The most notable obstacle I encountered when developing my program was how to separate and parse the data in the route string in the navigateRoute function. I created an initial version of the function that didn’t work correctly. After reading the processing strings announcement, I used a similar approach to make my function run correctly without excessive amounts of code.

isRouteWellFormed determines if a route string is syntactically correct. It first checks whether the first character of the string is a digit, in which case it returns false. It then loops through every character of the string. If a character is a correct direction letter, it continues to the next character. If the character is a digit, it checks whether there are three or more digits in a row, in which case it returns false. If a character is not a direction letter or a digit, it returns false. If it successfully loops through the entire string, it returns true.

isRouteWellFormed

if first character is a digit, return false

looping through route string

continue to next character if character is a direction letter

if character is a digit

return false if there are 3+ digits in a row

return false since character is not a direction letter or digit

return true since entire route string is well formed

navigateSegment determines the maximum number of steps the robot can take in a direction by first checking the validity of the input into the function and then making the robot take steps while checking if it hits a wall or goes off the grid. When that happens, the function returns the number of steps the robot took before it did.

navigateSegment

creates variable to count steps

if input is not valid or as specified

return -1

check direction input

run 1 of 4 loops based on direction

move robot a step

if out of grid or hit wall, exit

if accumulated steps is less than maxSteps, return steps

otherwise return maxSteps

navigateRoute navigates the route by first checking the validity of all the input. If any input is invalid, the function returns 2. Otherwise, the function sets nSteps to 0 and loops through the route. When it encounters a direction in the string, it calls a helper function processSegment to process the route segment that follows. If there’s an error in the segment, the function returns 3. After looping through the whole string, the function checks if the ending position of the robot matches the ending position in the input. If yes, it returns 0, otherwise it returns 1.

navigateRoute

create variables for position of robot and position in route string

if input is not valid or route is not well formed

return 2

loops through route

if character is a letter

call helper function processSegment

if there’s an error, return 3

if robot’s end position = end position in input, return 0

otherwise, return 1

The helper function processSegment processes segments in the route string that are separated by the navigateRoute function. It finds the distance the robot is supposed to go from the route segment and compares it to how far the robot can actually go according to the navigateSegment function. If they are different, that means the robot has hit a wall or gone off the grid, and thus the function returns false. Otherwise, the function moves the robot by the distance and adds the steps taken to nSteps. The position in the route string is incremented and the function returns true.

processSegment

if characters after direction are digits

save the characters in variable distance as an integer

otherwise, distance is 1

if the steps the robot can take in that direction do not equal distance

add steps that can be taken to distance and return false

otherwise, move robot by the distance and add the distance to nSteps

increment the position in the route string and return true

c.

All test cases should run successfully without stopping execution.

**setSize(5,5);**

**setWall(3,4);**

**setWall(1,2);**

**setWall(4,1);**

**setWall(5,4);**

**setWall(2,4);**

**setWall(3,3);**

**int len = -999;**

**assert(isRouteWellFormed(“”));**

- empty string is well formed

**assert(isRouteWellFormed(“n”));**

- single direction is well formed

**assert(isRouteWellFormed(“n1”));**

- single direction and single digit is well formed

**assert(isRouteWellFormed(“n12”));**

- single direction and double digits is well formed

**assert(isRouteWellFormed(“n12e3”));**

- two well formed segments are well formed

**assert(isRouteWellFormed(“N12e3”));**

- capital letters are also well formed

**assert(isRouteWellFormed(“N12Ew3”));**

- single direction with no digit is well formed

**assert(isRouteWellFormed(“N12ENw”));**

- last character is a digit is well formed

**assert(!isRouteWellFormed(“3S”));**

- first character is a digit is not well formed

**assert(!isRouteWellFormed(“x12”);**

- x is not a direction so not well formed

**assert(!isRouteWellFormed(“N150”);**

- three digits in a row is not well formed

**assert(!isRouteWellFormed(“e4+n7”);**

- symbols other than directions or digits is not well formed

**assert(navigateSegment(1, 5, 'S', 4) == 4);**

- navigates segment in south direction successfully with same number of steps as maxSteps

**assert(navigateSegment(2, 2, 's', 4) == 3);**

- navigates segment in south direction successfully with fewer number of steps as maxSteps

**assert(navigateSegment(3, 2, 'E', 4) == 0);**

- cannot take a step because of wall in east direction

**assert(navigateSegment(4, 4, 'w', 2) == 2);**

- navigates segment in west direction

**assert(navigateSegment(3, 4, 'w', 2) == -1);**

- cannot navigate segment because starting position is a wall

**assert(navigateSegment(1, 1, 'x', 1) == -1);**

- cannot navigate segment because direction is invalid

**assert(navigateRoute(1, 3, 5, 5, "s01W2s1Ese3S", len) == 0 && len == 10);**

- navigates route successfully and ends at correct ending position with correct number of steps

**len = -999;**

- resets len so we can test whether navigateRoute sets len

**assert(navigateRoute(1, 3, 5, 5, "s01W2s1Ese3", len) == 1 && len == 9);**

- navigates route successfully but ends at wrong ending position with correct number of steps

**len = -999;**

**assert(navigateRoute(1, 3, 5, 5, "3sw", len) == 2 && len == -999);**

- does not navigate route because route string is not well formed and len is not set

**len = -999;**

**assert(navigateRoute(1, 3, 5, 5, "s01W2s1E2", len) == 3 && len == 5);**

- navigates route unsuccessfully because robot hits wall, len is set to steps that could be taken

**len = -999;**

**assert(navigateRoute(1, 3, 5, 5, "s01W2s1W", len) == 3 && len == 4);**

- navigates route unsuccessfully because robots goes off grid, len is set to steps that could be taken