Task 1: Matrix Rotations [Scheme, 8 points]

Write a function `rotationInstructions` which receives a matrix `m`; represented as a list of lists; and a list of rotation instructions `instr`, represented as a list of pairs where the first element denotes the rotation direction and the second element the index of the corresponding row or column. **Note that the indexing in the instructions starts at** `0`. Possible rotation directions are `up`,`down`,`left`,`right`. Hence, `up` or `down` always refer to a **column** while `left`,`right` refer to a **row**. So, if we get rotation the instruction `(left 0)`, we rotate **row 0** to the left once. Similarly, `(down 0)` rotates **column 0** down once. See the examples below.

```
(define m '((1 2 3)(4 5 6)))
(define instr0 '((left 0)))
(define instr1 '((down 0)))
(rotationInstructions m instr0)
'((2 3 1) (4 5 6))
(rotationInstructions m instr1)
'((4 2 3) (1 5 6))
```

Implementation

Again, the **indexing used in** `instr` **starts at 0**. Also, you can expect only valid input. **Make sure your file is called** `task1.rkt` and starts with:

```
#lang racket
(provide rotationInstructions)
```

Example 1:

```
(define m '((1 2 3)(4 5 6)(7 8 9)))
(define instr '((left 1)(down 2)(right 2)))

(rotationInstructions m instr)
'((1 2 9) (5 6 3) (4 7 8))
```

Let's look at it in more detail. After consuming and executing the first instruction `(left 1)` (rotate **row** index 1 to the left once), we receive `'((1 2 3) (5 6 4) (7 8 9))`. Now we apply `(down 2)` (rotate **column** index 2 down once) to this partial result and we get : `'((1 2 9) (5 6 3) (7 8 4)`. Lastly, we apply `(right 2)` to that, yielding `'((1 2 9) (5 6 3) (4 7 8))`.

Example 2:

```
(define m1 '((1 2)(3 4)))
(define instr1 '((up 1)(right 1)))

(rotateInstructions m1 instr1)
'((1 4) (2 3))
```

Executing the first instruction `(up 1)` (rotate **column** index 1 to up once), we receive `'((1 4)(3 2))`. Now we apply `(right 1)` (rotate **row** index 1 right once) and we get : `'((1 4) (2 3))`.