Tanzeela Khan

SID: 204577214

Report

1. I had to create two different functions to check if the position was in bounds. When I initially checked for an empty grid position for tempr and tempc, I checked !isWalls before postionCheck. If the position was not in bounds, the isWall function would return an error. Checking positon first solved this problem and would immediately exit out of the statement. For navigateRoute, I had to create a separate function to update the starting position and add to within the loops of navigateRoute. I was having trouble with the program differentiating between adding steps to a segment and using that same steps to add more steps. I had to create two different steps values in which steps would append to the total steps.
2. Here is my pseudocode design:

Include libraries  
Introduce the prototypes  
Main function creates grid and test values of robot

Bool isValidDir determines if character is a valid direction

Bool isInBounds determines if digit is within bounds of grid

Bool isRouteWellFormed checks if route is syntactically valid

Check if there are no characters

Check if the first character is not a direction

Run through rest of the string

Check if the characters are valid

Checks if one or two characters behind is a direction

If runs through string without false, then true

Bool postiontcheck determines if coordinate is within the assigned grid, using libraries

Int navigateSegment returns max possible steps in one direction

Check if the beginning and end coordinates are not in valid empty positions, direction is not valid, or if maxSteps is negative

If east

Run through entire segment while adding to a temporary value of columns

Checks if valid position, add to counter

Run into problem, return current counter

If west

Run through entire segment while subtracting to a temporary value of columns

Checks if valid position, add to counter

Run into problem, return current counter

If north

Run through entire segment while adding to a temporary value of rows

Checks if valid position, add to counter

Run into problem, return current counter

If south

Run through entire segment while subtracting to a temporary value of rows

Checks if valid position, add to counter

Run into problem, return current counter

If never runs into problems, return counter

void update starting position by moving for each step depending on direction

if north

move starting row back by input number of steps

if south

move starting row forward by input number of steps

if west

move starting columns back by input number of steps

if east

move starting columns forward by input number of steps

int navigateRoute sets input steps equal to max number of steps robot takes when following entire route

checks if coordinate is not an empty grid position or the route isn’t syntactically valid

runs through all characters in route

start each segment at a direction

if two consecutive directions, the first direction is incremented by 1 unless it hits a wall

if the character after direction is a number

if two consecutive numbers after direction, combine to one number

update starting positon by actual steps for each segment

if actual steps in segment does not equal attempted steps, then wall has been hit

compares steps to check if hit wall

checks if the last coordinates are equal to end coordinates

1. Test values for isRouteWellFormed:  
   (“”) to check if empty string; should return false.  
   (“0”) (“1”) (“2”) to check if the first character was not a direction, should return false.  
   (“-1”) (“s-3”) (“e2-3”) to check negative number, should return false.  
   (“A”) (“wA”) (“Z”) (“A1”) (“X1”), (“Z5”) to check if char is is not a direction character, should return false.  
   (“W”) (“w”) to check if properly takes in single direction, should return true.  
   (“W1”) (“S2”) to check if will take in single segment of string, should return true.  
   (“w9e1”) (“s09n1s1”) to check if multiple segments accepted, should return true.  
   (“we”) (“sss”) to check if consecutive directions accepted, should return true.  
   (“n02”) (“s10”) (“e12”) to check if two consecutive digits accepted, should return true.  
   (“n392”) (“s8900”) to check if more than two consecutive digits, should return false  
   (“n0”) to check if zero segment accepted, should return true.

Test values for navigateSegment:

(1,5 ,“”,0) to check empty direction, should return -1.

(1,4,“e”,2) (2,2,“e”,2) (3,4,“e”,2) (3,2,“e”,2) to check if detects walls, should return -1

(-1,-1,“e”,2) to check negative coordinates, should return -1

(0,0,“e”,2) (4,3,“e”,2) to check edge coordinates, should return -1.

(34,15,“e”,2) to check out-of-bound coordinates, should return -1

(1,1,“e”,2) (1,1,“s”,2) (1,1,“s”,0) normal case, should return counter equal to maxSteps.

(z,w,“e”,2) to check invalid alphabet coordinates, should return -1.

(1,1,“A”,2) (1,1,“Z”,2) to check invalid alphabet directions, should return -1.

(1,1,“3”,2)(1,1,“-5”,2) (1,1,“0”,2) to check invalid digit directions, should return -1.

(1,1,“s”,-1) (1,1,“s”,-10) to check negative maxSteps, should return -1.

(1,1,“s”,20) (1,1,“s”,10) to check if too many steps, should return -1.

(1,1,“s”,3) (3,4,“w”,2) (3,4,“n”,2) to check if detects walls during steps, should return counter less than maxSteps.

Test values for navigateRoute:

(0,0,3,4,“N2eE01n0s2e1",n) (-5,-2,3,4,“N2eE01n0s2e1”,n) (3,4,0,0,“N2eE01n0s2e1",n) (3,4,-5,-2,“N2eE01n0s2e1",n) to check if coordinates are out of grid, should return 2.

(1,4,3,4,“N2eE01n0s2e1",n) (2,2,3,4,“N2eE01n0s2e1",n)

(3,2,3,4,“N2eE01n0s2e1",n) (3,4,1,4,“N2eE01n0s2e1",n) (3,4,2,2,“N2eE01n0s2e1",n) (3,4,3,2,”N2eE01n0s2e1",n) to check if detects walls at coordinates, should return 2.

(3,1,3,4,“",n) to check if accepts empty string, should return 2.

(3,1,3,4,“0",n) (3,1,3,4,“1",n) (3,1,3,4,“-4",n) (3,1,3,4,“a",n) (3,1,3,4,“z",n) (3,1,3,4,“x4",n) to check for invalid routes, should return 2.

(3,1,3,4,“392”,n) (3,1,3,4,“s8900”,n) to check if accepts integers for string, should return 2.

(3,1,3,4,“n",n) (3,1,3,4,“n2",n) (3,1,3,4,“n02”,n) (3,1,3,4,“n0”,n) to check if can follow one string to incorrect endpoint, should set nsteps to number of steps taken and return 1.

(3,1,3,4,“w3",n) (3,1,3,4,“w9e1",n) (3,1,3,4,“s09n1s1”,n) (3,1,3,4,“we”,n) (3,1,3,4,“sss”,n) to check if hits walls during route, should set nsteps to number of steps taken before hitting wall and return 3.

(3,1,3,4,“N2eE01n0s2e1",n) normal case and correct route to endpoint, should set nsteps to number of steps taken and return 0.