Aryan Aziz

Assignment 13 Report

Exercises

List of Files

A13E1.cpp
A13E2.cpp
A13E3.cpp
A13E4.cpp
A13P1.cpp
input.txt
input2.txt
output.txt

Projects

Understanding

I think this week's understanding was pretty straightforward for the exercises. With Conway's game of life being the biggest thing we need to do, the other exercises didn't see too bad.

The first exercise wanted a logging system that took in visitors names and nothing else, and allowed you to search through the log. I didn't see anything about it having to go into a file though, so I just kept it to a vector of strings.

The second exercise wanted us to take a data set and get the median and both quartiles.

The third exercise wanted us to take in 2 lists (in this case, ordered), grab all the data from the list, and then output the data into a third file fully sorted.

The fourth exercise wanted us to work on a way of error handling to ensure we had the integer needed for a function.

And finally our project wanted us to focus on Conway and his game of life.

Design

Honestly not too much design went into this week's assignments. I attached to this PDF my hand written notes for exercises 1-4, however they all seemed pretty self explanatory on the workflow and the needs so I did not have much pre-programming design.

As for Conway, I relied heavily on my design from last week for my implementation from this week.

Testing
I did basic testing for all of my programs to ensure they were working correctly, however for this week most of my focus was on the project and most of my testing was there. I caught myself, for the most part, actually testing my inputs/outputs rather than the program itself. I find myself compiling and running a test roughly every 5 minutes or so to test something else out.

Something I need to work on is a proper testing plan rather than this test on the fly approach. There are a lot of times that I catch myself missing a test until I think the program is completed, or really just testing random things as I go along. The compiling every 5 minute approach is definitely not ideal.

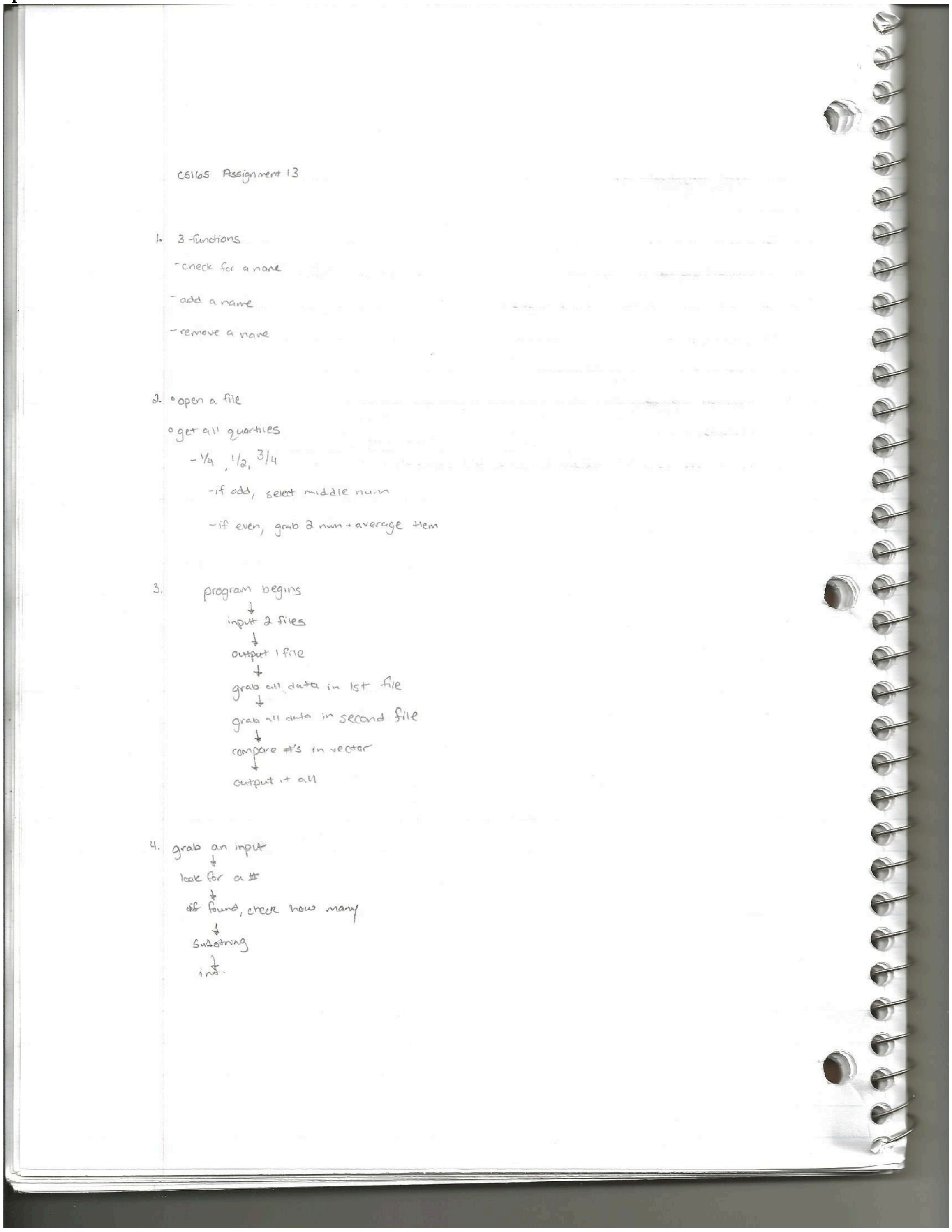
Reflection
The game of life program ended up being a lot more time consuming and labor intensive that I had imagined. Unfortunately between our midterm this week and my other midterm this week (as well as family obligations) I did not have as much time to put into the assignment as I wanted to, but I caught myself stressing to get the game of life completed.

Because of this, I didn't get to add all of the checks and features I wanted, however I (think) I've got it all running and smoothly. It definitely isn't the cleanest approach but it got the job done.

In terms of sticking to my original design, I kept pretty close to it. I think there were 3 major changes:

1. The ability to randomly generate a board. I realized that there will be more cells live than previously anticipated and adding them 1 at a time is annoying.
2. The copy array function. Originally I was just going to make this a part of my runGeneration program (so that I could make changes on the array and not effect other cells). However, I realized it would be easier since I would need to replicate the array and then place the replicated one back into the spot to just make a general function and pass the 2 arrays in whatever order I needed.
3. Switching my array from a character to an int. I found in doing the calculation for the neighbor it's a lot easier to have an integer array of 0's and 1's and use that to calculate the number of neighbors instead of doing a loop through each of the neighbor and determining if the neighbor was a specific character and then adding to a counter.
    a. This may have not been the proper way to get it done, but I thought it was a pretty elegant and clever solution. (Tooting my own horn a bit).

I also caught myself hanging on the array[value][value] and what value should be passed where a lot. I had to bring out the pen and paper and get my bearings on it quite a few times.



CS165 Assignment 13

1. 3 functions
   - check for a name
   - add a name
   - remove a name

2. ◦ open a file
   ◦ get all quartiles
     - 1/4, 1/2, 3/4
       - if odd, select middle num
       - if even, grab 2 num + average them

3. program begins
   ↓
   input 2 files
   ↓
   output 1 file
   ↓
   grab all data in 1st file
   ↓
   grab all data in second file
   ↓
   compare #'s in vector
   ↓
   output it all

4. grab an input
   ↓
   look for a #
   ↓
   ~~if~~ found, check how many
   ↓
   substring
   ↓
   int.