**Name:** Eshan Uniyal **UID:** 205-172-354

# Obstacles

1. **Ordering of “or” expressions in if statements**: At several points in the program, we need to write if statements to test whether the input coordinates are valid empty strings. For example, in the driveCourse function, such a statement might read:

if (isWall(sr, sc) || isWall(er, ec) ||

sr > getRows() || sr <= 0 ||

er > getRows() || er <= 0 ||

sc > getCols() || sc <= 0 ||

ec > getCols() || ec <= 0)

However, this is an initial implementation of the test is incorrect; the isWall() expressions should come last in the if statement, else we’d be testing for whether there’s a wall at a position we haven’t yet checked is within the grid. The correct implementation is as follows:

if (sr > getRows() || sr <= 0 ||

er > getRows() || er <= 0 ||

sc > getCols() || sc <= 0 ||

ec > getCols() || ec <= 0 ||

isWall(sr, sc) || isWall(er, ec))

1. **Converting digits in string to integers:** Without a straightforward function to convert strings to integers (as one may use in Python), it was tricky and quite interesting to initialise an integer with the character ‘0’ and use the encoding thus stored to find integer equivalents to digit characters.
2. **Writing useful test cases**: Given the number of functions and possible inputs and grids each may have to deal with, writing meaningful test cases to cover most such possibilities was challenging.

# Program Design

The program consists of the **three main functions** (isCourseWellFormed, driveSegment, and driveCourse) and four supplementary funcions (driveSegmentEast, driveSegmentWest, driveSegmentNorth, and driveSegmentSouth) that driveSegment calls to calculate the number of steps we can move in a particular direction.

Following is pseudocode for the isCourseWellFormed function:

test whether all the characters are digits or valid direction characters; if any characters are invalid, return false

test whether the first character is a letter; if not, return false

repeatedly for each character in the string:

initialise direction character to first character in string, and initialise string segment

if the character is a direction character,

initialise segment to the character

repeatedly for each character after the current character:

if character is a digit,

add it to segment

else,

break the inner loop, move to next segment (if any)

if the segment is longer than three characters, return false

all characters are valid and all segments are of valid length, return true

Following is pseudocode for the driveSegment function:

if start coordinates outside grid or a wall at start coordinates or negative

maxSteps

return -1

check the direction character, then initialise possible\_steps to store the number of steps we can move

assign possible\_steps to the relevant supplementary driveSegment function (e.g. driveSegmentWest), which returns the number of steps we can move in the direction

if invalid direction character, return -1

return whichever is smaller between possible\_steps and maxSteps

Following is pseudocode for the four supplementary functions used by driveSegment:

repeatedly for position inside grid:

if wall at position

return distance between start and current position

increment position

no wall encountered, return distance between start position and farthest position (in the same direction) in the grid

Following is pseudocode for the driveCourse function:

if (sr, sc) or (er, ec) are not valid empty grid positions, or course is not syntactically valid (checked via isCourseWellFormed), return 2

initialise constant integer zero with the character ‘0’ to store its encoding

initialise variables cr and cc to store current row and current column with sr and sc respectively

initialise n\_steps to store the total number of steps we can take following the course

initialise integer status with default value 1 to indicate the course was successfully completed

determine segments using a similar algorithm as in isCourseWellFormed

for each segment,

initialise steps with a default value of 0 to track how many steps are possible in the segment

if segment length is one,

if driveSegment returns one,

set steps to 1

else,

set status to 0 (course failed)

else,

initialise integer maxSteps with default value 0

if segment length is two,

set maxSteps to the difference between the encoding of segment[1] and constant integer zero

if segment length is three,

set maxSteps to ten times the difference between encoding of segment[1] and constant integer zero, plus the difference between encoding of segment[2] and constant integer zero

set steps to whatever value driveSegment returns given cr, cc, dir, and maxSteps

if steps is less than maxSteps,

set status to 0 (course failed)

add steps to n\_steps

change (cr, cc) depedending on direction character and steps

if status equals zero,

break out of loop

change parameter nsteps’ value to n\_steps

if status equals zero,

return 3 (course could not be completed)

else,

if (cr, cc) equals (er, ec)

return 0 (course was completed and finished at end coordinates)

else

return 1 (course was completed but did not finish at end coordinates)

# Test Data

**Testing isCourseWellFormed()**

assert(isCourseWellFormed("")); // expected true: empty string is valid

assert(isCourseWellFormed("N2eE01n0s2e1")); // expected true

assert(!isCourseWellFormed("3sn")); // expected false: starts with a digit

assert(!isCourseWellFormed("n2e001e1")); // expected false: 2+ consecutive digits

assert(!isCourseWellFormed("w2+n3")); // expected false: invalid character '+'

assert(!isCourseWellFormed("e1x")); // expected false: invalid character 'x'

assert(!isCourseWellFormed("3")); // expected false: only a digit

assert(!isCourseWellFormed(" ")); // expected false: only a space character

**Testing driveSegment()**

assert(driveSegment(-1, 1, 'n', 4) == -1); // start coordinate not in grid

assert(driveSegment(1, -1, 'n', 4) == -1); // start coordinate not in grid

assert(driveSegment(0, 1, 'n', 4) == -1); // start coordinate not in grid

assert(driveSegment(6, 1, 'n', 4) == -1); // start coordinate not in grid

assert(driveSegment(1, 6, 'n', 4) == -1); // start coordinate not in grid

assert(driveSegment(2, 2, 'n', 1) == -1); // start position occupied by wall

assert(driveSegment(1, 1, 'x', 4) == -1); // invalid direction character

assert(driveSegment(1, 1, 'n', -2) == -1); // negative maxSteps

assert(driveSegment(3, 4, 'n', 1) == 1); // can travel one step north

assert(driveSegment(3, 4, 'n', 2) == 1); // can only travel one step north

assert(driveSegment(3, 4, 's', 0) == 0); // maximum steps are zero

assert(driveSegment(3, 4, 's', 1) == 1); // can travel one step

assert(driveSegment(3, 4, 'S', 1) == 1); // capital direction character works

assert(driveSegment(3, 4, 's', 2) == 2); // can travel two steps

assert(driveSegment(3, 4, 's', 3) == 2); // can only travel two steps

assert(driveSegment(3, 4, 'e', 1) == 1); // can travel one step

assert(driveSegment(3, 4, 'e', 2) == 1); // can only travel one step

assert(driveSegment(3, 4, 'w', 0) == 0); // can travel 0 steps

assert(driveSegment(3, 4, 'w', 1) == 0); // cannot travel any steps

**Testing driveCourse()**

int nsteps;

assert(driveCourse(-1, 1, 3, 4, "e02sse1", nsteps) == 2); // invalid start coordinates

assert(driveCourse(1, 0, 6, 4, "e02sse1", nsteps) == 2); // invalid start coordinates

assert(driveCourse(1, 1, 6, 4, "e02sse1", nsteps) == 2); // invalid end coordinates

assert(driveCourse(1, 1, 0, 4, "e02sse1", nsteps) == 2); // invalid end coordinates

assert(driveCourse(2, 2, 3, 4, "e02sse1", nsteps) == 2);

// start coordinates blocked by wall

assert(driveCourse(1, 1, 3, 3, "e02sse1", nsteps) == 2);

// end coordinates blocked by wall

assert(driveCourse(1, 1, 3, 4, "n2e001e1", nsteps) == 2); // invalid course string

assert(driveCourse(1, 1, 3, 4, "e02sse1", nsteps) == 3);

// course could not be completed

assert(nsteps == 3); // travelled three steps in previous test

assert(driveCourse(1, 1, 3, 4, "eesse", nsteps) == 3);

// course could not be completed

assert(nsteps == 3); // travelled three steps in previous test

assert(driveCourse(1, 1, 3, 4, "w3s1e", nsteps) == 3);

// course could not be completed

assert(nsteps == 0); // travelled no steps in previous test

assert(driveCourse(1, 1, 3, 4, "S2ese02N1", nsteps) == 0);

// course completed at end position

assert(nsteps == 7); // travelled seven steps in previous test

assert(driveCourse(1, 1, 3, 4, "e02se", nsteps) == 1);

// course completed, but not at end position

assert(nsteps == 4); // travelled four steps in previous test