.

The first thing you should do is make sure that the single source compiles as is. Play around with it to get comfortable the program. The program implements Conway's Game of Life

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life, which is a simple simulation where cells are either alive or dead based simple rules based on under population or overcrowding.

Add a feature

When you run the program you should notice that the `<s>` options does nothing. Your job will be to implement this feature. Run the executable provided with this spec where the feature has been added to the game (For Windows : `CS20A_Project1_SOLN.exe`, a OSX executable will be added later)

When you press `<s>`, depending in how the current state evolved, you will see how many times a cell has been birthed, `'.'` indicates no change, `'A'` through `'Z'` indicated the number of times it has been birthed (corresponding to 1 through 26, capping at 26). We are not keeping track of deaths.

Your task is to implement this functionality. You will need to do the following:

- Define a class named `Stats` with the following public interface:

```
class Stats {
public:
    // Constructor/destructor
    Stats();
    ~Stats();

    // Accessors or Getters
    void display() const;

    // Mutators or Setters
    bool record(int r, int c, int val);

private:
    // TODO

};
```

- The constructor initializes the Stats's grid to correspond to the World's, however Stats keeps tracks of a count of births for each cell, whereas the World simply displays the cells' state, being represented by characters.
- The destructor cleans up the Stats object.
- The display member function prints the stats grid similar to what is seen in the example program.
- The record function updates the Stats grid at row *r* and column *c* by increasing that cell by *val*.

The class declaration (with any private members you choose to add to support your implementation) must be in a file named `Stats.h`, and the implementation of the Stats class's member functions must be in `Stats.cpp`. You must not add any other public members to the class. (This implies, for example, that you must not add a public default constructor.) The only member function of the History class that may write to `cout` is `Stats::display`.

Add a data member of type Stats (*not* of type pointer-to-Stats) to the World class, and provide this public function to access it; notice that it returns a *reference* to a Stats object.

```
class World {  
    ...  
    Stats& stats();  
    ...  
};
```

Whenever the world is updated we must check to see which cells have been birthed, those cells' position is then updated in the stats object by adding 1 with the `Stats::record` function. Note that if the cell's state has not changed or if it has died, we do not record anything.

The Game's `<s>` option should be completed to display the current stats grid. Press enter to continue. prompt and wait for the user to respond. (`cin.ignore(10000, '\n');` does that nicely.).

Turn in

By next meeting submit via canvas a zip file containing only the 15 files produced for this project:

<code>life.h</code>	<code>blinker.h</code>	<code>glider.h</code>	<code>world.h</code>	<code>game.h</code>	<code>stats.h</code>	<code>globals.h</code>	
<code>life.cpp</code>	<code>blinker.cpp</code>	<code>glider.cpp</code>	<code>world.cpp</code>	<code>game.cpp</code>	<code>stats.cpp</code>	<code>utils.cpp</code>	<code>main.cpp</code>

The zip file should be named `<lastname>_<id>.zip`.

If I take these 15 files, I must be able to compile them using VS2017 without any errors or warnings. Be sure to you do not introduce any compilation or link errors.

If you do not follow the requirements in the above paragraphs, your score on this project will be zero. "Do you mean that if I do everything right except misspell a file name or include an extra file or leave off one semicolon, I'll get no points whatsoever?" Yes. That seems harsh, but attention to detail is an important skill in this field. A draconian grading policy certainly encourages you to develop this skill.

The only exception to the requirement that the zip file contain exactly fifteen files of the indicated names is that if you create the zip file under macOS, it is acceptable if it contains the additional files that the macOS zip utility sometimes introduces: __MACOSX, .DS_Store, and names starting with ._ that contain your file names.