David An
Project 1: Design+Reflection

Design: The main goal of the program was to create a simulation of Langton's Ant. The program is responsive, meaning that it requires user input in order to successfully set-up and complete the program. This means that I will need to incorporate some sort of input validation that can check to see if the user's input is able to work in my program. Since there will be multiple instances where the user will be prompted for an input, I will need to create a re-usable method that can complete this input validation. I will also need to create re-usable menu functions that prompt the user to begin the game or play again once it has finished, per the project specifications. After prompting the user for the necessary properties to begin the game, I must first create the board (using a dynamic 2D array), and place the Ant object in the corresponding coordinates. This means I'll need to create an Ant class that keeps track of its current location on the board, its direction/orientation, and the color of the square that it's on. I'll then need to initiate the simulation by creating a loop that processes for every step the user has entered. Within this loop, I'll need to check first what color the current square is, since that affects the direction the Ant object will face. Then I'll move the Ant according to the rules of Langton's Ant, making sure to wrap the Ant around the board if it attempts to move outside its bounds. Before I move it, I'll need to flip the color of the square the Ant was just on. I'll then print the board at the end of the loop, and once the loop finishes I'll delete the dynamically allocated memory and use one of the menu functions to prompt the user to see if whether or not they would like to replay or quit the simulation.

Test Case	Input Values	Functions	Expected	Observed
Input too low	Input < 1	Main() inputValidation()	re-prompt for new entry	re-prompt for new entry
Input in correct range	1 < Input < Max (depends on prompt)	^	inputValidation returns input back to Main	inputValidation returns input back to Main
Input extreme low	Input = 1	^	inputValidation returns input back to Main	inputValidation returns input back to Main
Input extreme high	Input = Max	^	inputValidation returns input back to Main	inputValidation returns input back to Main
Input too high	Input > Max	^	Re-prompt for new entry	Re-prompt for new entry
Input is incorrect type	Input = String	^	Re-prompt for new entry	Re-prompt for new entry

Reflection: Creating the inputValidation method was difficult, as I was not able to account for different types at first. In order to account for floats, I tried to convert the user input into a string and then check to see if the required string and the user strings were equivalent. However, that did not work for instances where a 1 was required and the user input was 12 since once converted to a string the program accepted 12 as equivalent to 1. In the end, I decided to tell the user that decimal values that are passed are truncated and used by the program, so that the inputValidation method could instead focus on values that were out of range or values that were of the wrong type.

The menu function used to initiate the game was easy, but I had to figure out how I would design my main function so that I could restart my code if the user wanted to replay the simulation. I ended up using a do while loop since the code would be used at least once (if the user prompted it to), and this way the code could be re-used if the user wanted to play the simulation again.

The most difficult part of this project was figuring out the logic of checking where the Ant was on the board, and what its next move must be. At first, I designed this portion so that it counted the starting position as already being on a white square, so instead of moving north to begin (the Ant's default direction was north), the Ant would turn to right and move forward from there. This resulted in a different output than what was given, so I had to include a check to see if the program was processing the first step so that I could initiate the simulation by moving the Ant north first.

Overall, this project would have been much more difficult if I had chosen to begin coding without outlining the design of the program first. By doing so, I was able to write code according to the design, and iteratively test it to make sure that it was working the way I wanted it to. I also really enjoyed writing the inputValidation method, since it's an important practical skill to be able to check a user's input so that it won't crash your program. I know we'll be incorporating input validation in other projects as well, so it was good to learn how to implement that early on. This was also the first project that was big enough to require some sort of modularization to clean up the logic. I would have liked to write some methods that would have moved the Ant on the board so that I didn't have to re-use code, but given the time constraints I could not make that happen unfortunately. It was definitely mistake starting this project so late – I vastly underestimated how much time it would take to design, code, and test the program. The most important thing I learned from this project is to budget my time wisely and get started sooner rather than later.