CS162 – Project 1

Khoa Phan

4/16/17

Project Reflection

The Langston's Ant project was an interesting coding project to simulate a set of rules that must be followed in order to achieve a working code. There were many ways to approach my design. At first, I used a simple flow-chart. That became too messy once I realized the extent of each action I needed to account for. I then decided to use pseudocode to break down each function that was necessary. Here is the pseudocode I started with in order to help me with organizing my ideas.

```
/****

class menu(switch statement)
        1. play/start
        2. quit

class board(create board)
        default;
                prompt user for # of rows & cols & num of steps
                        data validation

        call to play/start
                create new dynamic array for board
                        createBoard();
                call ant class
                delete array

        # of steps
                prompt user for # of steps
                        data validation
                suggest #

        print board langstons ant
                display board (array)
                loop through boards, clear screen/step(?)
                for loop until counter == # of steps

class inputValidation
        input as string? and convert to int.
        if not, return message, not valid

class ant(pointer to board array)
        Ant class 'a';
        enum for N, S, E, W;
        variables for white/black spaces
```

```
        x, y coordinates member variables
        prompt user enter starting location
                xCoord and yCoord
                        data validation
        find coordinate from array location, write @
                default facing N.
                switch statement for N, S, E, W
                        write data into array, white square
                                turn right 90 degrees, change to black (#) move
                        write data into array, black square
                                turn left 90 degrees, change to white (empty space) move
                        for each direction
        wrap ant around board -> opposite side if collision

int main()
        establish variables, welcome user
        call menu class



***/
```

First was the menu class. I used a simple switch receiving a parameter input to then decide the switch case. I really do want to revise this a bit more to be able to be more dynamic. This led to my next idea of creation of the board. I wanted to establish the board the ant will be walking on before any other actions were taking. I had designed around making the board re-print itself to update the location instead of a "real-time" board that updates within the same screen. This was done as a way to be able to document each step, making sure I was getting the desired results.

Once the board had been established, the next part was to define the ant class, which will define its start location and the properties of the ant and the spaces it is on. This part in particular was the most time-consuming. Throughout the process, I needed to add more little detail for certain parts of code I did not initially account for. One of these being how the ant acts when it is facing a certain direction. Making sure I am reading the directions carefully was the first thing I encountered. At first, I had thought the ant was to move forward first then turn according the *new* location it was at, when in fact it was not. This changed around my design a bit as I had to set variables to and functions to work with the current location.

Here is what I had done initially in pseudocode:

Case NORTH:

        Move ant, (x, y-1), if(x, y-1) = B…antDir = WEST

                If(x, y-1) = W…antDir = EAST

This is incorrect so I did something like this:

Case NORTH:

        antLoc = (x, y), if (x,y) = B… antDir = WEST, move ant (x-1, y)

if (x,y) = B… antDir = EAST, move ant (x+1, y)

I had used online Langston's ant simulators to test my results to working programs. By using it, I was able to see where I had gone wrong and why. My testing plan was consistent. The easiest identifier for my results was any relative size grid, like 30x30 with 52 steps. This created a rotated heart shape that was easily identifiable.

Once this was working as intended, wall collision was the next step I needed to figure out. I thought that wrapping the ant around to the other side was the best approach to test to make sure my functions were working correctly. Here, I learned the value to understanding the array values and variable sizes. By knowing how the syntax works, the wrapping part became very clear on how I needed to approach it.

Next was validation. I had found a resource online that I cited in the code which used a similar input type. Mainly, this program accepts integers as a means of information. Doing so, the code took only doubles as a number, ignoring any other input then converting it to an integer. (I could possible just accept integers to begin with, but for testing purposes, I continued to use doubles). It also established a range function which would be used to help identify limits with the dynamic array.

Testing results were as followed:

INPUT (menu option 1 (start), prompt size of grid, number of steps, start location)	OUTPUT (result using 52 steps as a base, in description form)
1 -> 20, 20-> 52 -> 10, 10	Heart shape rotated 90 degrees
1 -> 20, 20-> 52 -> 20, 20	Heart shape, with black spaces between top and bottom of the board, correctly continuing the ant's path
1 -> 20, 20-> 52 -> 15, 20	
1-> -1, -15	Invalid as a result of negative grid size not allowed
1 -> 10, 10 -> -52	Invalid as a result of negative steps not allowed
2 ->	Quits program
1 -> 20, 20 -> 52 -> -10, -10	Invalid as a result of no negative numbers allowed
A, @, ., alkj12, 3…3, etc..	Invalid input, this applied to any location where user input is required. It does not allow for non-number input

These test allowed me to debug a lot of issues. The result is a working Langston's Ant program.

My experience throughout this project was positive, as I had learned a lot more on how to use pointers, functions, dynamic arrays and memory deallocation. Other than code, I had learned a lot to make sure my design intent is on the right track with better pseudocode and documentation. Most of this reflection are based off of notes I had taken while working, although I could improve a lot in recording more. There were points in the code that were very un-organized due to the fact that changing the functionality of the functions meant that I needed to fix something rather quickly due to the time constraints I had. Looking forward, I want to improve my process a bit more but overall the project was definitely a positive.