CS 201
Spring 2015
Program 4
Recursion

DRAFT
Assignment details are subject to change

Recursion problems are an alternate way to solve problems. Instead of using loops to iterate through a solution, you can use recursion to break a problem down into smaller problems, then solve those instead. Once smaller ones are solved, you can use those solutions to solve the larger ones.

For this assignment, you will be given a maze. You will write a program to solve it recursively. To solve it, you will have a starting position, and need to navigate to an ending spot. This is a pretty easy recursive problem to solve. However, there's a minor catch: you need to collect an item first, then find the end.

Input File

The program should ask the user for an input file. Once the program is able to open it, it will consist of the following:

- Two integers, the first being the width and the second being the height. These will never exceed 20x20. (Much bigger and the problem may take too long to solve.)
- A *width* x *height* grid of chars:
    - * (asterisk) is a wall
    - o (lowercase "O") is the starting position of the maze
    - ! (exclamation point) is the ending position of the maze
    - @ (at sign) is the item you need to collect

For example, a sample input file would have the following:

```
20 20
********************
*o  *    * *       *
*** ** *** * ** ** *
***      * * *  *  *
*   **** *   * *** *
* * *  * *****   * *
* * ** *       *** *
* *    * *******  @*
* ******   *   *****
*   * *  * *** * * *
*****   *  * * * * *
*   ***** ** * * * *
* *        *   * * *
* * * **** ** ** * *
***** *     *      *
*     ******* ** ***
* * * *     * *  * *
***** * *** * **** *
*       *!  *      *
********************
```

Assuming a standard array with positions starting at 0, the player would start in the upper left corner (1, 1), have to collect the item at (9, 19), then end at the ending position (20, 10).

NOTE: It's easy for a program to end up getting caught in a circle and going forever. We will guarantee the input file will have one exact solution to collect the item, then one exact solution to the exit.

Output

You should prompt the user for the input file and they should enter it in. It should continue until a file can be opened properly.

After solving the puzzle, you should have the following items on screen:

- The maze printed out
- A legend of what the symbols mean
- The number of moves
- A path to the solution shown by . (periods).

For example, given the above problem, the following could be on screen after solving the puzzle:

```
********************     Legend:
*o..*    * *   ....*     * - wall
***.** *** * **.**.*     @ - item
***......* * *..* .*     o – starting position
*   ****.*   *.***.*     ! – ending position
* * *  *.*****.  *.*
* * ** *.......***.*
* *    *.*******  @*
* ******...*   *****
*   * *  *.*** * * *
*****   *..* * * * *
*   *****.** * * * *
* *  ..... *   * * *
* * *.**** ** ** * *
*****.*     *      *
*    .******* ** ***
* * *.*.....* *  * *
*****.*.***.* **** *
*    ...*!..*      *
********************


Number of moves to solve: 85
```

Assumptions

You should assume the program:

- There will always be a wall around the outside of the maze. The player cannot escape.
- Mazes will always be solvable.
- There will always be exactly one path to the item, and exactly one path to the exit.
- There will be no loops in the maze. Recursively it gets crazy hard to solve so we won't give it to you.
  (Example: This will never happen:
  ```
  *****
  *o  *
  * * *
  *   *
  *****    )
  ```
- The input file will be valid. (It will have two integers. These will correctly tell you the size of the maze. There is no other data besides the two ints and the symbols specified earlier in this program.)
- The filename can be anything but will be one word. (Ex: maze.in, input.txt, thingy_to_solve.maze are all valid input files.)
- For this assignment, you may store the maze as a global variable. It can be solved without it though.

Ideas on Solving

Read these BEFORE starting to program:

- Your main function should ask for the input and then call a function to solve it.
- The solving function MUST be recursive and contain NO loops.
- Other parts of the program may use loops.
- It may make more sense to write it all in one file, however, you're not required to by any means.
- A "move" can consist of moving up, down, left, or right through the maze. You can not move into a wall. As you move around, you should know the player's X and Y positions in the maze. If you use an array, it is easy to check for a wall: You can't move north if X-1 is a '*'. Similar steps can be applied for the other three directions.
- If you hit a wall in three directions (ex: north, west, south all have walls), then you have to backtrack.
- When writing the recursive function, think about what changes for each move, and make those the parameters. Think about what the function is trying to solve for and make that the return value.
- Think through what each step in a maze solver would do. How do you know you're done? When you figure that out, those are your base cases.
- What steps in your algorithm can be broken down into smaller problems? These will be your recursive steps.
- Remember that for inputting the maze, spaces **are** significant, so you'll want to use the istream's .get() method rather than the stream extraction >> operator.

Testing and Other Suggestions

- **TEST EARLY, TEST OFTEN!** Write a little bit of code, then try to run it. Make sure it works. If you can get small parts of it to work correctly, then adding more parts to it means you're less likely to have crazy errors (or crazy numbers of errors) later on.
- **Feel free to modify the input file**. It will be easier to figure out what's wrong with a small puzzle than a large puzzle. You can also make up your own input files to try out. (And feel free to post those on Piazza for others to try out!)
- **Use Piazza**! We guarantee you will struggle with this, so ask your classmates. All three 201 sections have the same assignment, so you can get the thoughts and opinions of 65+ fellow students.
- **DO NOT WAIT UNTIL THE LAST MINUTE!** This program WILL take you quite a bit of time to write. We recommend getting started early. Think it through first before you start writing code. Perhaps even writing out ideas on paper first.

Finally…

DO NOT CHEAT ON THIS ASSIGNMENT! Your instructors know there are maze traversal algorithms online. We know how to search them just as well as you probably do (if not better). You need to write your own algorithm. You may discuss the algorithm with classmates and work together to think it through, but you MUST write your own code. Whether you work solo or with classmates, you may NOT use code resources online.

Anyone caught doing this will receive an automatic 0 and will be reported to the Associate Dean for an appropriate punishment.