Maze Solving:

* At a random start location and each following location after moving, the program checks the surroundings for the following:
  + The exit (S)
  + A wall (X)
  + A corridor (not accessed - /s, accessed once - :, accessed twice - |)
    - A turn or a junction
* If the exit is found, go to the exit
  + Display result – draw a line from start through corridors accessed once to exit
  + End program
* If the exit is not found:
  + Check the corridors:
    - If a corridor is not accessed, take that one, with this preference: E, N, W, S
      * Mark the corridor you left and the corridor you took with some identifying variable (: if not accessed, | if accessed once already)
    - If there is no corridor which has not been accessed, take a corridor which has been accessed once, with this preference: E, N, W, S
      * Mark the corridor you left and the corridor you took (: if not accessed, | if accessed once already)
    - If there is not corridor accessed one or fewer times, turn around
      * Mark the corridor you left and the corridor you took (: if not accessed, | if accessed once already)
  + If no corridor, continue in the last direction taken
* Some notes:
  + If the random start location was close to the exit, and no corridors existed between the start and the exit, this algorithm might not be able to draw a line to the exit but would still be able to find it. This would depend on which direction it takes from the start
  + It might be possible that the program blocks itself off from the exit, there may be need to increase the number of times a corridor can be accessed
  + It might explore the same path more than once, or appear to, but since corridors are being marked as inaccessible after two access times, it should not repeat more than twice

Maze Generation

* Requirements
  + There should be one path to the exit
  + There should be an exit
  + There should be a start
  + There should be walls around the maze
  + Maze should be varying in size
    - Maybe needs a min/max size
  + There should be no holes, or spaces not filled by either a wall or corridor
* Algorithm
  + Walls are generated around the perimeter of the size of the maze
  + A random location along the edge selected as a starting point (over writing a wall)
  + From another random location along the edge, randomly advance N, S, E, or W until the random starting point is found.
    - Each location is marked as a path in the maze, with walls on either side.
    - As the program advances through the maze, it may over wright the walls, reconnecting with part of the new path.
  + If the path is found again and it is not the first random location point, go back to the last index and move in another direction.
  + Once the iteration is complete, do the following until the maze is filled:
    - Randomly select another point not already in the maze
    - Advance until connected once with the existing maze (will not allow loop connections)
    - Repeat
* Notes
  + The maze above shouldn’t have loops, but there may be potential that the algorithm above produces some
    - Further checking could be introduced to avoid this
  + The maze will have a start and end, which would be connected from the first path generation
  + The complexity of this maze is questionable
  + The maze can vary in size