The Thirteenth South Jersey Regional High School Programming Contest
Department of Computer Science
Rowan University
Team Problem

Problem Summary:

Your team's job is to write a program that will take a maze, find its solution, and output this solution. Your program will do this for three different mazes.

Problem Description:

A maze is a two-dimensional structure made up of walls and spaces. A period ('.') is used to represent an open space in the maze and a '#' character is used to represent a wall in the maze. Your program has to solve three such test mazes, which are available on the web at http://www.rowan.edu/ baliga/contest/mazes.htm . In addition, after you submit your program, the judges will test your program by changing one of the test mazes in your code to a maze that you will not get to see ahead of time.

Here's an example maze of size 5:

```
#####
..#.#
#.#.#
#...#
###.#
```

We will call the top row, 'row zero,' and the left column, 'column zero'. A maze always has exactly two entrances. For example, the maze above has one entrance in row 1, column 0 and the other in row 4, column 3.

To solve a maze, you should enter the maze at any one of the entrances and find a way to the other. The above maze is the first test maze for your program and is provided as a single string, which comprises the rows concatenated together to give:

```
#####..#.##.#.##...####.#
```

The output for the above test maze should be:

```
#####
++#.#
#+#.#
#+++#
###+#
```

Note that the path from the entrance to the exit is marked with the character plus ('+'). Also note that your output should be two-dimensional (i.e. on multiple rows) and not all on one row like the maze you start with. All of the test mazes are square and the size of the largest maze is 13.

Program Requirements:

Your program will not take any input from the user. It will output the solutions to the three mazes specified on the above web page. This means that you will have to copy the maze strings provided on the web page into your code. When we test your code, we will remove one of the mazes that you use, and replace it with one we won't show you in advance, just to make sure that your code works for any maze up to size 13. Note that your output should be two-dimensional (i.e. on multiple rows) and not all on one row like the test mazes.

Hint: How to store the maze:

We suggest that you store the maze in a character array. You could use a one-dimensional array of characters to do this, or even a two-dimensional array of characters if you are familiar with multi-dimensional arrays.

If you want to use a one-dimensional array:

Suppose that your maze is of height and width 5.

> If your arrays start at index 0, we suggest that you store the character at row r and column c in this maze at subscript ((5*r)+c). So, for example, the top left corner of the maze (row zero, column zero) will be stored at subscript zero.
>
> If your arrays start at index 1, we suggest that you store the character at row r and column c in this maze at subscript ((5*r)+c+1). So, for example, the top left corner of the maze (row zero, column zero) will be stored at subscript one.

If you want to use a two-dimensional array:

> If your array indices start at 0, we suggest that you store the character at row r and column c in this maze at subscripts r and c. So, for example, the top left corner of the maze (row zero, column zero) will be stored at subscript *zero/zero*.
>
> If your array indices start at 1, we suggest that you store the character at row r and column c in this maze at subscripts r+1 and c+1. So, for example, the top left corner of the maze (row zero, column zero) will be stored at subscript *one/one*.

Hint: How to Split up the Problem

In order to solve this problem, we recommend the following division of the responsibilities amongst the team members. What we offer below are broad guidelines, not specific details. We do not claim that our guidelines are exhaustive; we believe they are quite useful. Depending on how you interpret our guidelines, you may need to create one or more extra functions to complete the problem. *If you feel that the problem can be solved in a different manner, you should feel free to implement your approach.*

In what follows, we use the word 'function' to mean functions/procedures/subroutines and the like. We recommend that the functions that you write be as general as possible. Remember that you need to solve three mazes, not just one! When deciding on the parameters accepted by a function, you must consider if the maze, its size etc. should be the parameters to this function. The

team members must discuss the functions that each person is responsible for and agree on the input and output parameters and (if applicable) return value. Once this agreement has been reached, each person can proceed to do his/her individual part.

Summary of our recommendations:

Here is a brief summary of how we suggest you split up the problem. Full details for each person are provided below.

> Person 1 provides functions for encoding the maze, printing the solution, and finding one of the entrances to the maze.
>
> Person 2 provides the functions for moving around in the maze (without moving through walls!), and marking the path as the maze is solved by person 3's functions. Person 2 also sometimes has to unmark a position that person 3's maze solving functions tried out, but hit a dead end.
>
> Person 3 implements the maze solving function using the functions provided by Persons 1 and 2. This function is called to solve each of the three different mazes.

Concerning the work of Person 1:

We recommend that person 1 be responsible for the following tasks:

1. To retrieve the three mazes from the web and store them in three different character arrays. Please note that you must be careful to cut and paste the entire maze from the Internet browser's window and not miss any characters at the end. It might be worth your time to count the number of characters that are pasted into your program. Note that since the mazes are square, the number of characters in the maze is the square of the maze size. So, a maze of size 13 will have 13 * 13 = 169 characters.

2. To write a function that finds the starting point (a row and a column) in a maze and an orientation into the maze. In the little maze above, if the starting point at the left row 1 is discovered, the initial orientation of the person is facing East. It may be worthwhile to define the four directions as constants, for example, North=0, East=1, South=2, and West=3. Note that implementing this function may possibly involve using the functions provided by Person 2.

3. To write a function that prints a maze.

Concerning the work of Person 2:

Person 2 provides the functions that allow Persons 1 and 3 to view the maze as a two-dimensional entity and to inspect and change the values in the maze. We recommend that Person 2 be responsible for the following tasks.

1. To write a function that computes whether it is possible to move from the current maze position in a given direction.

2. To write a function that retrieves the value of a character at a given maze position.

3. To write a function that sets the value of a given maze position to a given character.

4. To write the main program that uses the maze solver implemented by Person 3 on the three mazes created by Person 1 and prints their corresponding solutions using Person 1's print function.

When the solving function implemented by Person 3 tries to solve the maze, it needs to be able to inspect the values of the maze at different positions. For instance, if you move to a maze cell that is currently empty (i.e., it has a period character – you can check it using function number 2 above), you place a plus in the cell you just left to mark the path. Similarly, if you move to a maze cell that has a plus character in it, you put a period in the cell that you just left (using function number 3 above), since your are obviously backtracking along an old path.

Concerning the work of Person 3:

We recommend that Person 3 is responsible for the following task:

1. Implement a function that solves one given maze.

Person 1 will have implemented the first step in solving the maze, i.e., finding an opening in an outer wall. Call this the starting point. Then use the following maze solving algorithm. This algorithm is described as if a person is walking through the maze. At the starting point, touch the maze wall with your left hand, and proceed to follow the path which results from continually keeping your left hand touching the maze wall. This can be implemented as follows: move left if possible (note that left is not necessarily West. If you are currently facing South, then to move to the left you would go East). If you can not go left, move forward if possible; if not, move right if possible; if not, do a 180-degree turn and move forward to the maze cell that you just came from. *Note that every move is either horizontal or vertical; diagonal moves are not allowed.*

Person 3 may need to maintain an orientation variable for the 'person' moving around in the maze; they should use the direction encoding suggested above; i.e., if the person is facing South, the orientation variable should have the value 2. Whatever direction the 'person' is currently facing, if he/she needs to turn left, he/she can do so by subtracting one from the current orientation (Going left when facing North (value 0) gives -1, which should be converted to 3 (i.e., the direction West)). If the 'person' is to go forward, their orientation doesn't change, and they move in the direction specified by their orientation. If the 'person' is to go to the right, they have to face right by adding one to their orientation (modulo 4). A 180-degree turn is implemented by adding 2 (modulo 4) to the current orientation.

As the person walks along the maze, he/she marks his/her path by replacing the period character in the cells on the path with the plus character. Note that in the process of finding a solution, you may need to backtrack from 'dead ends'. When you backtrack, you must change the maze values of the corresponding cell from a plus character to a period character (its original value). The display of the solution *should not* include any such backtracking; it shoudl show the most direct route from maze entrance to exit, marking each square in this path with a '+'. Note that you can assume that all sample mazes supplied to you do not have 'loops' inside them; hence this algorithm will always eventually find a solution. Note that the inspecting and changing of the values of different maze cells can be accomplished by using the functions provided by Person 2. Note that if you arrive at a cell that has a period in it and that is on the 'edge' of the maze, then you have solved the maze.