```c
#include "functions.h"

int main(int argc, char **argv){
        int i, j;
        double x, y;
        char a;

        //scan the size of the matrix
        scanf("%d %d", &m, &n);

        //scan the matrix itself
        getchar();
        for(i = 0; i < n; i++){

                for(j = 0; j < m; j++)
                        w[i][j] = getchar();

                //throw away break line
                a = getchar();
        }

        scanf("%lf", &x);
        scanf("%lf", &y);

        scanf("%lf", &goal_x);
        scanf("%lf", &goal_y);

        printRRT(x, y);

        return 0;
}
```



```c
#include "functions.h"

//insert on front of the list of actions
void insertListActions(double x, double y){
        list new;

        new = malloc(sizeof(list_node));
        if(new == NULL)
                printf("error alocating memory for list node\n");
        new -> next = actions;
        new -> x = x;
        new -> y = y;

        actions = new;
}

//insert on front of the list of nodes visited
void insertListVisited(tree t, double x, double y){
        list new;

        new = malloc(sizeof(list_node));
        if(new == NULL)
                printf("error alocating memory for list node\n");
        new -> next = visited;
        new -> x = x;
        new -> y = y;
        new -> t = t;

        visited = new;
}

//prints a list
void printList(){
        list aux = actions;

        while(aux != NULL){
                printf("%f %f\n", aux -> x, aux -> y);
                aux = aux -> next;
        }
}

//insert a node in the tree
tree insertTree(tree p, double x, double y){
        tree new;
        int i;

        new = malloc(sizeof(tree_node));
        if(new == NULL)
                printf("error allocating memory for tree\n");
        new -> parent = p;
        new -> x = x;
        new -> y = y;

        return new;
}

//gets the sign of a number
int sgn(double x){
        if(x > 0)
                return 1;
        else if(x < 0)
                return -1;
        else
                return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>

//structure of each node in a tree
typedef struct tree_aux{
        double x;
        double y;
        struct tree_aux* parent;
} tree_node, *tree;

//structure of the list of actions taken
typedef struct list_aux{
        double x;
        double y;
        tree t;
        struct list_aux *next;
} list_node, *list;

//global variables
int n, m;
char w[1000][1000];
double goal_x, goal_y;
list visited;
list actions;
tree t;

//functions declaration
int sgn(double x);
void insertListActions(double x, double y);
void insertListVisited(tree t, double x, double y);
void printList();

tree insertTree(tree p, double x, double y);

void printRRT(double x, double y);
tree RRT(double x, double y);
tree nearest(double x, double y);
int step(double x, double y, double x2, double y2);
void randomCoord(double *x, double *y);
```



```
#include "functions.h"

//step towards it. We do the calculation using the fact that the triangles are s
imilar
int step(double x, double y, double x2, double y2){
        double a, b, add_a, add_b, hip;
        int sign_x, sign_y;
        int index_x, index_y;

        sign_x = sgn(x2 - x);
        sign_y = sgn(y2 - y);

        a = x2 - x;
        b = y2 - y;

        hip = sqrt(a*a + b*b);

        add_a = (0.25 * a) / hip;
        add_b = (0.25 * b) / hip;

        a = x;
        b = y;

        while(w[n - 1 -(int)floorf(b)][(int)floorf(a)] != '#'){

                //if the point is between cells, check all the adjacents cells
                if(fmod(a, 1) == 0 && fmod(b, 1) == 0){
                        if(w[n - 1 -(int)floorf(b)][(int)floorf(a) - 1] == '#')
                                break;
                        if(w[n - 2 -(int)floorf(b)][(int)floorf(a) - 1] == '#')
                                break;
                        if(w[n - 1 -(int)floorf(b)][(int)floorf(a)] == '#')
                                break;
                        if(w[n - 2 -(int)floorf(b)][(int)floorf(a)] == '#')
                                break;
                }

                if(fmod(a, 1) == 0){
                        if(w[n - 1 -(int)floorf(b)][(int)floorf(a)] == '#')
                                break;
                        if(w[n - 1 -(int)floorf(b)][(int)floorf(a) - 1] == '#')
                                break;
                }

                if(fmod(b, 1) == 0){
                        if(w[n - 1 -(int)floorf(b)][(int)floorf(a)] == '#')
                                break;
                        if(w[n - 2 -(int)floorf(b)][(int)floorf(a)] == '#')
                                break;
                }

                //increment the point
                a = a + add_a;
                b = b + add_b;

                //check if we already passed the point
                if(sgn(x2 - a) != sign_x || sgn(y2 - b) != sign_y)
                        return 1;
        }

        return 0;
}

//find the nearest point
tree nearest(double x, double y){
        double d = 999999;
        double a, b, hip;
        list aux;
        tree t;
```

```
        for(aux = visited; aux != NULL; aux = aux -> next){
                a = fabs(x - aux -> x);
                b = fabs(y - aux -> y);
                hip = sqrt(a*a + b*b);

                if(hip < d){
                        d = hip;
                        t = aux -> t;
                }
        }

        return t;
}

//generate random numbers for the coordinates with a goal bias
void randomCoord(double *x, double *y){
        double i;

        i = ((double)rand()/(double)(RAND_MAX));
        if(i > 0.05)
                *x = ((double)rand()/(double)(RAND_MAX)) * m;
        else
                *x = goal_x;

        i = ((double)rand()/(double)(RAND_MAX));
        if(i > 0.05)
                *y = ((double)rand()/(double)(RAND_MAX)) * n;
        else
                *y = goal_y;
}

//implements RRT algorithm
tree RRT(double x, double y){
        double random_x, random_y;

        srand(time(NULL));

        //initialize the tree
        t = insertTree(NULL, x, y);
        insertListVisited(t, x, y);

        while(1){
                //generate random numbers for x and y coordinates
                randomCoord(&random_x, &random_y);

                //find the nearest node to the random point
                t = nearest(random_x, random_y);

                //step towards it
                if(step(t -> x, t -> y, random_x, random_y) == 1){
                        t = insertTree(t, random_x, random_y);
                        insertListVisited(t, random_x, random_y);

                        if(sqrt((random_x - goal_x)*(random_x - goal_x) + (rando
m_y - goal_y)*(random_y - goal_y)) <= 1)
                                return t;
                }
        }
}

//prints result of executing RRT
void printRRT(double x, double y){
        tree t;

        t = RRT(x, y);
        if(t != NULL){
                while(t != NULL){
                        //inserts in the list
```



```
                        insertListActions(t -> x, t -> y);

                        t = t -> parent;
                }

                printList();
        }
        else
                printf("no solution found.\n");
}
```