Advanced Topics in Programming - Exercises

Requirements and Guidelines

The exercises in the course would require you to implement a search in a maze.

Exercise 1

In this exercise the program should get an input file which represents a maze and run an algorithm that attempts to solve the maze.

Input File

The maze input file is a text file in the following format:
Line 1: maze name / description - internal for the file, not used by the program
Line 2: MaxSteps=<num> MaxSteps for solving this maze. We assume different mazes may require different MaxSteps.
Line 3: Rows=<num> Number of Rows in maze
Line 4: Cols=<num> Number of Cols in maze
Lines 5 and on: the maze itself, as described below

Allow spaces in lines 2,3,4 - everywhere except inside word or number.

The actual size of the maze is determined by the values in lines 3 and 4. If there are missing lines in the maze (less than defined by Rows) the missing lines should be filled in with empty lines (all spaces). If there are missing columns (less than defined by Cols) they should be filled in with spaces.
In case the file contains additional rows and/or cols beyond the required number as set in lines 3 and 4, program shall ignore the redundant lines / cols.

Chars in the maze file and their meaning:

- # -- represents a wall
- space -- represents a pass
- @ -- represents the player in the maze (initial position of the player)
- $ -- represents the end of the maze (the treasure that we seek)

The chars @ and $ must appear once and only once!
All other chars not listed above are forbidden and wrong!

Example of a valid maze:

```
Nice simple maze
MaxSteps = 10
Rows = 4
Cols = 10
#####
# @ #    #
#   #  $ #
    #####
```

A possible solution for above maze, one of several possible shortest path, would be (steps described as arrows just for illustration):

```
  0123456789
0 #####  ↓←←
1 # @ #  ↓ #
2 # ↓ #  $ #
3 ←←← #####↓
```

Player Movement in the Maze

Note: the player doesn't know the dimensions of the maze nor his own 'location'!
The player can move in any of the four directions: UP, RIGHT, DOWN, LEFT.
The player can also choose to set a BOOKMARK in the current position. This helps the player to navigate throughout the maze. Setting a bookmark is considered a move, so the player stays in its current location during this turn.
There is only one single bookmark per player, so when a bookmark is set, previously located bookmark, if any, is gone.
The player can't move into a wall, a proper info would be revealed as described below.
The player can move beyond the last line / last column - that would bring the player back to the first (zero indexed) line / column respectively, without any special notice, as long as there is no wall preventing this "tunneling". Same goes for getting "back" beyond line/column 0.

The Flow of the Game

1. The game manager creates an object of class Player
2. Loop while steps <= MaxSteps
    a. The game manager calls player's `move()` method which should return enum value from the options: UP, RIGHT, DOWN, LEFT, BOOKMARK
    b. If the move brings the player to the $, stop the loop and print to screen: "Succeeded in <num> steps"
    c. If the move is into a wall, manager calls player's method `hitWall();`
    d. If the move is into a preset bookemark (not including bookmark that was currently set in this turn!), manager calls player's method `hitBookmark();`
3. If maze was not solved print to screen: "Failed to solve maze in <MaxSteps> steps"

You should implement in the first exercise:
- The Game manager
- A player that tries solving a maze

The program gets the maze file as a first command line argument, you should support both options:
- If the argument starts with / it is an absolute path
- If the argument doesn't start with / look for it in the current working directory

Output File

The second command line argument is the name of an output file to create.

- If the argument starts with / it is an absolute path
- If it doesn't start with / it should be created in the current working directory

You should create the output file according to the path+name given in the second command line argument.

The output file should be a text file (no special suffix: with the suffix provided by the user in the command line parameter), inside the file there would be a line per each step done by the player, from the options: U, R, D, L, B

Last line in the file would be: X -- in case of failing to solve the maze

Or: ! -- in case the maze was solved

(Thus total number of lines in output file would always be number steps + 1).

Errors

If there were errors while reading or analyzing the input file, you should print to screen the following errors. Note that since the check is automatic, printouts must be exact.

- Possible maze file argument errors (one of the below):

```
Missing maze file argument in command line
Command line argument for maze: <arg> doesn't lead to a maze file or leads to a file that cannot be opened
```

- Possible output file argument errors:

```
Missing output file argument in command line
Command line argument for output file: <arg> points to a bad path or to a file that already exists
```

Note

1. In case of a problem with the maze file, check also the output file to report both errors if exist.
2. In case of a problem with the output file - don't run the player, BUT check errors below to report additional errors inside the maze file itself, if such exist.

- Input file exists but seem to have a wrong format (lines 2-4)

```
Bad maze file header:
expected in line 2 - MaxSteps = <num>
got: <line 2 content>
expected in line 3 - Rows = <num>
got: <line 2 content>
expected in line 4 - Cols = <num>
got: <line 2 content>
```

Above is an example, report ONLY actual errors, but collect all errors and print them in the order in this document!

Note: you can decide if "strange" input like: `Cols=010` (as 10 columns) is OK or not, however: `Cols = a` or `Cols = 10 a` or `Cols = 10a` -- are all bad

Cont' next page

- Problems with the maze itself (only if lines 2-4 are legal):
  Recall that you ignore rows and columns that are beyond Rows and Cols size
  ```
  Bad maze in maze file:
  Missing @ in maze
  Missing $ in maze
  More than one @ in maze
  More than one $ in maze
  Wrong character in maze: <char> in row <row>, col <col>
  ```
  Above is an example, report ONLY actual errors, but collect all and in order!
  The last error, reporting wrong characters:
  1. should appear as a full line per each wrong character
  2. scanning complete row than getting down to next row
  3. for TAB (ascii = 9) line should say:
     ```
     Wrong character in maze: TAB in row <row>, col <col>
     ```
  4. ignore \r (ascii = 10) if appears as the last character of a line

In case of any error, don't run the player - no output file should be created.

Extra notes and clarifications:

You should replace triangle brackets (<>) with relevant string according to their content. For example: in case of an illegal character '+' in line 3 column 6 of the input file, the following message should print:

```
Wrong character in maze: + in row 3, col 6
```

(note that the triangle brackets are gone :-)

Note that all output shall be exact. While in the Input extra spaces can be ignored, in the output we expect the lines to be exactly as appears in this doc, spaces included.

No use of external libraries is allowed except for the standard library. In case you want to use an external library please post a request in the forum of the exercise and it will be considered.

Player algorithm requirements: in this exercise we expect a reasonable player (do your best effort...) but it is OK if you have a player that doesn't solve every maze. We do expect that the player will move (i.e. "only bookmarking" player is not enough) and will try not to repeat visited cells (so we expect the player to remember somehow visited locations).