Design!

Ant

- Starting row
- Starting column
- Direction he is facing -> starts with up.
- What color square is he on
- Current row number
- Current column number
- constructor that takes starting row and column as parameters and sets direction to up.
- Constructor that takes numrows and num cols constructor that chooses random starting row and columns.
- get and set functions for row, column, direction, previous square contents

Do I want to make a class for the grid… or simply make some spare functions to use in tandem with the Ant class? I think a class for the grid will make it more easy to follow and organize.

class Grid

- Constructor that takes in row and column parameters and dynamically allocates the array
- Destructor to delete the array.
- Print grid function – simply print the array
- Start function to actually start the simulation.. takes ant as parameter so it has access to ant info
- Step function/s to break the start function up a little smaller.
- Keep track of previous cell color – since it is part of the grid not the ant
- Function to change cell color

First step -> call to the menu function (or possibly class) to get all users input. If it is a function I can just pass in all the variables I need answers to by reference.

- Needs to ask how many rows and columns the user wants… can't really think of a way to validate this. I suppose they could choose as many rows and columns as they wanted.
- Next…. Does the user want a random starting location or do they want to choose one.
    - This could be a simple yes or no. cout << "Do you want a random start location?"

    - Here a validity check could be useful. Did the user choose one of the 2 answers (y or n)? If not, they need to answer again. Perhaps this is controlled by a loop…while (false) reask.

- Now, if they said no to that question then there is another question to be asked "What row and column do you want the ant to start in"?

    - This also needs a validity check. Is the row they entered within the range of rows provided? How about the column? If not, reask!(another loop)

- One more thing we must ask… "How many steps do you want the ant to take?"

    - I believe this will be controlled by a switch statement. An option menu will pop up with the steps the ant CAN take…. User must choose from the menu

    - Another validation here… did the user choose one from the menu, or did they pick a number that wasn't on the menu.

Next -> Ready to start constructing. Armed with our info we call the constructor for the Ant and the Grid.

- The ant has 2 possible constructors.. if the user picked y for wanting a random starting point, the default ant constructor will generate that start point.

Now -> We are ready to start the simulation. Call the start function from grid class. g.start(ant a) The start function will run on a while loop…until the number of steps have been reached! It will end with a counter to tell how many steps have been taken.

- This function will carry out the rest of the simulation. It will call a step function from the grid class

    - The step function is in charge of actually moving the ant. Swapping the colors around on the board and storing new info. **NEED TO CHANGE THE CURRENT LOCATION FOR ANT.**

    - If(white)

        - Call stepRight function

        - Get direction from ant. Depending on direction it will decide what square the ant is moving to.

        - Set previous color to black…. Because it was white.

            If(up)
                row, column + 1 change direction to right. Current array element = black. Store new ant locations color.. move ant
            Else if(right)
                Row+1, column set direction to down Current array element = black. Store new ant locations color.. move ant

Else if(down)

Row, column -1 set direction to left Current array element = black. Store new ant locations color.. move ant

Else

Row-1, column set direction to up. Current array element = black. Store new ant locations color.. move ant

- If(black)
  - Same steps as white… except it turns the square white instead of black. Need 2 because the direction moved is different.

If(up)

row, column – 1 change direction to left. Current array element = white. Store new ant locations color.. move ant

else if(right)

row-1, column set direction to up. Current array element = white. Store new ant locations color.. move ant

else if(down)

row, column+1 set direction to right. Current array element = white. Store new ant locations color.. move ant

else

row+1, column set direction to down. Current array element = white. Store new ant locations color.. move ant

within the start function will also be the print function… to print the grid state after every step.

Testing!

I will test each individual class by itself, just to insure its functions are working correctly, then I will start testing things together. Leaving out the menu class and just working with Ant & Grid I did these tests by manually storing values in variables:

Num Rows	Num Cols	Starting pos	Num steps	Expected	Resutls
5	5	[2][2]	10	Print board 10 times. Ant turns right on white and left on black. If ant	Worked perfectly.

				encounters edge of board it appears on opposite side.	
5	5	[2][2]	100	Print board 100 times. Ant turns right on white and left on black. If ant encounters edge of board it appears on opposite side.	Worked perfectly
5	5	[2][2]	1000	Print board 1000 times. Ant turns right on white and left on black. If ant encounters edge of board it appears on opposite side.	Worked perfectly
7	7	[2][2]	10	Same as above but the grid should be larger	Works perfectly
7	7	random	10	Same, but the ant starts at a different random place every time it is run this way.	Works perfectly

Now I added the menu, which added user interaction and tested it. Here were my tests for just the menu, and then I did all the tests above using the menu.

Prompt rows	Prompt Cols	Prompt y/n for rand	Prompt user start point	Prompt steps	expected	results
5	5	y	n/a	1	Prompt the user for num rows and cols. Ask if random start point. Should not prompt the user for starting point coordinates. Prompt for steps. Set steps to 10	Works perfectly Also checked this condition with all other step prompts. All 1-4 work.
5	5	n	[2][2]	1	Prompt the user for num rows and cols. Ask if random start point. Should prompt the user for starting point coordinates. Prompt for steps.	Works perfectly

					Set steps to 10	
5	5	a	n/a	1	Prompt for random should return “This is not a valid input” and prompt user to re-enter	Was not working at first. Realized I typed the condition check incorrectly.
5	5	y	n/a	5	Prompt for steps should return “This is not a valid input” and prompt user to enter again.	Works perfectly
5	5	y	n/a	-1	Prompt for steps should return “This is not a valid input” and prompt user to enter again.	Works perfectly

Reflection!

I took a long time to plan my project out so not too many things had to be changed in the end. Most of the issues with my design popped into my brain before I started coding the design. One of the first things I decided to change when I did my implementation was I changed where the random start point is generated. Originally I made an Ant constructor that took in the user entered starting point row and column as well as a any constructor that took in the num rows and cols and then made the random start point off that information. I realized this wouldn’t work because the overloaded functions would both have the same parameter list. I moved the random starting point function to the menu so that way the menu could just generate that when a user selects y rather than prompting for a starting point.

With the grid I decided to remove the function to change cell color. Rather than have a function to do that I just added it to the step functions because it was one line of code essentially… didn’t need to be its own function. I also changed the print grid to include an outline for the grid. When the grid was only printed a couple times it wasn’t a big deal but the bigger numbers of prints the boundaries became murky. Adding an outline to the grid made the grid move readable overall… this way you could see where it started and ended.

I did decide to make the menu its own class rather than a single function. I believe this makes it more reusable in the future because it outlines and organizes everything a lot better. The menu is in charge of prompting the user and getting their input.

A big problem I had starting out with this program is that it was definitely the hardest assignment I had been asked to accomplish to date. It felt like 161 teaches you to crawl and 162 expects your legs to be stable enough to start running. In 161 on the instructions were very clear and tight. Not much room for your own interpretation has he told us what classes we needed and what parameters they took. As well as what major functions we needed. Also I did not actually know how to dynamically

allocate a 2D array but one of the TA's posted a link to a stack overflow question regarding it. This greatly helped me figure out the structure.

Once I found a way to organize the program everything started to fall into place. I decided to use 3 classes because to me it made the code much more readable and easy to understand what is going on. Very little needed to be done in the main function. I integrated the validity checks into the menu class because that is the only place input was made… meaning it was the only place that would need to use these checks. As I mentioned in my testing phase I did run into one problem with my program and how it ran. This was simply because I forgot part of the conditional statement for the validity check, so it was just accepting all characters as being true. I also ran into a small problem with a validity function saying it was undefined…. I later found this was because I forgot the Menu:: in front of the function when I was typing the function definition. I believe the teachers hint to have us design it all out before we started programming anything was what kept a lot of the problems out of my program.