**Project 3 Report**

Notable Obstacles:

The main obstacle I encountered in project 3 was coming up with the logic to create my program. After reading through the specifications for the first time, I realized that this project was not going to be a simple project. Using what I learned from computer science class in high school, I began writing pseudocode for the different functions. After I finished writing my pseudocode, the next obstacle was converting it to actual C++ code. There were many instances that after I wrote the a piece code, I realized that there was an easier, more efficient, and more readable way to write my code. As a result, I rewrote my pseudocode and code several times. Comparing my final code to my initial code, my final code is significantly easier to follow and understand.

Program Design:

bool isRouteWellFormed(string route)

for each segment in the route

check if the first character is a directional character

if not, return false

check to see if up to next two characters are digits

if so, continue on to next segment

if the whole route is valid,

return true

int navigateSegment(int r, int c, char dir, int maxSteps)

if (r, c) is not in a valid empty grid position, or dir is an invalid direction character, or

maxSteps is negative

return -1

otherwise,

take one step in the specified direction until the number of steps taken equals

maxSteps

return the number of steps taken

int navigateRoute(int sr, int sc, int er, int ec, string route, int& nsteps)

check if route is valid and starting & ending positions are valid.

if not, return 2

for each segment in the route

set dir to directional character of that segment

store up to next 2 digits in maxSteps

set stepsTaken to actual number of steps taken and add it to totalStepsTaken

if robot runs into a wall or off the grid,

set nsteps to totalStepsTaken and return 3

change row/column of initial position according to how many steps taken in

specified direction

set nsteps to totalStepsTaken

if robot successfully navigated route to end coordinates,

return 0

otherwise if robot successfully navigated route but did not end at end coordinates,

return 1

bool isDirectionValid(char dir)

returns true if character is a directional character

bool isLocationValid(int r, int c)

returns true if if specified location is on grid

int stepsTakenInDir(int r, int c, char dir, int maxSteps)

returns the number of steps taken in a certain direction

Test Data:

Format: (row, column)

Size of grid: (3, 4)

Wall at: (1, 4), (2, 2), and (3, 2)

Picture of grid:

1234

1 ...\*

2 .\*..

3 S\*.E

* isRouteWellFormed("n23")
  + Should return true
  + Reason: to test if the function can successfully determine if 1 segment is valid
* isRouteWellFormed("n23S1ews8n")
  + Should return true
  + Reason: to test if the function can successfully determine if an entire route is valid
* isRouteWellFormed("")
  + Should return true
  + Reason: to test if the function can successfully determine if an empty route is a valid route
* isRouteWellFormed("s5w2g")
  + Should return false
  + Reason: to test if the function can determine that a route is not valid
* navigateSegment(3, 1, 'N', 2)
  + Should return 2
  + Reason: to test if the function can successfully determine number of steps taken
* navigateSegment(2, 4, 'w', 1)
  + Should return 1
  + Reason: to test if the function can successfully determine number of steps taken
* navigateSegment(1, 1, 'E', 3)
  + Should return 2
  + Reason: to test if the function would return the number of steps taken if there was a wall in its route; returned value should be less than maxSteps
* navigateSegment(1, 1, 'N', 2)
  + Should return 0
  + Reason: to test if the function can determine that it will run off the edge of the grid even though the starting position is valid
* navigateSegment(0, 0, 'N’, 2)
  + Should return -1
  + Reason: to test if the function can determine that its starting location is not valid
* navigateSegment(2, 2, 'w', 1)
  + Should return -1
  + Reason: to test if the function determines that the starting position is a wall
* navigateSegment(3, 1, 'x', 2)
  + Should return -1
  + Reason: to test if the function can determine that the specified directional character is not a valid directional character
* navigateSegment(3, 1, 's', -2)
  + Should return -1
  + Reason: to test if the function can determine that maxSteps is negative
* navigateRoute(3,1, 3,4, "N2eE01n0s2e", n)
  + Should return 0
  + Reason: to test if the function can determine that the robot will successfully navigate the route and end at the end position
* navigateRoute(3,1, 1,1, "n2", n)
  + Should return 0
  + Reason: to test if the function can determine that the robot will successfully navigate the route and end at the end position
* navigateRoute(3,1, 3,4, "N2eE01n0s1", n)
  + Should return 1
  + Reason: to test if the function can determine that the robot will successfully navigate the route but not end at the end position
* navigateRoute(3,1, 3,4, "n2E1", n)
  + Should return 1
  + Reason: to test if the function can determine that the robot will successfully navigate the route but not end at the end position
* navigateRoute(0,0, 3,4, "N2eE01n0s2e", n)
  + Should return 2
  + Reason: to test if the function can determine that the starting position is off the grid and not a valid starting position
* navigateRoute(3,1, 5,5, "n2", n)
  + Should return 2
  + Reason: to test if the function can determine that the ending position is off the grid and not a valid starting position
* navigateRoute(2,2, 1,1, "n2", n)
  + Should return 2
  + Reason: to test if the function can determine that the starting position is a wall
* navigateRoute(3,1, 1,4, "n2", n)
  + Should return 2
  + Reason: to test if the function can determine that the ending position is a wall
* navigateRoute(3,1, 5,5, "N2ex01n0s2m", n)
  + Should return 2
  + Reason: to test if the function can determine that the route is not a valid route
* navigateRoute(3,1, 3,4, "n0N1e1", n)
  + Should return 3
  + Reason: to test if the function can determine that the route will cause the robot to run into a wall
* navigateRoute(3,1, 3,4, "nN1w3", n)
  + Should return 3
  + Reason: to test if the function can determine that the route will cause the robot to run off the grid