IEEE Computer Society Problem

IEEE Computer Society was working on a little secret project in one of their labs around the world to build a little Robbie with human sense and lots of intelligence. While being in the lab all the time, the little Robbie decided to break himself out and search for a new adventure. While the little Robbie was looking for new adventure he decided to enter an ancient Incas maze located nearby. As little Robbie is too young, he has not yet learned any sophisticated ways to get out of the maze, but fortunately he was taught at school that the right-hand wall follower algorithm is always guaranteed to lead to an exit in the case of a simply connected maze. Nevertheless, this maze is a dynamic one and every time Robbie makes a move, the maze topology changes. Would Robbie be able to finally get out of the maze?

Task

Your task is to develop a program that can efficiently simulate the scenario of little Robbie being trapped in a dynamic maze.

a) The Linear Congruential Generator

Since randomness will be necessary for this problem, your first subtask includes the development of a custom pseudorandom number generator that will be used in all cases where a random number has to be drawn.

A Linear Congruential Generator (LCG) represents one of the oldest and best-known pseudorandom number generator algorithms.

The generator is defined by the recurrence relation:

$X_{n+1} = a * X_n + c \pmod m$

where:

- X is the sequence of pseudorandom values,
- m is the modulus, m > 0
- a is the multiplier 0 < a < m

- c is the increment $0 < c < m$
- X_0 is the seed or start value $0 < X_0 m$

source:wiki

For the case of our LCG, we will assume the X, m, a, c and X_0 values will be of *integer type*. Also we will always use

$m = 2^{48}$
a = 1664525
c = 1013904223

and only the start seed value will change (will be an input parameter provided from the standard input stream).

Whenever a random positive integer value within a specific range [min, max] is required, then such a value may generated using the absolute value of our custom LCG and the modulus operator. For example, if we want to draw a random integer in the range [1, 100] (i.e. between 1 and 100 inclusive) then the following formula can be used:

$$rand_{num} = X_{n+1} (\mathrm{mod}\ 100) + 1$$

For example, the first 15 randomly generated values for

$X_0 = 999$
$m = 2^{48}$
a = 1664525
c = 1013904223

alongside with their corresponding [1,100] mapping are presented at Table 1.

ID	Random Number	Correspondence to the [1, 100] range
1	-1618202598	97
2	-465452879	78
3	-673889700	1
4	-1515193845	44
5	789710830	31
6	-1782567307	6
7	-215945904	3
8	-528949137	36
9	-1227517182	81
10	-116640135	34
11	294841732	33
12	-580150509	8
13	2139776342	43
14	-1069785923	22

b) The Maze

The maze is composed of M rows and N columns (M and N being positive integer values) and consequently of M*N cells. Such an exemplary 10x10 maze is graphically depicted at Figure 1.

Regarding the maze, please also consider the following definitions as they will be used throughout the problem description:

- Corner Cells: The four cells located at the corners of the maze (highlighted in gray at Figure 1). These cells always accommodate the four pillars of the maze and are inaccessible to the robot. Also they cannot be used as start or exit positions.
- Border Cells: The cells positioned at the four edges of the maze - excluding the Corner cells (highlighted in red at Figure 1). These are the only cells that can be used as start and exit positions.
- Inner Cells: All the remaining cells of the maze - excluding the corner and border cells (highlighted in yellow at Figure 1). These cells cannot be used as start or exit positions, but may be accessible by the robot. Start Position: The position where the robot starts when it first enters the maze. This cell can be any of the Border cells.
- Maze Exit: The cell leading to the exit of the maze. This can be any of the Border cells, but it must be different to the start position (i.e. the exit and the start position cannot be located on the same Border cell)

Each one of the maze cells may or may not have a wall on each one of the 4 directions: North, South, East, and West (please refer to the compass at Figure

1). Walls (and equivalently openings) that are shared by adjacent cells will be accounted for both cells respectively. For example the East side wall of the rightmost cell in Figure 2b, should also be considered as a West side wall for the middle cell. Equivalently, the East opening of the middle cell should also be considered as an opening for the West side of the leftmost cell. The decision about whether a specific cell will have a wall on each one of the four possible directions will be based on a given positive integer Probability value

$1 ? P_{wall} < 100$

that will be received as input to the program from the standard input stream.

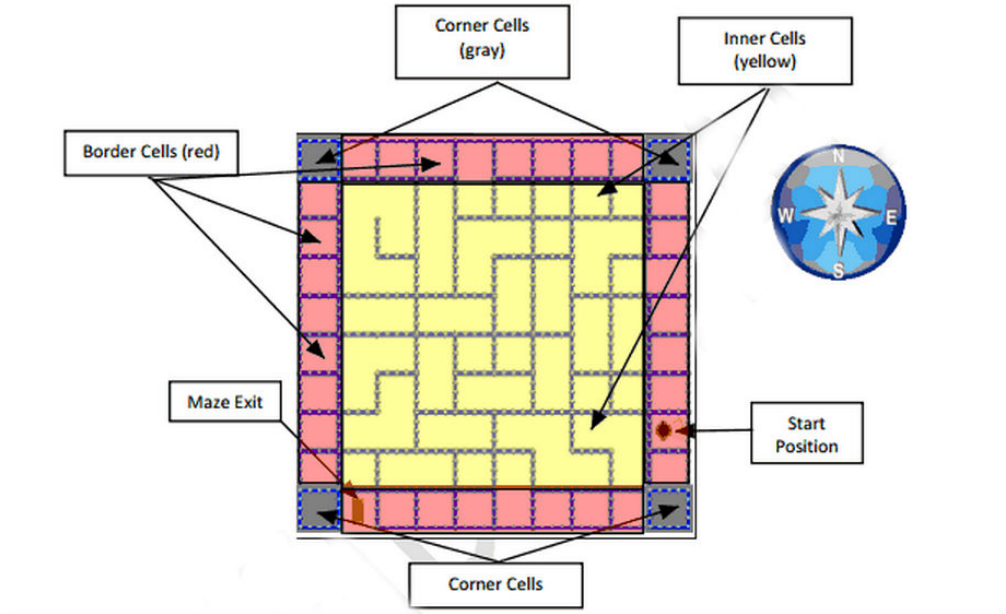


Figure 1. Graphical Depiction of a random maze topology. The pillars are located at the four corners of the maze and are highlighted in gray. The Border cells are highlighted in red and the start and exit positions are also depicted. The yellow highlighting signifies the inner cells of the maze.

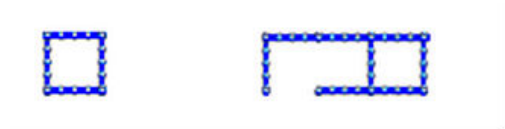


Figure 2. (a) A maze cell having a border on all 4 directions: North, South, East, and West. (b) Three adjacent maze cells. The first cell (on the left) has a North and West wall, the second cell (in the middle) has a North, South and East wall and the third cell (on the right) has walls on all 4 directions.

Also, please take in account the following *Ordering Rules*:

1. Ordering Rule 1: Whenever a full traversal of the maze is required, it should always be performed by row (meaning that starting from the first row of the maze, at first all the columns of the current row should be visited and then the program should proceed to parse all the columns of the subsequent row).

2. Ordering Rule 2: Whenever a full traversal of the four possible directions is required, it should always be performed according to this order: a) North, b) South, c) West and d) East.

Regarding the construction of the maze, the following steps should be followed in the specified order:

1. An M x N maze should be constructed based on the given M and N values.

2. For the starting maze topology, all maze cells (i.e. Corner, Border and Inner Cells) should be considered to have walls on all four directions (North, South, East, and West).

3. Then, one Border Cell should be selected at random as the Start Position of the maze. This should be done by drawing a random number from the custom Linear Congruential Generator between [1, n], where n is the total number of Border Cells. The order of the Border Cells should be as described by the Ordering Rule 1 (i.e. at first all the Border Cells of the 1st row should be considered, then the ones at the 2nd row and so forth).

4. Afterwards, another Border Cell (different to the Start Position) should be selected at random as the Maze Exit. This should be done by drawing a random number from the custom Linear Congruential Generator between [1, k], where k is the total number of Border Cells, after having removed the Border Cell that is used for Start Position from the set of Border Cells. Then, a. If the Maze Exit is located at the Northern Edge of the Maze, then it should have a North Side opening.
   b. If the Maze Exit is located at the Southern Edge of the Maze, then it should have a South Side opening.
   c. If the Maze Exit is located at the Eastern Edge of the Maze, then it should have an East Side opening.
   d. If the Maze Exit is located at the Western Edge of the Maze, then it should have a West Side opening.

5. Finally, for each one of the Inner Cells (again the inner cells should be traversed by row as described at the Ordering Rule 1):
   a. A random number between [1,4] should be drawn using the custom Linear Congruential Generator:

- If this random values equals to 1, then this Inner Cell should have an opening at the Northern Side and equivalently the cell positioned northern to this cell should have an opening at its Southern side.

- If this random values equals to 2, then this Inner Cell should have an opening at the Southern Side and equivalently the cell positioned southern to this cell should have an opening at its Northern side.

- If this random values equals to 3, then this Inner Cell should have an opening at the Western Side and equivalently the cell positioned western to this cell should have an opening at its Eastern side.

- If this random values equals to 4, then this Inner Cell should have an opening at the Eastern Side and equivalently the cell positioned eastern to this cell should have an opening at its Western side.

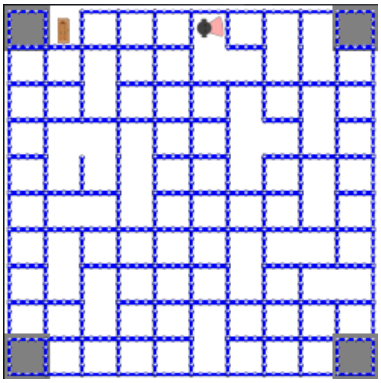


Figure 3 shows the starting maze topology when using

$X_0 = 999$
$m = 2^{48}$
$a = 1664525$
$c = 1013904223$

as initial values for the custom Linear Congruential Generator and following the above steps.

c) The Robot

The robot can only look at one direction at a time (North, South, West or East) and it can only perform one of the following two moves:
a. Step: If there is no wall at the direction the robot is currently looking at, and also there is a wall to the right of the robot then it can move one step towards the direction it is looking at (i.e. it may progress to the next cell)
b. Turn: If there is a wall at the direction the robot is looking at, or there is no wall to its right then the robot should make a turn. The turn should always be performed in a clockwise direction.

When the robot enters the maze for the first time it should always look towards the east. Then it should start traversing the maze trying to find the exit simply by following the right-hand wall follower algorithm, meaning that it should keep its right hand in contact with the wall located to its right

see also Maze Solving Algorithm for more details on the wall follower algorithm

Every time the robot makes a move (either a step or a turn), the Inner Cells of the maze should be dynamically updated as follows:

1. For each Inner Cell (traversed according to the Ordering Rule 1) each one of the 4 directions (ordered as North, South, West and East – see Ordering Rule 2) should be re-examined. In particular:

a. A random number between [1, 100] should be drawn from the LCG and if this random value is smaller to a given (as input to the program) integer Probability value 1 ? P_{wall} ? 100 then this innerl cell should ahve a wall at this

direction. If the random value is larger or equal to P_{wall} a. then this Inner Cell should have an opening at this direction. Every time a wall appears or disappears at a specific cell, the corresponding state of its adjacent cells should also be updated (for example, if a cell gets to have a wall on the North Side, then the cell positioned northern to this cell should also be updated to have a wall at its Southern side).

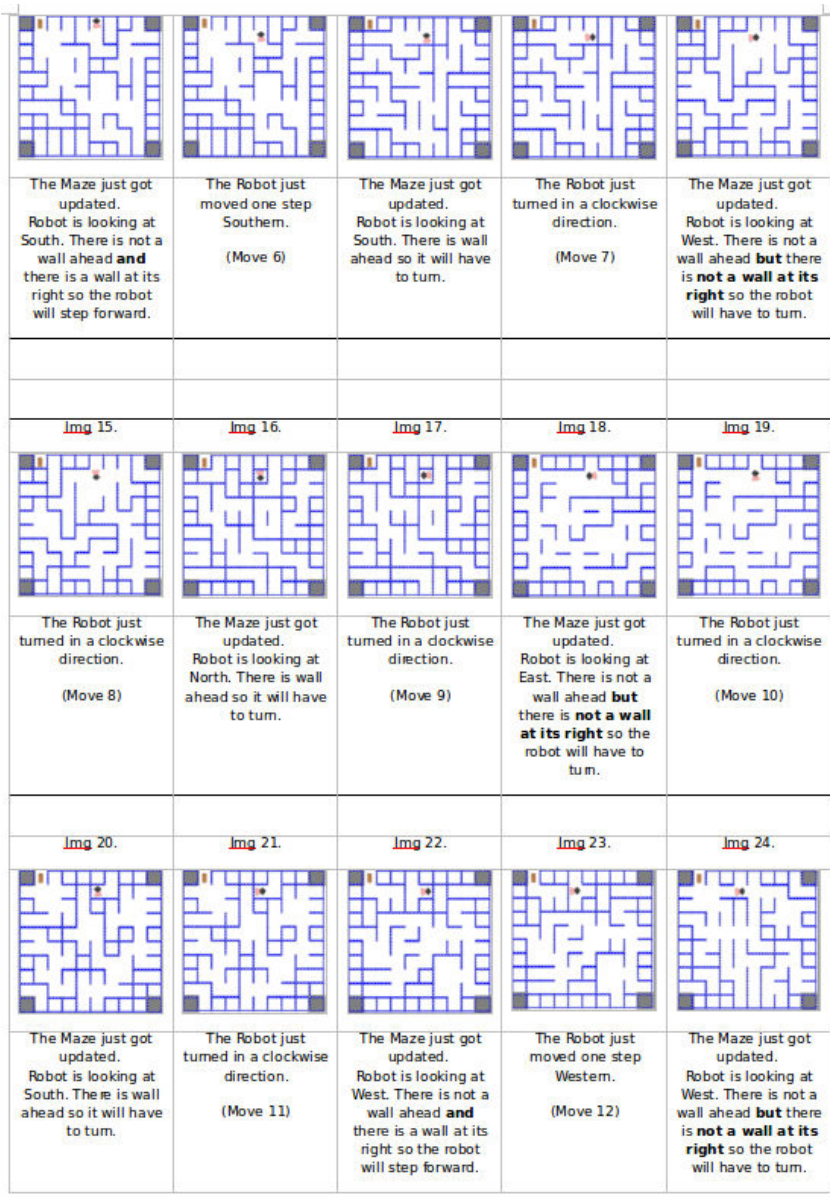
Note: During this maze update step, it might happen that a cell (when traversed in its order) may not to have a wall on one directions (e.g. its South Side), but it finally gets to have a wall at that direction because of an adjacent cell of it. For example, it might happen that a cell (A) was decided not to have a wall at its Southern Side when this cell was traversed, but it finally ended up with a Southern Side wall because another Cell (B) positioned to the South of Cell (A) was later on traversed and was randomly assigned a wall to its Northern Side.

Figure 4, shows the robot and maze topologies for a set of 12 robot moves (steps or turns) according to the aforementioned guidelines using

$X_0 = 999$
$m = 2^{48}$
$a = 1664525$
$c = 1013904223$

as initial values for the custom Linear Congruential Generator.

Starting Topology	Img 1.	Img 2.	Img 3.	Img 4.
				
Initial Maze Topology. Robot is looking at East. There is wall ahead so it will have to turn.	The Robot just turned in a clockwise direction. (Move 1)	The Maze just got updated. Robot is looking at South. There is wall ahead so it will have to turn.	The Robot just turned in a clockwise direction. (Move 2)	The Maze just got updated. Robot is looking at West. There is wall ahead so it will have to turn.
Img 5.	Img 6.	Img 7.	Img 8.	Img 9.
				
The Robot just turned in a clockwise direction. (Move 3)	The Maze just got updated. Robot is looking at North. There is wall ahead so it will have to turn.	The Robot just turned in a clockwise direction. (Move 4)	The Maze just got updated. Robot is looking at West. There is wall ahead so it will have to turn.	The Robot just turned in a clockwise direction. (Move 5)



Since the robot fuel is not endless, the robot can only perform a limited number of moves (steps or turns) before it runs out of fuel. The number of moves that the robot can make before it runs out of fuel will be given to the program as input from the standard input stream. The program should terminate when the robot reaches the maze exit or when the maximum number of moves has been reached.

Input For the LCG, you should always use the following starting values:

$m = 2^{48}$
$a = 1664525$
$c = 1013904223$

Also, your program will receive the following input (one parameter per line) in the following order from the Standard Input Stream:

1. Seed: A positive integer value used as initial seed (start value X0) for the LCG.

2. M: A positive integer value representing the number of rows of the maze.
3. N: A positive integer value representing the number of columns of the maze.
4. $1 < P_{wall} < 100$: A positive integer value representing the probability of having a wall on each one of the four possible directions (North, South, West, East).
5. Moves: A positive integer value representing the number of moves that the robot can make before running out of fuel.

Output

- If Robbie manages to find the Maze Exit before it runs out of fuel the following message should be printed to the Standard Output Stream:

  Robbie got out of the maze in moves.

  e.g. Robbie got out of the maze in 2798 moves.

- If Robbie runs out of fuel before managing to find the exit, then the following message should be printed to the Standard Output Stream:

Robbie was trapped in the maze.

Note: Each sentence ends with a dot followed by newline character.

Sample Input 1

```
999
10
10
50
1000
```

Sample Output 1

```
Robbie got out of the maze in 179 moves.
```

Sample Input 2

```
1100
20
30
50
3000
```

Sample Output 2

```
Robbie got out of the maze in 1962 moves.
```

Problem Author: IEEE

Suggest Edits

Emacs | Normal | Vim

Select Language: C

save code

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

int main() {

    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}

```

Line: 1 Col: 1 Count: 190

☐ **Use a custom test case**

Upload Code as File

Compile & Test | Submit Code