Introduction:

In this assignment, you are to write a scheme procedure called *maze-solver* that finds all unique paths from the start of a maze to its end. In real mazes, you might have multiple entrances or exits but we will only consider mazes with 1 start point and 1 end point. In addition, you can assume that these will be distinct.

Problem:

The maze-solver procedure takes three arguments: a maze, the start location and the end location. It returns a list of unique paths through the maze.

*The Maze*

The maze is a list of lists where each of the inner-lists describes a row of the maze (sequentially, from top to bottom). An inner-list consists of 0s or 1s, with 0 indicating a free space where a person can walk while a 1 indicates a solid block that cannot be traversed. You may assume that each inner-list consists of the same number of elements, which is the number of columns in the maze. The number of inner-lists corresponds to the number of rows in the maze.

Here is an example maze of a 6 x 5 maze (where X is a solid block and . is a space):

```
X X . X X
. X . X X
. X . X .
. X . . .
. X X X .
. . . . .
```

This corresponds to the following list:

```
(1 1 0 1 1)
(0 1 0 1 1)
(0 1 0 1 0)
(0 1 0 0 0)
(0 1 1 1 0)
(0 0 0 0 0)
)
```
The start and end locations

The start and end locations are both lists containing exactly two elements: (x y), where (0 0) corresponds to the top-left corner of the maze. The x values increase going down and the y values increase going right.

Return Value

You start off at S. This and E are presumed to be empty spaces since you can “stand” on these. On each turn, you can move a single step North, South, East or West as long as doing so would not make you “hit” a wall. The solution paths output by your program must be unique. That is, a square should never be visited more than once. Remember that you are to output all possible unique paths! It is possible that there is no path from the start to the end. In that case, the return value is the empty list. When there are multiple paths, the return value is a list of paths. Each path in turn is a list of two element lists: ((x y) (x y) … (x y)). The first element in this list is always the start location and the last element is always the end location.

Example:

Suppose we have the following maze (S and E stand for the start and end location, respectively):

X X E X X
. X . X X
. X . X .
. X . . .
. X X X .
. . . . . S

The single path out of this maze is (5 2) → (5 3) → (5 4) → (4 4) → (3 4) → (3 3) → (3 2) → (2 2) → (1 2) → (0 2).

In scheme, we would have the expression:

(maze-solver
  `( (1 1 0 1 1)
    (0 1 0 1 1)
    (0 1 0 1 0)
    (0 1 0 0 0)
    (0 1 1 1 0)
    (0 0 0 0 0)
  )
  '(5 2 '(0 2)))
which evaluates to:

$(((5\ 2)\ (5\ 3)\ (5\ 4)\ (4\ 4)\ (3\ 4)\ (3\ 3)\ (3\ 2)\ (2\ 2)\ (1\ 2)\ (0\ 2)))$

**Submission:**

To facilitate automatic testing of your program, you should name the file containing your program “a1.scm”. Only the code needs to be submitted electronically. Use the following command:

```
submit -c csc324h -a A1 a1.scm
```

Your code should be well documented with comments. In addition, you need to submit a test report that shows the tests you performed on your program. A rationale should be provided for the test cases and these should be exhaustive. A hardcopy of the code and test report must be submitted in tutorial. For the sake of readability (and as a matter of good coding style), you should implement your solution using a number of helper functions. Keep checking the course web page to become aware of any late-breaking information about the assignment as it becomes available.

**Tentative Marking Scheme**

- 50% Automated Testing
- 20% Thoroughness of test report
- 10% Comments
- 20% Coding Style