Design:

For the Langton's Ant assignment I'll need to communicate with the user, validate their input, and then use that input to perform a Langton's ant simulation. It's also required that there be separate files with reusable functions for both the data validation and the running of the simulation along with its user input needs.

So, I believe I'll need a main file, just for running the program and that should be about it. After that, I think it would make the most sense to make a LangtonsAnt class which will manage the simulation itself, including attributes, and all of the functions required to run the simulation. It should be able to run on its own in a default state but also able to accept user input for running modified versions of the simulation. The LangtonsAnt class will need an attribute to act as a sentinel for the main file to check after each iteration before running it again.

The LangtonsAnt class, will run a menu via the menuShell() functions with its constructor function which will request input from the user to determine how it should run. If the user chooses to make a custom simulation, then it should also use the menu functions to request additional information for each attribute that needs to be defined. All user input for the menuShell() functions should be run through isValidInput() to avoid breaking things.

Once all user input is received, the LangtonsAnt program should adjust its attributes, define the simulation's 2D array, and run until it has iterated the number of times provided by its default attributes or the user.

For each iteration (or step for the Ant) LangtonsAnt should check the validity of the move forward of where it is facing and if it is not a valid space to move to turn 180 degrees and reassess without iterating the number of steps taken. If the space to move forward to is valid, then move to that space, turn the ant based on the color of the space, swap the color of the space, and iterate the number of steps taken. Then print the state of the board out for the user.

Basic outline:

- Class for each ant on a grid
    - Variables (all private)
        - Rows & columns
            - (Default, Random, User)
            - Add 2 to user input to compensate for edges
        - Steps to do
            - (Default, Random, User)
        - Direction facing
            - (Default, Random, User)
            - Use an enum
        - Mode running in
            - (Default, Random, User)
            - Use an enum
        - Steps completed
        - Starting Coordinates

        - Number of failed steps (for debugging)
        - Menu strings
            - Welcome to Phillip Wellheuser's Langton's Ant Program
            - How would you like to run the program?
                - Default Mode: All values used are preset.
                - Custom Mode: Number of rows, columns, steps to do, starting coordinates, and the direction the ant starts facing are chosen by the user
                - Random Mode: All variables are set within a predetermined random range.
            - Others as needed for each type of user input
    - Main
        - Quit
        - Start
            - Creates a new langtonsAnt
    - langtonsAnt() --constructor
        - setAntParameters()
            - Rows & columns
                - (Default, Random, User)
                - Add 2 to user input to compensate for edges
            - Steps to do
                - (Default, Random, User)
            - Direction facing
                - (Default, Random, User)
                - Use an enum
            - Mode running in
                - (Default, Random, User)
                - Use an enum
            - Steps completed
            - Starting Coordinates
            - Number of failed steps (for debugging)
            - Menu strings
            - while(stepsCompleted<stepsToTake)
                - moveAnt()
                - printGrid()
        - moveAnt()
            - If the ant is on a **white space**, turn **right** 90 degrees and change the **space to black**.
            - If the ant is on a **black space**, turn **left** 90 degrees and change the **space to white.**
            - If hitting an edge turn 180
        - initGrid()
            - Create a dynamically allocated 2D array
            - Size input +2 (for borders
            - Start all white except for edges and ant

        - printGrid()
            - Black tiles
            - White tiles
            - Horizontal borders
            - Vertical borders
    - ~langtonsAnt()
        - Displays an end message
        - Deallocates all space for the 2D array
- isValidInput()
    - Validates whatever kind of input I need to throw at it
    - Receives:
        - Type of var input should be
        - Actual input
- menuShell()
    - Runs several types of menus with options set through parameters
    - Uses isValidInput() to validate user input

Tests

LangtonsAnt Class			
Test Cases	**Input**	**Expected Outcomes**	**Observed Outcomes**
Face into every corner both ways on both colored tiles to see if it breaks when stepping forward	Each corner facing into each edge	Ant should turn 180 degrees and continue on its way	As expected
Does the ant move in the correct direction initially?	Set ant to facing each direction and run 1 step to see if it moves in that direction.	Ant should step once in the direction it is facing and leave the prior tile as black	As expected
Crash after 10,000 steps	10,000 steps, 40x50 grid, facing any	No crashes after all iterations	As expected
Crash after 100,000 steps	10,000 steps, 40x50 grid, facing any	No crashes after all iterations	As expected
Does the program run for non-square grids?	AxB grids, where A != B	Ant should remain inside grid for all steps and not crash over many iterations when program	As expected

		accepts non square dimensions	
Does the ant ever leave the board?	Random	No matter the number of iterations, the ant should remain on the board	As expected
Does the ant turn the correct direction when stepping on white and black tiles?	20 steps, observe iterations for proper movement.	Ant turns right on white tiles, left on black tiles	As expected
Can the user place the ant outside of the board or on edges?	Grid 10x10 Ant start 10,10	Program does not allow input located on a border or outside of it	As expected
Does the program print the board after each iteration?	10 steps	10 grids print	As expected
Are there any memory leaks?	Run through valgrind	No leaks detected	As expected
Is all user input reflected properly on the grid?	Input custom attributes	Custom attributes are evident in output	As expected
menuShell & isValidInput functions			
Does all text provided to menu display correctly?	Select each option, fail each input twice	All text is accurate to situations and prompts only repeat once for failure enter correct input	As expected
Does the menu repeat continually requesting information?	Fail each input at least twice	All prompts will only repeat once for failure	As expected
Does inputting non-integer numbers	Input non-integer numbers to all inputs: 99.9, 0.0, 1.0,	Program does not crash, nor repeat	As expected

for requests for input cause the program to crash?	10.0, 55.5	prompts excessively	
Does inputting non-integer strings for requests for input cause the program to crash?	Input non-integer strings to all inputs: “Hello”, “test”	Program does not crash, nor repeat prompts excessively	As expected
Does inputting symbol strings for requests for input cause the program to crash?	Input symbols to all inputs: #, %, @, *, (, ^, %	Program does not crash, nor repeat prompts excessively	As expected
Does inputting negative integer strings for requests for input cause the program to crash?	Input negative numbers to all inputs: -5, -10, -100, -999	Program does not crash, nor repeat prompts excessively	As expected

Reflection:

Having finished this project, I fortunately didn’t have to make very many design changes. My basic design was pretty much spot on. I did end up making more menu options than I needed just to deal with different sorts of input and I realized that a few of them were not quite to the specifications of what was requested in the assignment so I made new but similar ones that fit better. I had a similar situation with my input validation functions as I made my initial program request a Y/N answer from the user but the specifications requested more of a Play/Quit option for the user to choose from.

I also didn’t have to look up pretty much anything that I hadn’t read in the textbook or used on a prior lab. In fact, the only places where I ended up looking something up was when I forgot how to create enums and when I made the previously mentioned Y/N input validation function. I also had to remind myself how random worked for the extra credit by finding it in the textbook, but then it was just a matter of plugging in values to an equation.

For testing, I fortunate in that most errors were dealt with along the way just in periodic checks to see if I’d broken anything with new additions to the program. In the end when testing against my test plan everything went smoothly. The main area where I was worried about the program breaking was when interacting with user input on custom simulations, however my input validation seems to have covered it thoroughly enough that I can’t find any issues. I also think I could have had better forethought on

testing, had I watched the lecture on testing prior to writing my program, I would have thought to implement more modular unit testing, at least in a more formal capacity. I think I did an alright job testing my new functions and additions to the program as I added them but I could have done so in a more organized and formal fashion.

The areas where I struggled were time management and properly setting up my 2D array when it was printed as a grid. The whole project took much longer than I expected because I just tend to program a little more slowly than I'd like and I frequently frequently type errors into my code due more to inattention than to a misunderstanding of theory. I have to think and spend a bit longer working with every snippet of code than would be efficent. The rest of my time was spent due to my second struggle. I tend to get confused when working with rows and columns because I will think of them as x and y coordinates. The x coordinate value is actually the xth column, and I confused the heck out of myself by mixing the two terms in writing my program. I had to sort through all of the parts of my program that depended on the grid to make sure they used the same terminology so that my grid wasn't printing properly while the ant moved at a 90 degree angle to how it should be moving. In the future I'll be making sure to work with one or the other.

Successes were abound though, or at least it seems that way before being graded. I felt pretty confident working on this project and I actually lost track of time once or twice while working on it because I was in the zone, which is awesome. I enjoy solving the problems in programming and I enjoy getting an increasingly complex program to run correctly. It also feels great when there's a small error and you are able to know exactly what it is and how to fix it just by seeing the error when the program runs. I feel like this project was good for reinforcing my organization and planning habits mostly, and overall it was a helpful experience.