Week 1, Assignment 1(Reflection Document)

Woohyuk Yang

yangwo@oregonstate.edu

1. Requirements:

The user inputs the size(rows and columns) of grid where the ant will move on, and choose whether randomly locate the ant on the grid or to set the ant to a certain position(described with x-coordinate, and y-coordinate. Now the user inputs the number of steps the ant will move on the grid based on the rule to move. The rule which the ant can move by is if the cell(one element of grid) is white then the ant turns right 90∘(degree), and change the cell to black. If the cell is black then the ant turns left 90∘(degree), and change the cell to white. The white cell will be shown like “ ” (blank) , and the black cell will be shown like “#”, and the ant will be printed like “ *(asterisk)” .

2. Class Design

1) Initial design

Grid class

variables

- Size of rows, size of columns
- Grid color
- Char **grid

Function

- UpdateLocation(ant position, grid color)
- GridColorChange(ant position, grid color)

Other set and get functions

I thought the Grid class could be same like the Grid class I made last time for my modA. I can dynamically allocate a grid with sizes(rows , columns) the user would input and let the ant class object bind with grid class so ant class can use some variables stored in grid class object using some set and get functions of the grid class.

```
Ant class
    variables
        countMove
        antPosition for x, antPosition for y
        moveStep
        antHeading
    Function
      Ant(Grid *pGrid)
      Move(Grid *pGrid)
      StartPoint(Grid *pGrid)j
      AntHeadDecide();
```

I designed the Ant class to be really similar with Critter class I made for my modA before. But I included antHeading , and moveStep variables to store the value the user would get to input. AntHeadDecide() function had been added to let user to decide where to locate the ant object.

2) Final design (I changed my design on my implementation)

```
Grid class
    variables
        Char **grid;
        Char **colorGrid;
        Char **printGrid;
        Size of gird(rows, columns)
    Functions part
        UpdateLocation(ant position, grid color)
        ColorChange()
        FinalUpdate()
```

And I added two more functions like ColorChange, and FinalUpdate to the original class and both were designed to use two different grids.

```
Ant class
    variables
```

countMove

antPosition for x, antPosition for y

moveStep

antHeading

Function

Ant(Grid *pGrid)

Move(Grid *pGrid)

StartPoint(Grid *pGrid)j

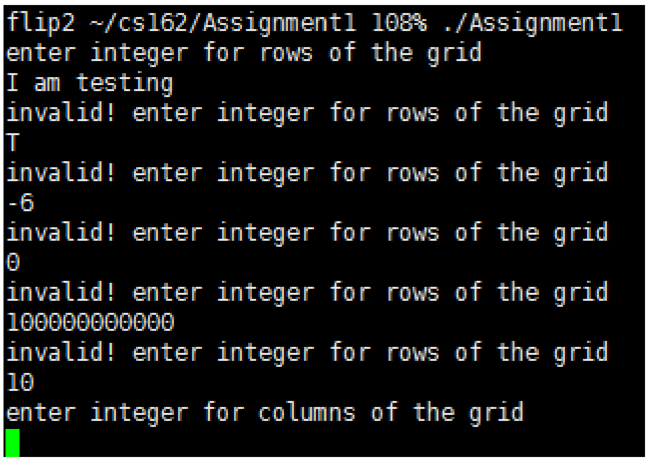
AntHeadDecide();

ValidMoveCheck();

I only added one function called "ValidMoveCheck" to ant class which validate the move of the ant. This function was made as a member function to figure out whether the ant gets to be crushed after a certain movement.

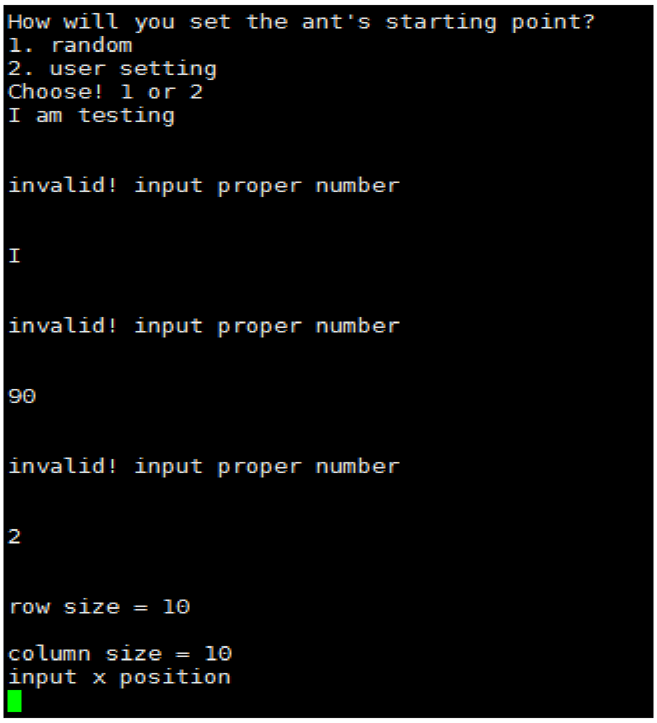
3. Test Plan

The user has to input few times to launch the ant goes around the grid. The user firstly type the size of grid, respectively. I specialized the user to input only integers which are thought to be appropriate but users can type any number or even sentences. So I wrote some code to validate the user's input and I tested with strings, characters, real number, negative integers, 0, and two big integer.(I set Max value for the grid size.) Every cases worked well and user had to repeat the same process if he/she type invalid things till he/she typed the desired integers. I tested with strings, characters, real number, negative integers, 0 with other cases when the user needs to input some integers. I think the picture below is the picture can represent my test on this subject.



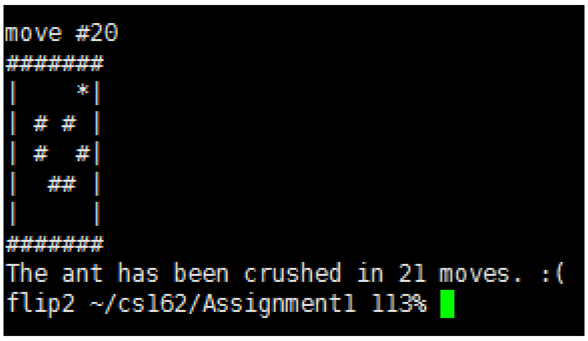
```
flip2 ~/cs162/Assignment1 108% ./Assignment1
enter integer for rows of the grid
I am testing
invalid! enter integer for rows of the grid
T
invalid! enter integer for rows of the grid
-6
invalid! enter integer for rows of the grid
0
invalid! enter integer for rows of the grid
100000000000
invalid! enter integer for rows of the grid
10
enter integer for columns of the grid
```

I made the user to choose whether to set the starting point of the ant randomly or not and when the user chose to set the position of the ant, the program worked well assigning the ant to valid position for its starting point. I also let the user to choose the ant's heading direction randomly or to input the heading user himself or herself. Like the former case, the function worked well for both cases. Like the picture I posted below, I tested with cases letting the user to decide the things about ant's starting point and ant's heading direction.



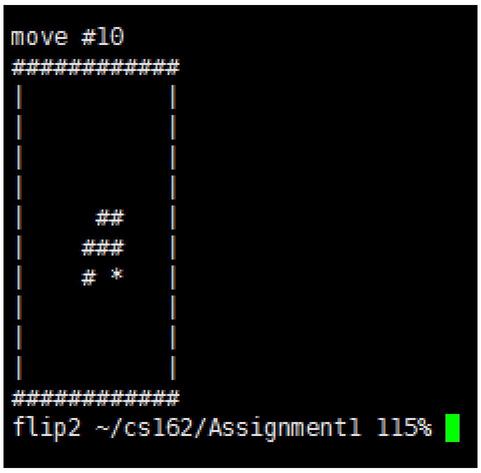
```
How will you set the ant's starting point?
1. random
2. user setting
Choose! 1 or 2
I am testing


invalid! input proper number


I


invalid! input proper number


90


invalid! input proper number


2


row size = 10

column size = 10
input x position
```

With movement of the ant, I tested two cases. The first case is when the ant gets crushed before it finishes the step input by the user. And the second case is that the ant stops moving after the number of steps, the number of steps were input by the user. My program worked well with both cases.



```
move #20
#######
|    *|
| # # |
| #  #|
|  ## |
|     |
#######
The ant has been crushed in 21 moves. :(
flip2 ~/cs162/Assignment1 113%
```

I put 50 steps but the ant has been crushed because the grid is too small(5 x 5). The program finished because the ant had been crushed.



```
move #10
############
|          |
|          |
|          |
|          |
|     ##   |
|    ###   |
|    # *   |
|          |
|          |
|          |
############
flip2 ~/cs162/Assignment1 115% 
```

I put 10 steps (10X10 grid) and the ant finished after moving 10 times. 10 is quite small number but I think that could represent the second case I mentioned above.

4. Reflection

I changed my design doing my implementation. I think that's the most dynamic change from my initial plan. At first I thought I could store color information of each cell(the elements of dynamically allocated 2D array) as an integer type variable, but implementing the real code I changed my mind. Just using integer type of color variable was way harder to code. Because I could not compare grid cells with integer variable in straightforward way.

So I decided to make another grid which can store colors of each cells making me able to access to each cell's information. ant moves so I thought it would be easier and more reasonable to implement in that way. So I made two grids; the first one is to store the position of an ant and the other one is to store whether one cell is black or white. I added one more double pointer variable and allocated another grid. I could save both information separately successfully.

As I had to display on console each move of the ant, I could not display both ant's position and cell's color easily. At first I tried to initialize the last position of an ant to "*" on color information stored grid. But The color grid lost color information which should be used to decide next ant's move. I had got stuck at this part first. I could make another grid which could be only used to display the current status of the grid. So main idea is I get three grids, and the first one is to store the current position of the ant. The second grid is to store which cell is black and white to let the ant decide which way to move. The third grid is a copy of the second, color stored grid and the ant's last position is initialized with "*"(which indicates the ant) on it.

I made a program to halt when the ant hits the wall, which means when ant gets to move out of the grid. It was not a requirement but I wrote a code like that because that does make sense the most. The ant can not move in “empty space”. That does physically make sense. I used sleep() and sys(clear) functions to make the ant looks like moving along its movement. I could adopt the way of using those functions from modA I made before and it was useful and efficient to do that.

I made validating functions as member functions so I did not have to write down same codes again and again. As I designed the program even considering the ant’s first heading direction(I think some people did not because they thought it is not that important..), I got to have several cases to check in some specific ways which are only applicable to certain cases. It was hard for me to apply member functions to every cases. Next time designing a program, contemplating the usage of member functions could be helpful to write a code in the end.