Anthony Clemente
CS 162
Project 1
16 April 2017

DESIGN

Program main steps:

-Establish 2D array of chars to represent cells
-Place ant at appropriate location
-Iterate though desired number of steps
    -For each step, change ant to appropriate position and change color (char) of the proper index
    -Display current array (board) to user
-Display final array after number of appropriate steps have been taken

Key Points:
-Prompts to user should have attached input validation
-Board size should likely be limited (max width of 80 is probably good)
-Program should display a menu giving user choices
-2D array must be dynamically allocated and deallocated (thus preventing memory leaks)
-Need to prevent ant from exiting board by using some consistent rule

Classes program will need:

-Ant
    -Will control the allocation and deallocation of 2D array
    -Will set indices of the array to appropriate chars
    -Keep track of all relevant movement information (direction facing, previous color of index, etc)

-Menu
    -Will display options to user
    -Check if option is a valid one

-InputValidation
    -Will hold functions that validate user input

Pseudocode/Program Logic Flow:

Main Function:

```
-Display menu with options to user
        Options will include:
                1.  Choose Ant starting position
                2.  Random ant starting position
                3.  Quit Program
```

```
If option 1 chosen
        -Declare two integer variables representing rows and columns of the 2D array

        -Prompt user for desired values (rows and columns)
        -Input function will guarantee an integer is entered (causing re-prompt if necessary)


        -Declare integer for desired steps

        -Prompt user for desired values (desired steps)
        -Input function will guarantee an integer is entered (causing re-prompt if necessary)


        -Declare integers for Ant starting coordinate

        -Prompt user for desired values (ant starting position)
        -Input function will guarantee an integer is entered (causing re-prompt if necessary)

        -Create Ant object with the chosen rows, columns, steps, and starting location for the
        constructor

If option 2 chosen
        -Declare two integer variables representing rows and columns of the 2D array

        -Prompt user for desired values (rows and columns)
        -Input function will guarantee an integer is entered (causing re-prompt if necessary)


        -Declare integer for desired steps

        -Prompt user for desired values (desired steps)
        -Input function will guarantee an integer is entered (causing re-prompt if necessary)

        -Create Ant object with the chosen rows, columns, and steps. (this will cause constructor to
        place ant randomly)


Ant Constructor:

        -Set all member variables appropriately with constructor inputs if possible
        -Other variables will be initialized to default values (ant current spot color, direction facing,
        current step)

        -Dynamically allocate a 2D array of chars of appropriate size
        -Set each char in the array to the space character to start
```

```
    -Using user values (if provided) set ant starting position by changing correct index to '@'
    -Otherwise this is set randomly
```

Back in Main function:

```
    -Using a member function print out the initial board to the user

    -Set up a while loop with the condition being a function that returns if a step is taken. Function
    will return false if the desired number of steps has already been reached
        -For each step that is taken print out the board
```

Ant Step Function:

```
    -First check if the number of steps ant has taken is greater than or equal to the desired number
    of steps

        -If it is return false

    -Use a switch value to check the direction ant is facing (direction will be an enum)

        -For each case use if statement to check color of square ant is on

            -if ant is on a white square (' ')

                -Turn ant to right (adjust direction accordingly)

                -Change current square to black ('#')

                -Check the next square to move to and record its color

                -Update the ant's position and fill the correct square with '@' symbol

            -else if ant is on a black square ('#')

                -Turn ant to left (adjust direction accordingly)

                -Change current square to white (' ')

                -Check the next square to move to and record its color

                -Update the ant's position and fill the correct square with '@' symbol

    -If the ant will attempt to step out of bounds instead wrap around to the other side of the board

        -If moving to the right of array (current column + 1 >= total columns)

            -Set current column to 0

        -If moving to the left of array (current column == 0)

            -Set current column to total columns minus 1

        -If moving above the array (current row == 0)
```

```
                -Set current rows to total rows minus 1
        -If moving down out of the array (current row + 1 >= total rows)
                -Set current row to 0


    -Increment the number of steps
    -Return true


Back in main function:
    -Once step function returns false show menu to user
    -Continue program until the user chooses to exit
```

TESTING

Input	Expected Output
Menu Option: “1” Rows: “10” Columns: “10” Desired Steps: 5 Starting Row: 4 Starting Column: 4	@ # ##
Menu Option: “2” Rows: ”10” Columns: “10” Desired steps: 5	(same shape as above but may not be near center of board since start is random)
Menu Option: “1” Rows: “10” Columns: “10” Desired Steps: “5” Starting Row: “9” Starting Column: “9”	# # # @
Menu Option: “3”	(program exits immediately)

REFLECTIONS

In 161, we were introduced to the idea of writing a pseudocode outline before writing our actual program. The example in the first course was a very short program all contained within the main function, so it was easy to sit down and start with the first line and proceed through the program until the end, writing what the code would do. I found that a little more challenging with this program since I wondered what to do when I reached a function call that would take the program out of main. I wondered if I should just insert the pseudocode related to this function at that spot, or if everything in the design should be laid out separately and then the main function pseudocode could be described. I'm not sure I necessarily set it up "correctly" but I decided to use the former method since I decided there wouldn't be too many "jumps" out of main. I think an issue I had was because this was a bigger program, I was very anxious to start coding and as a result my initial design was not as clear and as complete as it should have been, even if this would have helped with writing the actual program more.

There were several things that changed from my design to my actual code. One of these things was, I decided to make only one ant constructor instead of 2. I originally had planned to make two versions of the constructor one where 5 values (including the ant starting row and column) were used, and one where only 3 values were used. The version with only three values was going to automatically set the starting position of the ant randomly. I decided to simplify the class and make only one constructor and instead make sure in main, that the values given to the constructor were random for these values.

For the ant step function, with the switch statement with each case having the same basic if statement, I realized that as I wrote it out there was a lot of code written multiple times. This was because no matter if the ant was facing north or south to start that step, if it turned east the movement was identical. Because there was repeated code I decided to separate this out into 4 "sub functions" each controlling movement in one of the directions. This made the ant step function a little cleaner looking and without a bunch of repeat code which was instead condensed into the functions that were called. I still believe there is huge room for improvement with this code in these functions as I feel they are the core of this simulation, but as it is now I feel it is better than how it was originally written.

In the main function, I realized that if I wanted the ability for multiple simulations to be run without exiting the program, I had to set up a while loop that would keep running until the user selected "3". The initial set up in the design with an if statement didn't make any sense for this!

Also, to prevent more rewritten code and handle the case where the user wanted a random ant starting position, I decided to move most of this code to a function called "playGame" that takes a boolean as a parameter to indicate a user choice for ant starting location, or random starting location. This made the main function much cleaner and more understandable I feel.

In the end, I am mostly pleased with how the program ended up but feel like I needed to be more careful in the design stage which would have avoided some of the issues I listed above. I was too quick to want to jump into the programming to see results. The material from week 2 is helping me get a better grasp on the design aspect, and show how taking the time up front to carefully plan you code can pay off in the end.