**Pseudocode**

For isRouteWellFormed:

check for a direction, return false if not

loop from next character to the end of the string:

go back to start of the loop of the next index as long as there is a valid direction

check if the char is printable but isn’t a digit and if so, return false

otherwise //the character is sure to be a digit

repeatedly:

count the number of digits as long as each is a digit thrice and return false if over 2

return true // all tests have passed

For navigateSegment:

change direction to uppercase

check for invalidity (row/column is out of bounds, wall, maxSteps negative, printable but not a direction)

return -1 if fails

if there is a valid direction,

change proposed no. of steps to be no. rows/col depending on dir if proposed steps is greater

loop for a ‘no. of steps’ number of times until a wall is reached

count the number of actual steps

return no. of actual steps

For navigateRoute:

return 2 if route is syntactically incorrect

otherwise

repeating to the end of the string:

check for a direction without a digit to the right

shift position; return 3 if wall is reached and take a step each time

check for digits

shift position; take ‘digit’ no. of steps, return 3 is wall is reached

return 0 if the final position matches the intended endpoint

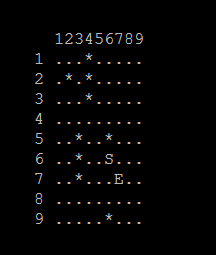
return 1 if it doesn’t

**Brief description of notable obstacles:**

The lengthiness of the spec was indication enough that this project would be a challenge. After attempting to write the first function, I first had trouble with my route.size() being unsigned. I fixed it by casting it to an int to remove the warning. I spent a while on how to check for consecutive direction chars without any digits, but eventually got it to work by checking for a digit for one index in front first. I had used isalpha to check for non-digits but I realized it didn’t account for all printable characters and changed it to isprint. For my second method, I had to use trial-and-error several times; ultimately, writing the pseudocode turned out to be the most helpful for the function’s design. After I got the method to navigate properly for one direction, reproducing it for other directions was not as difficult. The third method required the most thought. The challenge was changing the position of the robot every step while incrementing the step counter and checking for walls or out-of-bounds each step. Through a helper function which accepted the position by reference, I was able to test each step, and change the position to compare the final positions with the desired end points when the robot reached its destination.

1. Test cases for isRouteWellFormed:

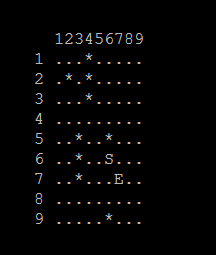
* True
  + n0, n, n1, nn, ne, e0, e2, e22, nnne0e0e0n, w, s, nswe, nSwE
* False
  + asdf, “”, x, w+n, w234, 234w, 789, 2w, 23w, nab, na2, a23

1. Test cases for navigateSegment:

* navigateSegment(6,6,n, -1)
  + returns -1
* navigateSegment(6,6,n, 1)
  + returns 0
* navigateSegment(5,5,n, 1)
  + returns -1
* navigateSegment(10,10,n,1)
  + returns -1
* navigateSegment(6,6,ee,2)
  + returns 2
* navigateSegment(6,6,s,1)
  + returns 1
* navigateSegment(6,6,w,2)
  + returns 2

1. navigateRoute (for a 9x9 grid with 6,6 starting positions and 7,7 ending positions)
   1. the int ns implicitly tests for the navigateSegment function

Test cases:

* navigateRoute(6,6,7,7, “asdf”, ns)
  + returns 2, ns = ns
* navigateRoute(10,10,7,7, “ss”, ns)
  + returns 2, ns = ns
* navigateRoute(6,6,10,10, “nn”, ns)
  + returns 2, ns = ns
* navigateRoute(6,6,7,7, “123”, ns)
  + returns 2, ns = ns
* navigateRoute(6,6,7,7, “ ”, ns)
  + returns 2, ns = ns
* navigateRoute(6,6,7,7, “”, ns)
  + returns 1, ns = 0
* navigateRoute(6,6,7,7, “s”, ns)
  + returns 1, ns = 1
* navigateRoute(6,6,7,7, “se”, ns)
  + returns 0, ns = 2
* navigateRoute(6,6,7,7, “wsssss”, ns)
  + returns 3, ns = 3
* navigateRoute(6,6,7,7, “n”, ns)
  + returns 3, ns = 0
* navigateRoute(6,6,7,7, “e2s01w”, ns)
  + returns 0, ns = 4
* navigateRoute(6,6,7,7, “n0n0”, ns)
  + returns 1, ns = 0
* navigateRoute(6,6,7,7, “e10”, ns)
  + returns 3, ns = 3
* navigateRoute(6,6,7,7, “wn6”, ns)
  + returns 3, ns = 6
* navigateRoute(6,6,7,7, “w10”, ns)
  + returns 3, ns = 2
* navigateRoute(6,6,7,7, “sss”, ns)
  + returns 3, ns = 2