a.) Three were four main obstacles I had to overcome. The first was determining how to structure if-statements to capture only valid course strings. The second was appropriately developing the function *driveSegment* to account for all possibilities. The third was structuring the the if-statements in the function *driveCourse* to account for all the possibilities, e.g. the direction character is the last character, is followed by another direction character, etc. The four was realizing that I needed to exploit ASCII code to convert character digits like ‘2’ into their numeric value, e.g. *2*.

b.) The pseudocode for my project is presented below. There were three smaller, helper functions in addition to these three primary functions, but their structure is so simplistic that they need not be thoroughly documented.

**bool isCourseWellFormed(string course):**

if course is empty string

return true

if first character in string is not alphabetic

return false

If the string one has a single character and it’s not alphabetic

return false

for each character in the string

if the ith character is not alphabetic or a digit

return false

If the ith character is a alphabetic

If the ith character is not an N, E, S, or W

return false

if ith character is a digit

if the next character (ith + 1) is a digit

return false

If the next character (ith + 2) is a digit

return false

return true

**int driveSegment(int r, int c, char dir, int maxSteps):**

if position is not valid

return -1

If input for direction is not valid

return -1

If input for steps is negative

return -1

if direction is North (‘n’ or ‘N’), repeatedly:

decrease row value by one

add one to step counter

If step counter equals maxsteps,

return max steps

If a nonvalid position is encountered,

return step counter

if direction is South, repeatedly:

increase row value by one

add one to step counter

If step counter equals maxsteps,

return max steps

If a nonvalid position is encountered,

return step counter

if direction is East, repeatedly:

increase column value by one

add one to step counter

If step counter equals maxsteps,

return max steps

If a nonvalid position is encountered, r

eturn step counter

if direction is West, repeatedly:

decrease column value by one

add one to step counter

If step counter equals maxsteps,

return max steps

If a nonvalid position is encountered,

return step counter

**int driveCourse(int sr, int sc, int er, int ec, string course, int& nsteps):**

If the start position or the end position is not valid, return 2

if the course is not well-formed, return 2

If the course string is empty

if the start position and the end position are the same return 0

Else return 1

set *nsteps* to zero

For each char in the course string

If the ith char is alphabetic and is the only char or last char

call driveSegment

If driveSegment < 1, return three

call update position

If there is a next char (i +1)

If the ith + 1 char is alphabetic

add driveSegment to nSteps

If driveSegment < 1, return three

call update position

If the ith + 1 char is a digit and the last digit

addl driveSegment (called with [i + 1) to nSteps

If driveSegment < numeric value of [i + 1] digit, return three

call update position

If there are at least two more char (i + 1 and i + 2)

If the ith + 2 char is alphabetic

addl driveSegment (called with [i + 1]) to nSteps

If driveSegment < numeric value of [i + 1], return three

call update position

If the ith + 2 char is a digit

add driveSegment (called with 10\*[i+1] + [i + 2]) to nSteps

If driveSegment < numeric value of 10\*[i+1] + [i + 2] return three

call update position

If the start position equals end position, return 0

else return 1

b.)

*assert(isCourseWellFormed("N") == true);*

*assert(isCourseWellFormed("N2eE01n0e2e1") == true);*

*assert(isCourseWellFormed("NW42") == true);*

*assert(isCourseWellFormed("w2+n3") == false); //non-alphabetic char*

*assert(isCourseWellFormed("N144") == false); //three consecutive digits*

*assert(isCourseWellFormed("e1x") == false); // invalid alphabetic char*

*assert(isCourseWellFormed("3sn") == false); //starts with digit*

*assert(isCourseWellFormed("") == true);*

*assert(isCourseWellFormed(" ") == false); //invalid char (blankspace)*

*assert(isCourseWellFormed("EEEE") == true);*

*assert(isCourseWellFormed("43424e") == false); //multiple failed conditions*

*assert(isCourseWellFormed("434423x") == false); //multiple failed conditions*

*assert(isCourseWellFormed("EEE43WWW2NNNN3") == true);*

*\*\*\** Test for driveSegment are grid-dependent. The tests below check that the function returns expected values for positions given in lower and uppercase, for invalid inputs, and for cases where an invalid position is reached. \*\*\*

*assert(driveSegment(1, 1, 'e', 7) == 2); //iterates until invalid position is reached*

*assert(driveSegment(1, 1, 'S', 3) == 3); //*

*assert(driveSegment(1, 1, 'N', 7) == 0);*

*assert(driveSegment(1, 1, 'W', 7) == 0);*

*assert(driveSegment(3, 1, 'W', 7) == 0);*

*assert(driveSegment(4, 10, 'W', 7) == 6);*

*assert(driveSegment(3, 8, 'E', 7) == 0);*

*assert(driveSegment(5, 7, 'N', 7) == 4);*

*assert(driveSegment(1, 10, 'W', 7) == 5);*

*assert(driveSegment(4, 1, 'e', 7) == 1);*

*assert(driveSegment(2, 5, 's', 7) == 0);*

*assert(driveSegment(3, 1, 'W', 7) == 0);*

*assert(driveSegment(3, 1, 'd', 7) == -1); //invalid inputs*

*assert(driveSegment(3, 1, 'e, -7) == -1); //invalid inputs*

\*\*\* Test for driveCouse are grid-dependent. \*\*\*

*assert(driveCourse(1, 1, 3, 1, "ss", nSteps) == 0 && nSteps == 2 ); consecutive alphabetic characters, returns correct value*

*assert(driveCourse(1, 1, 3, 1, "E", nSteps) == 1 && nSteps == 1); handles single alphabetic character, return correct value*

*assert(driveCourse(5, 3, 5, 1, "w2", nSteps) == 0); handles digit following alphabetic character*

*assert(driveCourse(5, 3, 5, 2, "w2", nSteps) == 1);*

*assert(driveCourse(2, 1, 24, 11, "nE2see6s31ne", nSteps) == 3); much more complex course, handles two consecutive digits, returns 3 when appropriate (for grid size 25x25)*

*assert(driveCourse(5, 3, 5, 1, "wefafd", nSteps) == 2) handles invalid course string*

*assert(driveCourse(54, 3, 5, 1, "we3n", nSteps) == 2) handles invalid start/stop position*

*assert(driveCourse(5, 3, 5, 1, "", nSteps) == 1) handles empty string*

*assert(driveCourse(5, 3, 5, 3, "", nSteps) == 0) handles empty string when start position equals start position*