1. One notable obstacle that I encountered was figuring out how to convert a number character in a string to an actual integer within the program. In my last function, “obeyPlan”, I had a for loop iterate through each character of the plan and then perform an action based on the character displayed. However, one issue that I encountered was that every time there was a number in the plan, I didn’t have a way to convert that character into an actual integer within the program so that I can perform calculations with that integer. After some google searches, I realized that the command “ - ‘0’ ” would allow me to utilize the character in the plan as an actual integer. In doing so, I was able to use this integer to move the position of the car accordingly. Furthermore, another obstacle that I encountered was trying to figure out how I would read two-digit numbers within the plan. In my program, I created a for loop so that it would iterate through each character and if there happened to be 3 integers in a row, then I had the program return 2. However, I wasn’t sure how to connect the two consecutive digits into one number. It turns out I can multiply the first number by 10 because it is in the tens place and then add the next digit, which would be in the ones place, in order to create one number.
2. My program consists of three different functions:

**Pseudocode for “bool hasCorrectForm(string plan)”**

if plan is blank, return true

if plan doesn’t end in a turn letter, or there are 3 numbers in a row, return false

repeatedly:

for every integer and turn letter in plan,

increment count

if count equals plan size, return true; if not, return false

**Pseudocode for “int determineSafeDistance(int r, int c …)”**

return -1 if car starts off grid, there is a wall at starting point, or maxCount is negative

if direction is east, west, north, south

repeatedly:

if car is not at edge, there is no wall in front, and maximum steps is less than count

increment count

move position of car accordingly based on direction

break

return count, or the number of steps

return -1 if dir is not a direction letter

**Pseudocode for “int obeyPlan(int sr, int sc …)”**

return 2 if plan doesn’t have correct form, there is a wall at start/end point, start/end position is not within grid, or direction character is not a direction

repeatedly:

return 3 if position is ever past grid

if character in focus is a turn letter,

set new direction based on turn letter and direction

if character in focus is a digit, change character in focus to an integer value

if following character is also digit, multiply first number by 10 and add following integer

calculate safe distance the car can travel

change position accordingly considering safe number of steps

keep track of number of steps car takes

return 0 if car ends up at end position

return 1 if car doesn’t end up at end position

1. Test Cases

**hasCorrectForm(string plan)**

* assert(hasCorrectForm("2R1r")) tests a single correct case
* assert(!hasCorrectForm("1Lx")) tests if a non-turn letter is in the plan
* assert(hasCorrectForm("")) tests if an empty string is a plan
* assert(!hasCorrectForm("2R1r3")) tests if a plan can end on a non-turn letter
* assert(!hasCorrectForm("231R1r")) tests if a plan can have 3-digit numbers
* assert(!hasCorrectForm("23l R1r")) tests spaces in a plan
* assert(hasCorrectForm("LL2R2R2L1R")) tests capital case turn letters
* assert(hasCorrectForm("ll2r2r2l1r")) tests lower case turn letters
* assert(!hasCorrectForm(" ")) tests if a space character is a plan

**determineSafeDistance(int sr, int sc …)**

* assert(determineSafeDistance(3, 1, 'N', 2) == 2) tests a single correct case
* assert(determineSafeDistance(4, 1, 'N', 2) == -1) tests when the start point is outside the grid
* assert(determineSafeDistance(0, 1, 'N', 2) == -1) tests when the start point is outside the grid
* assert(determineSafeDistance(2, 5, 'N', 2) == -1) tests when the start point is outside the grid
* assert(determineSafeDistance(1, 4, 'N', 2) == -1) tests when the start point is on a wall
* assert(determineSafeDistance(3, 1, 'e', 2) == 0) tests when there is a wall in front of the car
* assert(determineSafeDistance(3, 1, 'n', 5) == 2) tests when the plan tells it to go 5 steps, but it can only go 2
* assert(determineSafeDistance(1, 3, 's', 2) == 2) tests when the plan can go the max distance
* assert(determineSafeDistance(2, 4, 'w', 3) == 1) tests when the car hits the wall and max distance is greater than safe distance
* assert(determineSafeDistance(2, 3, 'e', 2) == 1) tests when the car hits the edge of the grid
* assert(determineSafeDistance(3, 1, 'N', -1) == -1) tests when the maxSteps is negative

**obeyPlan(int sr, int sc …)**

* assert(obeyPlan(3, 1, 3, 4, 'S', "LL2R2r2L1R", len) == 0 && len == 7) tests a single correct case
* assert(obeyPlan(3, 1, 3, 4, 'N', "1Lx", len) == 2 && len == -999) tests when the plan is not in a correct form
* assert(obeyPlan(2, 4, 1, 1, 'w', "3R1L", len) == 3 && len == 1) tests when the car hits a wall
* assert(obeyPlan(3, 1, 3, 4, 'n', "2r2l", len) == 1 && len == 4) tests when the car does not end up in the correct ending position
* assert(obeyPlan(2, 2, 3, 4, 'n', "2r2l", len) == 2 && len == -999) tests when the start position is on a wall
* assert(obeyPlan(1, 5, 3, 4, 'n', "2r2l", len) == 2 && len == -999) tests when the start position is not within the grid
* assert(obeyPlan(6, 2, 3, 4, 'n', "2r2l", len) == 2 && len == -999) tests when the start position is not within the grid
* assert(obeyPlan(2, 3, 1, 4, 'p', "2r2l", len) == 2 && len == -999) tests when dir is not a direction letter
* assert(obeyPlan(3, 1, 3, 4, 'n', "2R2r1r2l", len) == 3 && len == 5) tests when the car hits a wall with a more complex plan
* assert(obeyPlan(3, 1, 3, 4, 'n', "2R2r3l", len) == 3 && len == 6) tests when the car goes off the grid
* assert(obeyPlan(3, 1, 3, 4, 'e', "2R2r3l", len) == 3 && len == 0) tests when the car hits a wall and travels a total amount of 0 steps
* assert(obeyPlan(3, 1, 3, 4, 'e', "", len) == 1 && len == 0) tests when the plan is an empty plan
* assert(obeyPlan(5, 6, 12, 20, 'e', "10L4r4r11r", len) == 0 && len == 29) tests when numbers are greater than 10 and the plan consists of two-digit numbers
* assert(obeyPlan(5, 2, 12, 20, 'e', "19r", len) == 3 && len == 18) tests when car goes off grid with 2-digit numbers in the plan