Benjamin Tate
1/17/16
CS 162 – Section 400
Assignment 1 Reflection

Design:

My design process was done (quite messily) on paper, which makes it hard to replicate on a computer, but below is my best approximation, with the pseudocode cleaned up to represent the order in which I planned to implement my ideas, rather than the order in which I thought of them.

Initialize enumerator for starting pattern (possible values: oscillator, glider, and gun)

<u>Array builder/generation increment functions:</u>

Create copy() void function to copy contents of one matrix to another (two 2D arrays of ints as parameters):

```
    Iterate through rows:
        Iterate through columns:
            Arr1[row][column] = arr2[row][column]
```

Initialize 76x30 array of zeroes

Create buildStart() void function to build initial pattern (pattern, and 2 ints for starting coordinates as parameters)

76x30 was chosen as the dimensions, because 5 extra rows on the top and bottom, and 18 extra columns on each side is big enough to allow the center of the glider gun (the biggest starting pattern) to be started on any corner. The 40x20 window will range from [5][18] to [24][57].

```
    If pattern == oscillator:
        Array[r][c] = 1;
        Array[r][c-1] = 1;
        Array[r][c+1] = 1
    If pattern == glider:
        *build glider*
    If pattern == gun:
        *build glider gun*
```

Create nextGen() void function to update array instantaneously each generation (76x30 array of ints as parameter)

```
    Loop through each cell and determine # of neighbors:
        Initialize int count = 0
        Count = [i-1][j-1] + [i-1][j] + [i-1][j+1] +
                [i][j-1] + [i][j+1] +
                [i+1][j-1] + [i+1][j] + [i+1][j+1]
    Initialize 76x30 array of zeroes “temp” to hold changes for next generation
    Iterate through each cell and determine if it stays the same:
```

If array[i][j] = 1:
    If # neighbors <= 1 or >3:
        Does not stay the same.
Else if array[i][j] = 0:
    If # neighbors = 3:
        Does not stay the same.
If cell stays the same:
    Temp[i][j] = array[i][j]
Else:
    If array[i][j] = 1:
        Temp[i][j] = 0
    Else if array[i][j] = 0:
        Temp[i][j] = 1

Copy contents of temp to array using copy() function

Main function:

Initialize ints: generations, startRow, startCol
Initialize Pattern enumerator

Prompt user for starting pattern (oscillator, glider, or gun)
Store user response as Pattern

Prompt user for starting coordinates
Store row in startRow
Store column in startCol

Prompt user for number of generations to run simulation for
Store in generations

Create matrix with pattern at starting location, using buildStart()

Print starting (1st gen) matrix, row by row:
    Top border (|-|-|-| etc.)
    Iterate through visible rows:
        Left border ("|")
        Iterate through visible columns:
            Array[i][j] << "|" (each cell and right border)
        Bottom border for each row

Initialize count = 0
While count < generations:
    Call nextGen()

    Print updated matrix, row by row

Count++

Reflection:

Overall, my final code resembles my initial design pretty well, though there were some changes that needed to be made. Some were largely cosmetic (such as rearranging the order of function definitions to reflect the order in which they would be called, to make the program easier to follow), but others were a result of me failing to realize certain aspects of what would be required. For example, I realized about halfway through writing the program that it would be far simpler to use a class to handle the matrices. That way, my one file wouldn't get so cluttered with functions. This also allowed me to more easily use variables in multiple functions. Also, I realized at some point that when prompting the user for a starting coordinate, I would have to translate that from a reference to the 40x20 window that they would be aware of to a coordinate within the 76x30 matrix that I would actually be using. That is why I used windowRow/Col in addition to startRow/Col in my final product.

Other changes stemmed primarily from the fact that I decided to use a class to handle the matrices. This meant that I could do a lot of the function calling within the Life constructor, which meant that parameters of some functions changed (for example, nextGen() no longer required parameters, as it already had access to the array that I was planning to make its parameter). I also had to add getCellVal() to allow access to the values of each cell of my main array.

Other than that, I added a setNeighbors() function, which created another array called neighborCount that would keep track of neighboring, living cells for each cell (occupied or not) in the main matrix. That provided an easier way to keep track of neighbors in the nextGen() function. I also found during testing that printing my tables full of zeroes for unoccupied cells and ones for occupied ones, was incredibly difficult to read, so I added if statements in my main function to print a space if the cell was empty, and an X otherwise.

Once I got the program compiling, which only required simple syntax and typo corrections, testing was fairly straightforward. I first made sure that the oscillator was working by putting it towards the middle of the window and running it for 5 or so generations, and when I saw that it was, I went back in and "drew" the glider and gun patterns into my buildStart function, and ran separate tests for each of them. At first, it seemed as though the gun wasn't working, but after a bit of puzzling over it and researching, I realized that I had just underestimated how long it took to "shoot" a glider out, so I went back and ran it for about 50 generations, which was enough time to see that it was in fact working well. Next I tested each from close to a corner, making sure that they still looked right, and that gliders would just disappear off the edge, and it all worked perfectly, so I was finished!