Homework 8

DS-210 @ Boston University

Before you start. . .

Collaboration policy: You may verbally collaborate on required homework problems. However, you must write your solutions independently without showing them to other students. If you choose to collaborate on a problem, you are allowed to discuss it with at most 2 other students currently enrolled in the class.

The header of each assignment you submit must include the field “Collaborators:” with the names of the students with whom you have had discussions concerning your solutions. If you didn’t collaborate with anyone, write “Collaborators: none.“ A failure to list collaborators may result in a credit deduction.

You may use external resources such as software documentation, textbooks, lecture notes, and videos to supplement your general understanding of the course topics. You may use references such as books and online resources for well known facts. However, you must always cite the source.

You may not look up answers to a homework assignment in the published literature or on the web. You may not share written work with anyone else.

Submitting: Solutions should be submitted via Gradescope.

Grading: Whenever we ask for a solution, you may receive partial credit if your solution is not sufficiently efficient or close to optimal. For instance, if we ask you to solve a specific problem that has a polynomial– time algorithm that is easy to implement, but the solution you provide is exponentially slower, you are likely to receive partial credit.

Explaining: Always explain your work clearly in your writeup, even if it is correct. A good explanation can help you get points back if there are mistakes in your code. A missing or bad explanation can result in points being deducted.

Questions

To solve problems in this homework, you should use Rust. Your solution to the homework should consist of two compilable Rust projects, solving each of the questions. Remember to include the header “Collaborators“ in your source files.

1. (20 points)

   (a) Define a generic data type Point<T> representing points in the Euclidean plane with coordinates of type T.

(b) Implement two methods for values of this type: .clockwise() and .counterclockwise(). They should return a new point, corresponding to rotating the Euclidean plane around the origin (i.e., point (0, 0)) by 90 degrees, clockwise and counterclockwise respectively.

Hint 1: Look at a few examples of points undergoing such a transformation. What is the general formula?

Hint 2: To implement the above methods, you will need to require that T implements certain traits. Notice that this is T and not Point itself. The compiler is quite helpful in telling you which ones but in case reading the compiler output is difficult, you will need at least the traits Neg and Copy (and possibly a few others). The neg trait is useful if you want to be able to negate a number. The way you specify which traits the type T needs to comply with when implementing the methods of Point is like this:

```
impl<T:Copy + Neg<Output = T>> Point<T> {
        // Your implementation should be here.
}
```

(c) Show two examples of such points, one with coordinates of type f64 and the other with coordinates of type i32. Rotate one of them clockwise and the other counterclockwise by 90 degrees.

(d) Write at least one test verifying the functionality of your code. Check out the asserq_eq! macro in the rust language documentation for how to use it in your tests.

2. (20 points)

In Conway's Game of Life, a two dimensional square board is populated with some live cells. There can only be a single cell in a board square. Cells compete with each other for food so if a cell has too many neighbors it dies from malnutrition. Cells can also die of loneliness so if a cell is isolated it also dies. There is a goldilocks situation where if a neighborhood has the right amount of cells and empty space a new cell can be born. The exact circumstances of what happens are as follows:

Nb1	Nb2	Nb3	
Nb4	Target	Nb5	
Nb6	Nb7	Nb8	

If Sum(Neighbors) == 2 then Target stays as is
else If Sum(Neighbors) == 3 then Target becomes alive
else Target becomes dead

In each iteration of the game all board squares are evaluated and a new board is produced where board spaces are populated with newborn cells or emptied if cells that were there died. It is assumed that the board wraps around at the edges so cells always have 8 possible neighbors. For example:

Nb2	Nb3		Nb1
Target	Nb5		Nb4
Nb7	Nb8		Nb6

Code Conway's game of life for a board that is 16X16 and show how it evolves for 10 iterations with the initial condition where live cells are aca locations [(0,1), (1,2), (2,0), (2,1), (2,2)]. Make sure the calculation of liveness for a square is a separate function and write a test or tests that ensure its correctness.

Hint: Use Vectors of Vectors!
Hint2: For each generation use a new matrix as the output and the old matrix as the input to your calculation. Then clear the old one and swap them for the next iteration.