1. The notable obstacle I met is that after finishing all codes and running it with the 6 example tests, I found that I have passed all the tests except for test 6. I then started to look at my codes to find the error in my code but after an hour I still have no error in my logic. Then I started to use “cerr” to print out the value of my variables in my loops so that I could check whether there is something wrong within my loops, and I indeed find the unexpected value in a specific loop. I then check it together with my previous modification of the variable and ultimately found that I forgot to use the “tolower()” function to lower the alpha in the “dir” variable, which causes an error that ruins the tests. After the correction, my program can perform accurately.
2. Pseudocode
3. function bool hasCorrectForm(string plan)

find plan size;

repeatedly:

find if the next letter is R,r,L,or l

if yes,

continue checking

else

the plan is not correct, break

if the plan is more than 3 letters

repeatedly check if it has 3 continuous letter that are all digits

if yes,

the plan is not correct, break

else

continue checking

if the plan’s last letter is digit

then the plan is not correct, break

return whether the plan is correct or not

1. function int determineSafeDistance(int r, int c, char dir, int maxSteps)

if the point(r,c) or dir or maxSteps is not valid,

return -1

else

check the direction is north, south, west, or east

repeatedly check if (r,c) is valid and the total step is smaller than the maxSteps

if the next point it will go is valid

increase one step and change it present position

else

stop moving and record the present steps

return total steps it has been going

1. int obeyPlan(int sr, int sc, int er, int ec, char dir, string plan, int &nsteps)

check if the initial position, end position, dir, and plan are not valid

if yes, return 2 and end checking

else

lower all the letters in original plan string

repeatedly check the next char in plan is alpha or digit

if the next char is alpha

change direction according to the alpha

else

check if there is any continuous two numbers

if yes

the moving steps would be calculated by 10\* tens digit + single digit

check if the moving with the maxSteps would step into wall or out of maze

if yes

return 3 and end the loop

else

change current position according to the maxSteps

record total steps

if the ultimate position is the same with the expected end position

return 0 and change the nsteps to the total steps

else

return 1 and change the nsteps to the total steps

1. testcases:

setSize(12, 15);

setWall(1, 3);

setWall(1, 4);

setWall(1, 7);

setWall(2, 1);

setWall(2, 4);

setWall(2, 11);

setWall(3, 2);

setWall(3, 5);

setWall(3, 8);

setWall(4, 6);

setWall(5, 3);

setWall(5, 6);

setWall(7, 3);

setWall(8, 8);

setWall(9, 15);

setWall(12, 11);

setWall(10, 5);

assert(hasCorrectForm("2R1r")); //the plan is normal plan

assert(hasCorrectForm("1r2L3llr23R")); // the plan is a complicated normal plan

assert(hasCorrectForm("")); // the plan is empty

assert(hasCorrectForm("l")); // the plan has zero digit

assert(hasCorrectForm("R")); //the plan has upper alpha

assert(hasCorrectForm("2l")); // the plan has one digit

assert(!hasCorrectForm("r2")); // this is not the plan because it ends without direction

assert(hasCorrectForm("llrrlrlr")); // this is a normal plan without numbers

assert(hasCorrectForm("12l")); // the plan has two digits

assert(!hasCorrectForm("l234r")); // the plan has more than two digits

assert(!hasCorrectForm("1r2")); // the plan's last letter is a digit

assert(!hasCorrectForm("2")); //the plan is only one digit without direction

assert(!hasCorrectForm("123")); // the plan has more than two digits

assert(!hasCorrectForm(" ")); // not a plan; composed of space

assert(!hasCorrectForm("1l2R 3r")); // not a plan; seperated by space

assert(!hasCorrectForm(" 3rl1 ")); // not a plan; surrounded by space

assert(!hasCorrectForm(",1l2r!")); // not a plan; surrounded b comma and other things

assert(!hasCorrectForm("2rd1l")); // not a plan because of d appeared in the plan

assert(determineSafeDistance(1,3,'n', 2) == -1); //the point(1,3) is a wall

assert(determineSafeDistance(99, 2, 'w', 2) == -1); // the point (99,2) is not valid in the maze

assert(determineSafeDistance(2,199, 'w', 2) == -1); // the point (2, 199) is not valie in the maze

assert(determineSafeDistance(4, 2, 'd', 4) == -1); // the dir'd' is not a correct direction

assert(determineSafeDistance(4, 7, 'e', -2) == -1); // the maxStep is negative

assert(determineSafeDistance(6, 12, 'w', 0) == 0); //maxStep is 0

assert(determineSafeDistance(8, 2, 'n', 10) == 4); //during the moving process, it will meet a wall

assert(determineSafeDistance(10, 6, 'W', 3) == 0); // it cannot move to the left because the wall is on the left

assert(determineSafeDistance(3, 1, 'w', 2) == 0); // it cannot move to the left because that is out of the maze

assert(determineSafeDistance(10, 15, 'e', 2) == 0); //it cannot move to the right because that is out of the maze

assert(determineSafeDistance(4, 1, 'e', 4) == 4); //go east and the plan is valid

assert(determineSafeDistance(4, 1, 's', 5) == 5); //go south and the plan is valid

assert(determineSafeDistance(8, 5, 'W', 3)== 3); // go west and the plan is valid

assert(determineSafeDistance(12, 6, 'n', 5) == 5); // go north and the plan is valid

assert(determineSafeDistance(12, 6, 'N', 4) == 4); // go north with capital dir char

int len = -999;

assert(obeyPlan(8,2,8,4,'s',"ll2R2r2l", len) == 0 && len == 6); // it successfully follow all the steps to arrive at the expected point

len = -999;

assert(obeyPlan(12,1,1,10,'e', "9l11r",len) == 0 && len == 20); // the plan has two continuous number

len = -999;

assert(obeyPlan(12, 1, 1, 10, 'e', "fdfs", len) == 2 && len == -999); //the plan is not valid

len = -999;

assert(obeyPlan(8,2,8,4,'s', "ll2R2r3r", len) == 1 && len == 7); // it successfully follow the plan but is not end at the expected end point

len = -999;

assert(obeyPlan(99,2,8,2,'s', "ll2rlr", len) == 2 && len == -999); //the initial point is not available

len = -999;

assert(obeyPlan(2,3,99,3,'w', "ll2rl2r", len) == 2 && len == -999); //the end point is not available

len = -999;

assert(obeyPlan(2,4,8,4,'s', "llr2rl", len) == 2 && len == -999); // the initial point is a wall

len = -999;

assert(obeyPlan(2,2,3,2,'s', "lr2rlr1", len) == 2 && len == -999); // the end point is a wall

len = -999;

assert(obeyPlan(2,2,4,2,'d', "lllr2lll", len) == 2 && len == -999); // the direction char is not correct

len = -999;

assert(obeyPlan(2,2,4,2,'w', "234l", len) == 2 && len == -999); //the plan is not a correct plan

len = -999;

assert(obeyPlan(2,2,11,2,'w',"1lrl8r", len) == 3 && len == 0); // the plan will walk into a wall

len = -999;

assert(obeyPlan(12,1,1,5,'w', "rr9l11RRR3l", len) == 3 && len == 22); // the plan will walk into a wall

cerr << "All tests succeeded" << endl;