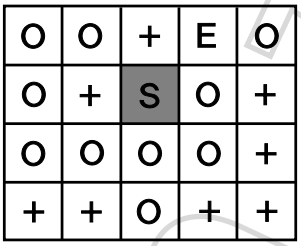
CS121 - Computer Science II
Lab Assignment #7
Fall 2011

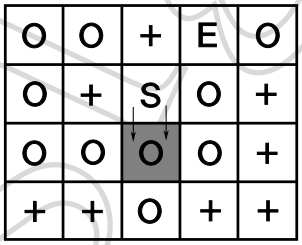
The purpose of this exercise is to use a backtracking approach to solve a maze problem. The backtracking algorithm will utilize recursion. This assignment will demonstrate the power of backtracking and recursion.

Consider a maze with four rows and five columns, as shown below:

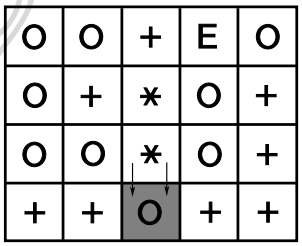


The **O**'s represent open spaces, where a person walking through the maze is able to move to. The **+**'s represent blocked positions that cannot be occupied. The **E** position is the exit from the maze, and **S** represents the starting point. From the current position, a person can only move up or down, or right or left - no diagonal moves are allowed, and the person cannot leave the maze except via the exit. The object is to try to find a path from the starting position to the exit.

As an example, consider that we are at the starting position. We can either move to the right or down; we can't move to the left or up, since those squares are blocked. Assume that we decide to move down. The shaded square shows our new position:

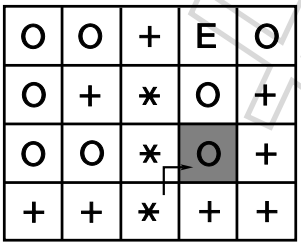


From here, we can move in any direction, although moving up will simply put us back to the starting point, something that we don't want to do. Assume that we again choose down - the situation now looks like:

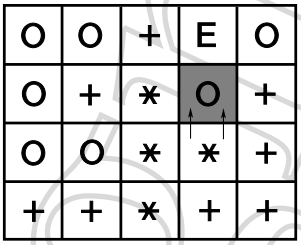


In this view, we have replaced the squares in the path we have traversed so far with *'s - these can be treated as if they are blocked squares, since we don't want to choose a place where we have already been. At this point, we are trapped - we have no moves from this point, so clearly the path we have taken does not lead to the exit.

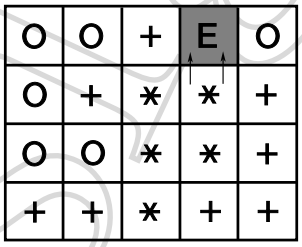
At this point we need to retrace our steps and try a different path - this is the backtrack. So we move back one step and try another one of the possible moves from this position, for example to the right:



The only possible move from here is up:



Then up again:



Tada! We have reached the exit!

This problem can be solved using a backtracking algorithm. The basic idea is this:

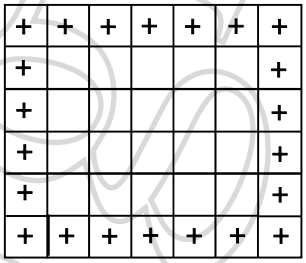
- From the starting position, choose a direction and move there.
- If the resulting square contains a + or *, we are blocked, so return and try another direction.
- If the resulting square is open, then move there and repeat the process (this is the recursion).
- If the resulting square is the exit, then we are done!

For this lab, you are to write a program that:

- Inputs the maze from a file. The first line of the input file should be the size of the maze (ie, number of rows and columns), followed by lines of characters (**O**, **+**, and **E**) that define the maze.
- Prints the maze, to verify that it was input properly.
- Asks the user the coordinates of the starting point.
- Prints out whether the exit is reachable from that starting point.
- Repeats the last two steps until a negative row value is input.

Some hints:

- Handling the border locations present some complications. There are two possible solutions:
  - Before moving to a position, check to see that the move is within the bounds of the maze. If you don't, and then move there anyway, you won't get the correct answer. Worse yet, you might get a core dump!
  - Include border rows and columns containing +'s around the maze. For example, for the example maze above, use the following border:



    This method allows you to treat the borders like any other blocked square. However, if you use this method, the borders should not be visible in any way to the user of the program.

- After each puzzle, you must "reset" the maze back to its original state. That is, convert all the *'s back to **O**'s.
- For any recursive algorithm, it is a good idea to identify the smaller or degraded problem situation, and the stopping case. What are they in this problem? (They aren't as obvious as in some other problems!)
- Note that this approach will not necessarily find the shortest path to the exit - that is a more difficult problem!

As usual, submit your program via `cscheckin`.

An example output dialog might look like:

```
$ ./maze
Enter file name of the maze to input: maze1.dat
The maze is:
 0 0 + E 0
 0 + 0 0 +
 0 0 0 0 +
 + + 0 + +

Enter the row and column of the starting point (negative row to quit): 1 1
There is a path from 1 1 to the exit

Enter the row and column of the starting point (negative row to quit): 1 5
There is a path from 1 5 to the exit

Enter the row and column of the starting point (negative row to quit): 4 1
There is no path from 4 1 to the exit

Enter the row and column of the starting point (negative row to quit): 2 2
There is no path from 2 2 to the exit

Enter the row and column of the starting point (negative row to quit): 3 3
There is a path from 3 3 to the exit

Enter the row and column of the starting point (negative row to quit): -1

Done!
```