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, OSX: CS20A_Project1_SOLN.

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 simple displays the cells' state, being represented by characters.
- The destructor cleans up the Stats object. Similar to how the World cleans up after itself.
- 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 increased that cell by val. The record function should return false if the coordinates are out of bounds, and true otherwise.

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 Stats 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:

life.h	blinker.h	glider.h	world.h	game.h	stats.h	globals.h	
life.cpp	blinker.cpp	glider.cpp	world.cpp	game.cpp	stats.cpp	utils.cpp	main.cpp

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.

Additional guidelines and hints:

- If a .cpp file uses a class or function declared in a particular header file, then it should #include that header. The idea is that someone writing a .cpp file should not worry about which header files include other header files. For example, a .cpp file using an A object and a B object should include both A.h (where presumably the class A is declared) and B.h (where B is declared), without considering whether or not A.h includes B.h or vice versa.
- In your .h and .cpp files, for any #include of a header defining one of the classes in this project (e.g., #include "game.h"), if you can replace that header with an incomplete type declaration of the class it defines (e.g. class Game;) and all code that must compile does compile, and all code that must not compile does not compile, then you must do so.
- Except to add the member function World::stats or to change string to std::string, you must not make any additions or changes to the public interface of any of the classes. (You are free to make changes to the private members and to the implementations of the member functions.) The word friend and the word sequence pragma once must not appear in any of the files you submit.
- Your program must not use any global variables whose values may change during execution. Global constants (e.g. MAX_ROWS) are all right.
- Except in Game::play (or a function it calls) and Stats::display, your program must write no output to cout beyond what is already in the program or is required by this spec. If you want to print things out for debugging purposes, write to cerr instead of cout. When we test your program, we will cause everything written to cerr to be discarded instead — we will never see that output, so you may leave those debugging output statements in your program if you wish. Note that cerr is an output stream just like cout, just one for errors.

- If we replace your main.cpp file with the following, the program must build successfully:

```
#include "game.h"
#include "game.h"
#include "world.h"
#include "world.h"
#include "stats.h"
#include "stats.h"
#include "life.h"
#include "life.h"
#include "blinker.h"
#include "blinker.h"
#include "glider.h"
#include "glider.h"
#include "globals.h"
#include "globals.h"
int main()
{}
```

- If we replace your main.cpp file with the following, the program must build successfully

```
#include "stats.h"
int main()
{
	Stats s;
	s.record(1, 1,1);
	s.display();
}
```

`stats`.h must not contain any #include line that, if removed, still allows the above replacement main.cpp to compile successfully.

- If we replace your main.cpp file with the following, the program must build successfully

```
#include"blinker.h"

int main() {
	Blinker b(1,1);
}
```

`blinker`.h must not contain any #include line that, if removed, still allows the above replacement main.cpp to compile successfully.

- If we replace your main.cpp file with the following, the program must build successfully

```
#include"world.h"
int main() {
	World w;
	w.addLife(nullptr);
}
```

`world`.h must not contain any #include line that, if removed, still allows the above replacement main.cpp to compile successfully.

- If we replace your main.cpp file with the following, the program must build successfully

```
#include"world.h"
#include "blinker.h"
int main() {
	World w;
	Blinker b(2, 2);
	w.addLife(&b);
}
```

- If we replace your main.cpp file with the following, the program must build successfully

```
#include"world.h"
#include "blinker.h"
#include<iostream>
using namespace std;

int main() {
	int num_steps = 0;
	World w;
	Blinker b(2, 2);
	w.addLife(&b);

	do {
		w.update();
	} while (w.hasWorldChanged() && (num_steps++) < 10);
	w.stats().display();
	cout << "===" << endl;
}
```

When Executed we must see:

```
...........
...........
..E........
.F.F.......
..E........
...........
...........
...........
...........
...........
...........

===
```

- The following should display " OUT OF BOUNDS ".

```
#include "stats.h"
#include "globals.h"
#include<iostream>
using namespace std;
int main() {
	Stats s;
	if (!s.record(-1, (MAX_ROWS+1) , 0)) {
		cerr << " OUT OF BOUNDS " << endl;
	}
}
```

- If we replace your main.cpp file with one the following, the program must **NOT** build successfully:

```
#include "blinker.h"
int main() {

	World w;
	Blinker b(2, 2);

}
```

```
#include"world.h"
int main() {

	World w;
	Blinker b(2, 2);
}
```

```
#include "game.h"
#include "stats.h"
#include "life.h"
#include "blinker.h"
#include "glider.h"
#include "globals.h"

int main() {
	World w;
}
```

```
#include "stats.h"
#include "life.h"
#include "blinker.h"
#include "glider.h"
#include "world.h"
#include "globals.h"

int main() {

	Game g(nullptr, 0);
}
```

```
#include "stats.h"
#include "world.h"
#include "globals.h"
#include "game.h"
int main() {

	Life l;
}
```