Programming Project 1: Conway's Game of Life

Read everything carefully! Any violation to any item stated in this specification will result in penalties.

Overview:
The goal of the project is to organize the code according to the multiple file etiquette we discussed in class. This involves separating the single provided source into the specified files.

There might be many language details that are unfamiliar to you. That is completely fine, we will discuss them in time, and **you do not need to know anything about them to complete this project**. In addition to File Etiquette and Class Implementation, there are several aspects of programming that this project stresses.

- Working with unfamiliar code and being able to get a general understanding of the overall flow of execution.
- Figuring out what is being specified, and for what aspect of the project the given information relates. Read everything first and devise a plan, as you become more familiar with what you need to achieve modify the plan accordingly.

Part 1: Organize the code (85%):
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. When complete the main function will be the only function in `main.cpp`.

Make sure each file #includes the headers it needs. Each header file must have include guards. Only include header files when definitions are actually needed; declare objects in situations when the compiler only requires to "know" about the objects existence but not its definition.

Now what about the global constants? Place them in their own header file named `global.h`.

Note that there are two versions of clearScreen depending on platform, **do not** break up the preprocessor directives; keep both versions.

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://conwaylife.com/wiki/Conway's_Game_of_Life

This is a simple simulation where cells are either alive or dead based simple rules based on under population or overcrowding. For those who are feeling adventurous look for some Life patterns and try to implement them.

Part 2: Analysis (15%):
You should notice that the class Simulation has the function **report** does not do anything useful other than printing "Hello World!" to the console. Replace "Hello World!" with a brief discussion addressing the questions below. This may require stepping through the code to get an overview of execution, hand tracing is an option but using the debugger with break points is much more efficient.

Consider the following 3 points, examine, and answer the 3 questions accompanying each point.

- Learn to use your debugger, this can be done before separating into individual files. Note that all enrolled students have access to Visual Studio 2019 through SMC Virtual Lab: https://www.smc.edu/administration/information-technology/vcl/

  Place a breakpoint in the function Simulation::one(int t) at the line int k = watchme; In the original provided source it will be line 457. From there you should be able to identify the values for the member variables for the Simulation object. In particular, note that the starting value for Simulation:: watchme is 1. Step through the rest of the code and make note each time Simulation::watchme changes in value. Do this until you find yourself in the main function. In VS2019 this occurs in about 40 steps. You should find that Simulation::watchme changed four times; so it took on total of five different values including the starting 1. These values are part of a five digit number with the first, 1, being the least significant digit.

  Once you have identified the entire five digit number, pass it into the decode function provided in the short program provided in decode.cpp(create a separate project to run this). What is the decoded sequence of characters? If the decoded sequence does not make sense, then you may have entered the sequence into the decoder in the incorrect order. There should be no duplicate values.

- After separating into individual files: If we replace your main.cpp file with the following, the program must build successfully. Note there are **no mistakes** here, this is a valid test case. If this main ends up not compiling, what might be the underlying issue?

```
#include "simulation.h"
#include "simulation.h"
#include "matrix.h"
#include "matrix.h"
#include "life.h"
#include "life.h"
#include "box.h"
#include "box.h"
#include "snake.h"
#include "snake.h"
#include "bigblink.h"
#include "bigblink.h"
#include "boat.h"
#include "boat.h"
#include "global.h"
#include "global.h"
int main(){}
```

- After separating into individual files: If we replace your main.cpp file with the following, the program must **not** build successfully. If this main results in a successful build, why might that be?

```
#include"simulation.h"

int main() {
  Matrix m;
  Simulation s(nullptr, 0);
}
```

Other than in the function report and necessary inclusions or declarations, do not make any changes that would modify the behavior of the program from what was originally given. Do not modify any of the identifiers used for classes, functions and variables.

Submission:

Submit via canvas one zip file containing **only** the 16 source files produced for this project, the files names and **extensions** must be exactly as given below:

life.h	box.h	snake.h	boat.h	bigblink.h	matrix.h	simulation.h	global.h
life.cpp	box.cpp	snake.cpp	boat.cpp	bigblink.cpp	matrix.cpp	simulation.cpp	main.cpp

Any code you submit must compile in VS2019 or newer, even if incomplete; any project that does not compile will receive a zero score.

The directive *pragma once* **must not** appear anywhere in anything you submit, for this or any other project for this course.

Only submit the files specified; do not include any project/IDE specific files. The only exception to this is any files generated in the compression process, notably on Mac OS.

Additional Considerations

- **Work incrementally/iteratively**, this is life advice for all programming in all situations. Making large sweeping changes to any code base without validating is the extremely counterproductive. For this project specifically, start with just moving only one class definition into its own header file, reconcile what needs to be reconciled and compile. If successful, only then go onto moving more code, e.g. that class' implementation, or another class' definition into its header file, rinse/repeat.
- Compile often and submit regularly, your previous submissions will be overwritten on canvas. What you submit must compile, incomplete code that works is always better than any code that does not compile regardless of how "correct" most of the implementations are; You don't get paid for broken code.
- Start with making sure the original source compiles, then spend some time playing around with the code to try to get a sense of what is going on. Feel free to go cray, you can always re-download the source. Start at the "top", the main function. See what objects are created and then take a look at those objects' constructor/member function calls. It is ok if you don't completely understand what is going on, there is some inheritance action going which we'll discuss in detail later.