Author: Benjamin R. Olson
Date: April 13, 2014
Class: cs162 - Introduction to Computer Science II
Description: Project 1 Reflection

****
Reflection:

Process if implementation:
I started by writing the function calls in main, before defining them.
After setting up the files and blank function implementations,
I wrote a function, then tested it, wrote another function, then tested it,
and so on. Half-way through, the program as a whole became too difficult
to focus on, so I began a scaled-down version, a single main.cpp file,
to focus on some of the fine details of logic, and then periodically, once
mastered in the small version, went back to the larger, complete version
to implement the same thing (conceptually) that I had learned from the smaller version.

While implementing the scaled-down version, for edge cases I decided not to
calculate "ghost" cells because I count them dead anyway and the purpose
of that particular function was counting the neighbors only to find how many
neighbors are live; counting how many cells are outside the grin boundaries
was an unnecessary and unused calculation for my purposes. Also, instead of
using a neighbors array to store the values of each neighbor (to a given cell),
I decided this array was also unnecessary in favor of simply checking each
neighbor to see if is live and if so, adding it to a counter "on the spot".
Nevertheless, the neighbors array helped me, while testing, to see that I was
navigating to the correct neighbor cells of a given cell.

I started to notice that one of my functions required much more coding
than I had expected: int countLive(...), which counts the number of live
cells surrounding a certain cell. To reduce code duplication, maybe
I could have written a separate function where each function gets
one of the eight possible neighbor cells.

One of the most difficult errors to debug was that I was
running into a segmentation fault just before main returned. I realized
the cause was that there were some variables in generation.cpp
that were not in the corresponding generation.hpp.

Discovery during implementation:
I changed my approach half-way through the implementation. Instead of
(1) starting with separate files and empty function bodies and filling
these out little by little, the implementation process may have gone faster
had I (2) started writing everything in main.cpp and extract into functions
as I go. I'm still undecided as to whether approach (1) or (2) is better,
but I believe we're being encouraged to do something more similar to
approach (1). I may keep this in mind from project to project.
In any case, stepwise refinement can be applied in both approaches.

****
Tests durning implementation (see file proj1_tests.pdf):

Test group 1: using gdb to navigate test1.cpp
Test group 2: using gdb to navigate test2.cpp

Tests after implementation:

This was simply running the final product and visually inspecting.

Testing mainly took place during implementation, with a combination of
cout statements and gdb: function calls and printing out the values
of variables.

****
Differences from and additions to original design:

I needed to swap the order in which two functions were called in main
(displayGrid() and calcGeneration()).
I added function continueGeneration(), which was not in the original design,
to prompt the user to either display the next grid or return to the main menu.
In addition to main.cpp, I was debating about whether to have two files () -
one for the funcitons and one to implement the data structure, or one file to contain both.
I decided on just one class file in order privatize the data and give public functions direct
access to the data.
The one function I implemented in main.cpp I did so because it was not
really conceptually part of the object instantiated from the Generation class.
In other words, I decided it should not be a part of the Generation class
because it does not deal with changing the state of the Generation object,
but only deals with deciding what configuration will be initialized in the object.
I believe the greatest difference between design and implementaiton was this:
I decided not to pass arguments to most of the functions within the Generation class.
This made it easier to call the object's functions and allowed the object to do its work
more independently from the caller.