Assignment 2
Deadline: 11:59PM on Feb 23

Requirements
In this assignment, we will write a particle simulator that will show as the location of particles after a designated number of seconds. We are assuming that we are working on an XY grid that is specified by the dimensions in the input file. Your particles will only move within this boundary. The grid is boxed by borders. If your grid is 20 by 20, the vertices of the borders are at (-1,-1), (-1,20), (20,-1), and (20,20). You are given an input file which contains a set of (x, y) coordinates and (x,y) velocities. These coordinates are initial positions and velocities of the particles. The unit of velocity is 1 unit per second. For example, let's assume that input.txt contains the following inputs:

```
>> cat input.txt
20
10
0,0,1,0
0,1,1,0
1,1,1,0
E
```

All input files will hae E designator at the end marking the end of file. This input file has the dimesion of 20 by 10 where 20 is the X dimension and 10 is the Y dimension. In other word, X coordinates can go from 0 to 19 and Y coordinates can go from 0 to 9. In this case, we have 3 particles located at (0,0), (0,1), and (1,1). Their x-direction initial velocity is 1 for all particles and y-direction initial velocity is 0 for all particles. After one second, the following are their new respective coordinates: (1,0), (1,1), and (2,1).

Particles can only move in x-axis, y-axis or 45 degree angle directions. In case of 45 degree angle directions, the x and y directional velocity will have the same magnitude. A couple of example velocities are (1,1) and (1,-1).

Your program will simulate this movement and output the final position in the output file. For the above example, if we ran the simulation for 3 seconds, the final output positions will be the following: (3,0), (3,1), and (4,1).

Your output file must be a graphical position of particles with borders. Below is the sample output file for 5 by 5 grid with (3,0), (3,1), and (4,1) coordinates:

```
>> cat output.txt
*******
*     *
*     *
*     *
*  ++*
*  +  *
*******
```

Note that the output is shown for 5 by 5 as an exemplary purpose only. Your program must be able to print the grid specified by the input file. Your border must be marked with * symbols while your particles must be marked with + symbols.

Your particle can bound off the border and you should account for those. For particles moving in x and y directions, the bounding just reverses the direction while keeping the same magnitude of the velocity. For example, (1,0) with -1 x-direction velocity will be at (0,0) after one second, (1,0) after two seconds, and (2,0) after three seconds.

For those traveling at 45 degree angle, the bounding will reflect the angle. Here is a particle with (1,1) with 1 x-direction velocity and -1 y-direction velocity. The position will be (2,0) after one second, (3,1) after two seconds, and (4,2) after three seconds.

Finally, these particles have a property where if they collide, they disappear. Let's assume that we have two particles with following properties:
(1,3) at 1 x-direction velocity and 0 y-direction velocity
(2,2) at 0 x-direction velocity and 1 y-direction velocity
These two particles will both be at (2,3) after one second. Since they are at the same coordinate at the same time, they are considered collided, so these two particles will be removed from the system. This means these two particles do not exist anymore. You also need to take care of path crossing collision. We have two scenarios. Let's take a look at each one of them.
(0,0) at 1 x-direction velocity and 1 y-direction velocity
(1,0) at -1 x-direction velocity and 1 y-direction velocity
In this case, the final positions of the particle will be at (1,1) and (0,1). In this case, they are not considered collided. Our coordinate system can only deal with integer coordinates. The path cross in this case occurs when two particles cross at (0.5, 0.5), yet we do not consider such coordinates, so we ignore such crossing cases. However, consider the following.
(0,0) at 2 x-direction velocity and 2 y-direction velocity
(2,0) at -2 x-direction velocity and 2 y-direction velocity

Now with all information given, your job is to write a program that can simulate this and write the result to a output file. You are not given a skeleton file. There are sample input and reference files provided with Makefile later.

How to Compile and Run

- The Makefile for assignment1 is provided.
- The Makefile is supposed to work with a1.c, input.txt, output.txt and ref.txt files so, make sure to save your files accordingly.
- Run the following command in vs code Terminal.

  ```
  make
  ```

  It should compile the code without any errors.

`make convert_input`

It should convert the input.txt file to unix encoding.

`make run`

It should run the compiled code.

- Run the following command to delete the out file.

  `make clean`

  It should delete the specified file.

  `make convert_output`

  It should convert the output.txt file to unix encoding.
- Run the following command to test your output with provided relevant reference output.

  `make check`
- You are not supposed to make any changes in the Makefile.
- Make sure to install dos2unix utility, if not already installed using the following command:

  `sudo apt-get install dos2unix`

  For Mac

  `brew install dos2unix`

Restrictions

- No new restrictions added in this lab
- In any corner cases, write “Error” to the output file and exit gracefully

Grading

Any grading failure due to not following instructions will result in 0. You will get one chance to show your work to the instructor.

- (1 point) All files are submitted correctly using the instructions below.
- (4 point) Generate a correct solution to the problem(s) in this lab. Three test inputs will be used.
- (4 point) Handle corner cases gracefully

Submission Files

- You must submit only one file named: **assignment2.c**
- Submit it to learning hub before the deadline