Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2023-2024

Take-Home Exam 2 – Maze Solver
Due: 5 August 2024 11.55pm (SHARP)

DISCLAIMER:

Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different from the provided samples; however checking these cases is highly encouraged and recommended.

You can NOT collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!

Introduction

In this Take-Home Exam (THE), you will be practicing the concept of 2D Linked Lists. You will implement a maze game using **a 2D linked list structure**. See Figure 1 for the visualization of a sample of the data structure. The tasks include reading the content of a file that represents the maze, constructing a 2D Linked List and then populating the content of the file to the 2D linked List. In the next sections, we will mention the input and output format, the rules for the maze game and the users' interactions with the game.

IT IS A MUST TO IMPLEMENT THIS THE USING A 2D LINKED LIST STRUCTURE. Any other types of implementations that do not adhere to this rule will be severely penalized. Whether to use a class-based implementation of the 2D Linked List, or not, is up to you. It is of the highest importance to deallocate the memory you allocated during your program before terminating.

Inputs to Your Program

The first input to your program is the file name containing the structure of the maze. The maze is represented as a grid where each cell can be a path (.), wall (#), item (I), player start position (X), or exit (E). In case the input file name is invalid, the program will print a predefined message and quit. Please refer to the sample runs for the format and content of all the messages.

During the execution of the program, the user will be interacting with the game by entering one of the following uppercase English letters as given in Table 1. In case of any other inputs, the input will be considered as invalid and a specific message will be displayed. Please refer to the detailed explanation list of the inputs later in the document for the specific messages.

User Input	Expected Action
R	Move the player position one node to the right, if possible
L	Move the player position one node to the left, if possible
U	Move the player position one node in the upward direction, if possible
D	Move the player position one node in the downward direction, if possible
P	Print the current state of the maze
C	Collect an item at the current position, if possible
Q	Quit the game

Table 1: Input to the program and their expected actions.

Format of the Input File Content:

The maze file will have a regular rectangle shape. i.e., all the rows will be of the same length(i.e., the number of characters on each line will be the same). So, you can safely assume that while reading the files' content and while constructing the 2D linked list.

The maze file will contain characters (called `cellContent`) in the format given in Table 2.

Symbol	Indication
#	Represents a wall
.	Represents a valid path cell
I	Represents an item
X	Represents the player's position
E	Represents the exit

Table 2: Symbols found in the input file and their meaning

Notes:

It is guaranteed that any input file that will be used in this THE will be of the aforementioned valid formats; i.e., no invalid files will be used in this THE, and your program doesn't have to check the validity of the file's content. Once your program manages to open the file and read its content, you are good to go with the rest of the program's flow. In other words, the file will only contain characters from this set of characters: {(.), (#), (I), (X), or (E)}.

No empty spaces nor empty lines will be found in any file. No empty input files will be used in this THE. However, there is no limitation on the number lines (rows) nor the number of columns in the file. As mentioned above, the number of characters on each line in the file will be the same.

Also, in any input file:

- There will be exactly one 'X' to indicate the starting position of the player.
- There will be exactly one 'E' to indicate the exit location.
- There can be zero or more multiple 'I' characters in the maze to represent items.
    - The value for any item found in the maze is fixed to the value *204* :)

Finally, you should read the file only for once, and then proceed with the rest of the program's flow.

Data Structure to be Used

You must use a 2D linked list for the implementation of the maze, as visualized in Figure 1. Each node in the linked list represents a cell in the maze and should be able to connect to all of its adjacent nodes (up, down, left, right). You should encapsulate the cell content, and pointers to adjacent nodes in a struct.

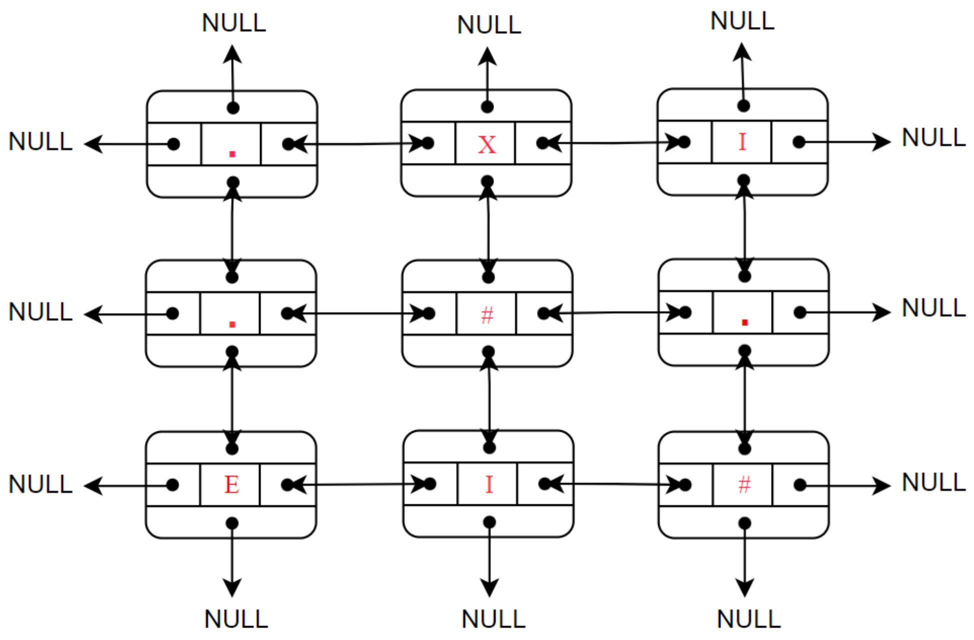

Figure 1: A sample representation of the 2D Linked List structure used in this THE

Below is a struct for the maze node in the 2D linked list that you can consider implementing.

```
struct MazeNode {
    char cellContent;
    MazeNode *right, *left, *up, *down;
}
```

Each `MazeNode` should have a `char` variable to store the content of the cell, and 4 `MazeNode` pointers, in the four directions to connect the node to its neighbors. Note that you may add to the content of the `MazeNode` struct if you find a need to.

Output of Your Program

Your program will have different types of outputs according to the input entered by the user. Each input will produce a specific output to the screen. The main inputs have been briefly described in Table 1 above.

When the maze is successfully loaded from the file in the beginning of the program, a specific message will be displayed "Maze loaded. Start exploring!", and the execution of the program will continue. On the other, if the program cannot open the file, the following message will be displayed and the program will terminate.

```
Unable to open file.
Failed to load maze. Exiting...
```

Below we explain the main inputs in detail:

1. **Print Maze (Input 'P'):**

   This input prints the current status of the maze to the screen. When the maze is displayed, there will be the player's position (X), an exit cell (E) and the rest of the cells will be a combination of (.) for valid path cells, (I) for items, and (#) for walls.

   Also, the score will be displayed when this input is entered by the user.

2. **Move Player (Inputs 'U', 'D', 'L', 'R'):**

   When a user inputs any of these values, the player's position will be moved on the maze, if applicable, and a message will be displayed to indicate either the direction of movement, if it was successful (e.g. "You moved right."), or any of the two following messages in case the movement was not possible.

- When there is a wall

  ```
  Cannot move: There is a wall in that direction.
  ```

- When it's out of the maze's scope

  ```
  Cannot move: Out of the maze bound.
  ```

3. **Collect Item (Input 'C')**

This input allows the user to collect an item, if it exists on the current position cell; i.e., if the player's position is on one of the cells that contain an item.

There are two cases:

- If the item is successfully collected, a message will be displayed to the screen and the value of the item - which is always 204:) will be added to the score of the player.

  ```
  Item collected! Your score is now TotalItemScore.
  ```

  Note here that the cell that was previously an (I), once collected, will become a regular path cell (.)

- If, on the other hand, no item is present at the player's position at that time, the following message will be printed and the score will not be affected.

  ```
  No item to collect here.
  ```

4. **End the Game (Input 'Q'):**

This inputs ends the game, prints the final score and terminates the program

A message similar to the one below will be displayed:

```
Game ended by the player.
Final score: TotalItemScore
```

5. **Invalid input**

In case the user enters any invalid input, a specific message will be displayed:

```
Invalid input. Please try again.
```

When the player reaches to the exit cell (E) on the maze, the game ends and the following messages will be displayed:

```
Congratulations! You reached the exit!
Final score: TotalItemScore
```

Note about printing the score:

Please keep in mind that *TotalItemScore* is a placeholder for the value of the score at that moment in during the game. Please refer to the "Sample Runs" section for examples with exact numbers.

Note about stepping on an (I) cell:

Just passing by an (I) cell doesn’t mean collecting the item, when the player (X) steps on an (I) cell, the program should print a message:

```
"You found an item worth 204 points!"
```

In this case, the user can collect the item by entering the (C) input. If on the other hand the user decided not to collect the item, the item will stay on this cell, and should be visible again in the maze once X steps out of that cell.

Please refer to the "Sample Runs" below for any specifics of the outputs.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The **bold** phrases are the standard input (cin) taken from the user. You have to display the required information in the same order and with the same words/spaces as here; in other words, there must be an exact match!

You will submit your code through CodeRunner, and you can always use the "Show differences" feature to make it easier for you to figure out the mismatches in your output, if any.

Sample Run 1:

```
Enter the maze file name:
myfile
Unable to open file.
Failed to load maze. Exiting...
```

<u>Sample Run 2</u>:

```
Enter the maze file name:
m1.txt
Maze loaded. Start exploring!
Enter input (R/L/U/D/P/C/Q):
P
E...
....
#..X
Current score: 0
Enter input (R/L/U/D/P/C/Q):
Q
Game ended by the player.
Final score: 0
```

<u>Sample Run 3</u>:

```
Enter the maze file name:
m2.txt
Maze loaded. Start exploring!
Enter input (R/L/U/D/P/C/Q):
f
Invalid input. Please try again.
Enter input (R/L/U/D/P/C/Q):
P
###
...
X..
I.#
..I
#E#
Current score: 0
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
P
###
...
...
X.#
..I
#E#
Current score: 0
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
P
###
```

```
...
...
I.#
X.I
#E#
Current score: 0
Enter input (R/L/U/D/P/C/Q):
U
You moved up.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
C
Item collected! Your score is now 204.
Enter input (R/L/U/D/P/C/Q):
P
###
...
...
X.#
..I
#E#
Current score: 204
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
P
###
...
...
.X#
..I
#E#
Current score: 204
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
P
###
...
...
..#
.XI
#E#
Current score: 204
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Congratulations! You reached the exit!
Final score: 204
```

<u>Sample Run 4</u>:

```
Enter the maze file name:
m3.txt
Maze loaded. Start exploring!
Enter input (R/L/U/D/P/C/Q):
P
####.##
.X..I.#
#.#.#.#
#I..I.E
##II###
Current score: 0
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
Enter input (R/L/U/D/P/C/Q):
P
####.##
X...I.#
#.#.#.#
#I..I.E
##II###
Current score: 0
Enter input (R/L/U/D/P/C/Q):
L
Cannot move: Out of the maze bound.
Enter input (R/L/U/D/P/C/Q):
D
Cannot move: There is a wall in that direction.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
C
Item collected! Your score is now 204.
Enter input (R/L/U/D/P/C/Q):
P
####.##
....X.#
#.#.#.#
```

```
#I..I.E
##II###
Current score: 204
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
D
Cannot move: There is a wall in that direction.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Congratulations! You reached the exit!
Final score: 204
```

Sample Run 5:

```
Enter the maze file name:
m1.txt
Maze loaded. Start exploring!
Enter input (R/L/U/D/P/C/Q):
P
E...
....
#..X
Current score: 0
Enter input (R/L/U/D/P/C/Q):
R
Cannot move: Out of the maze bound.
Enter input (R/L/U/D/P/C/Q):
P
E...
....
#..X
Current score: 0
Enter input (R/L/U/D/P/C/Q):
D
Cannot move: Out of the maze bound.
Enter input (R/L/U/D/P/C/Q):
P
E...
....
#..X
Current score: 0
```

```
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
Enter input (R/L/U/D/P/C/Q):
P
E...
....
#.X.
Current score: 0
Enter input (R/L/U/D/P/C/Q):
D
Cannot move: Out of the maze bound.
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
Enter input (R/L/U/D/P/C/Q):
L
Cannot move: There is a wall in that direction.
Enter input (R/L/U/D/P/C/Q):
P
E...
....
#X..
Current score: 0
Enter input (R/L/U/D/P/C/Q):
U
You moved up.
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
Enter input (R/L/U/D/P/C/Q):
P
E...
X...
#...
Current score: 0
Enter input (R/L/U/D/P/C/Q):
L
Cannot move: Out of the maze bound.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
P
E...
.X..
#...
Current score: 0
Enter input (R/L/U/D/P/C/Q):
U
You moved up.
```

```
Enter input (R/L/U/D/P/C/Q):
P
EX..
....
#...
Current score: 0
Enter input (R/L/U/D/P/C/Q):
U
Cannot move: Out of the maze bound.
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
Congratulations! You reached the exit!
Final score: 0
```

Sample Run 6:

```
Enter the maze file name:
m5.txt
Maze loaded. Start exploring!
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#X...#.#
#.#.#...
II..#.E.
##I##I##
Current score: 0
Enter input (R/L/U/D/P/C/Q):
C
No item to collect here.
Enter input (R/L/U/D/P/C/Q):
U
Cannot move: There is a wall in that direction.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
R
Cannot move: There is a wall in that direction.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#...X#.#
```

```
#.#.#...
II..#.E.
##I##I##
Current score: 0
Enter input (R/L/U/D/P/C/Q):
D
Cannot move: There is a wall in that direction.
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#..X.#.#
#.#.#...
II..#.E.
##I##I##
Current score: 0
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
D
Cannot move: There is a wall in that direction.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#....#.#
#.#.#...
II.X#.E.
##I##I##
Current score: 0
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#....#.#
#.#.#...
IX..#.E.
##I##I##
Current score: 0
Enter input (R/L/U/D/P/C/Q):
```

```
U
You moved up.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#....#.#
#X#.#...
II..#.E.
##I##I##
Current score: 0
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
C
Item collected! Your score is now 204.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#....#.#
#.#.#...
IX..#.E.
##I##I##
Current score: 204
Enter input (R/L/U/D/P/C/Q):
U
You moved up.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#....#.#
#X#.#...
I...#.E.
##I##I##
Current score: 204
Enter input (R/L/U/D/P/C/Q):
U
You moved up.
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#.X..#.#
#.#.#...
I...#.E.
##I##I##
Current score: 204
Enter input (R/L/U/D/P/C/Q):
```

```
U
You moved up.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
P
##X###..
#....#.#
#.#.#...
I...#.E.
##I##I##
Current score: 204
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
P
##I###..
#.X..#.#
#.#.#...
I...#.E.
##I##I##
Current score: 204
Enter input (R/L/U/D/P/C/Q):
U
You moved up.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
C
Item collected! Your score is now 408.
Enter input (R/L/U/D/P/C/Q):
P
##X###..
#....#.#
#.#.#...
I...#.E.
##I##I##
Current score: 408
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#.X..#.#
#.#.#...
I...#.E.
##I##I##
Current score: 408
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
```

```
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#....#.#
#.#.#...
IX..#.E.
##I##I##
Current score: 408
Enter input (R/L/U/D/P/C/Q):
L
You moved left.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#....#.#
#.#.#...
X...#.E.
##I##I##
Current score: 408
Enter input (R/L/U/D/P/C/Q):
C
Item collected! Your score is now 612.
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#....#.#
#.#.#...
X...#.E.
##I##I##
Current score: 612
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#....#.#
#.#.#...
.X..#.E.
##I##I##
Current score: 612
Enter input (R/L/U/D/P/C/Q):
R
You moved right.
```

```
Enter input (R/L/U/D/P/C/Q):
D
You moved down.
You found an item worth 204 points!
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#....#.#
#.#.#...
....#.E.
##X##I##
Current score: 612
Enter input (R/L/U/D/P/C/Q):
C
Item collected! Your score is now 816.
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#....#.#
#.#.#...
....#.E.
##X##I##
Current score: 816
Enter input (R/L/U/D/P/C/Q):
U
You moved up.
Enter input (R/L/U/D/P/C/Q):
P
##.###..
#....#.#
#.#.#...
..X.#.E.
##.##I##
Current score: 816
Enter input (R/L/U/D/P/C/Q):
Q
Game ended by the player.
Final score: 816
```

Some Important Rules

In order to get full credit, your program must be efficient, modular (with the use of functions), well commented and properly indented. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments may decrease your grade in case that we detect them. When we grade your THEs, we pay attention to these issues. Moreover, **we may test your programs with very large test cases**. Hence, take into consideration the efficiency of your algorithms other than correctness.

Sample runs give a good estimate of how correct your implementation is, however, we will test your programs with different test cases and **your final grade may conflict with what you have seen on CodeRunner**. We will also **manually** check your code, indentations and so on, hence do not object to your grade based on the **CodeRunner** results, but rather, consider every detail on this documentation. **So please make sure that you have read this documentation carefully and covered all possible cases, even some other cases you may not have seen on CodeRunner or the sample runs**. The cases that you *do not need* to consider are also given throughout this documentation.

Submit via SUCourse ONLY! Paper, e-mail or any other methods are not acceptable.

The internal clock of SUCourse might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your THE on time:

"No successful submission on SUCourse on time = A grade of zero (0) directly."

What and where to submit (PLEASE READ, IMPORTANT)

It'd be a good idea to write your name and last name in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your full name is "Duygu Karaoğlan Altop", and if you want to write it as comment; then you must type it as follows:

// Duygu Karaoglan Altop

You should copy the full content of the .cpp file and paste it into the specified "Answer" area in the relevant assignment submission page on SUCourse. ***Please note that the warnings are also considered as errors on CodeRunner, which means that you should have a compiling and warning-free program***.

Since the grading process will be automatic, you are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be zero (0). Any tiny change in the output format will result in your grade being zero (0), so please test your programs yourself, and against the sample runs that are available at the relevant assignment submission page on SUCourse.

In the CodeRunner, there are some visible and invisible (hidden) test cases. You will see your final grade (including hidden test cases) before submitting your code. There is no re-submission. You don't have to complete your task in one time, you can continue from where you left last time but you should not press submit before finalizing it. Therefore, you should make sure that it's your final solution version before you submit it. Also, we still do not suggest that you develop your solution on CodeRunner but rather on your IDE on your computer.

You may visit the office hours if you have any questions regarding submissions.

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse.

Plagiarism

Plagiarism is checked by automated tools, and we are very capable of detecting such cases. Be careful with that. Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT send any part of your code to your friends by any means or you might be charged as well, although you have done your THE by yourself. THEs are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

In case of plagiarism, the rules on the Syllabus apply.

Good Luck!
Ahmed Salem, Duygu Karaoğlan Altop