Cosc 1p03 Assignment 4

(Due date November 28th 16:00 est, Late date Dec 1st 16:00 est)

Read the entire assignment from start to finish before starting to work on it. Read it multiple times if you need to.

Background

It's Valentine's Day and you need to deliver a love letter to your crush who is somewhere in the hallways of the school. Unfortunately, the hallways are densely crowded, and you don't know where they could be.

Part A

Picture a single file hallway. You are at one end and your crush may or may not be somewhere in the line of people ahead of you. To deliver your letter, you would need to pass it along to every person in the hallway until it reaches your crush. But if your crush is not in this hallway, you would need to make sure that the letter returns to you.

On the envelope, you would need a simple set of instructions that every person in the hallway could follow when they got the letter. The instructions might say something like this:
If you are the crush, then keep the letter.
If you are at a dead end, then return the letter to whoever gave it to you.
If you are not the crush, then pass the letter to the next person in the hallway and tell them to read these instructions.
If the letter has been returned to you after you have already performed the last step, then return the letter to whoever gave it to you.

If you think through these steps with a small example, the letter will always either make it to your crush or get returned to you if they are not in the hallway. This is also a recursive set of instructions. Steps 1 and 2 are both trivial cases (or base cases). Step 3 has the recursive step ("tell them to read these instructions"). Step 4 handles backtracking.

You are given a hallway as an ASCIIDatafile (hallway.txt). All walls will be represented as '#'. The size of the hallway (rows, col) is given as an integer pair (eg. 3, 11). Below is a sample.

```
3    11
###########
#         #
###########
```

You will have to picture the blank spaces as students in the hallway. When you run your letter sending code, a path should be traced from the starting location, S, to your crush, marked with an E. The resulting output would look like this:

```
###########
#S>>>>>>>E#
###########
```

The starting location and ending location are not included in the ASCIIDatafile so be sure to print them in your code. For Part A, the start and end locations should be hard coded to always be at the ends of the hallway.

Your output should also be different if your crush is not in the hallway. Like this:

```
###########
#S........#
###########
```

To test this scenario, hard code the end of the hallway to not have an E.

The Algorithm

This letter passing must be done recursively. You will need to create a method findPath(int x, int y) which receives the location in the hallway as a coordinate pair (x,y). You can modify the parameters in the header to help with the path tracing. The method will include the recursive algorithm that every student in the hallway will follow. This include the two base cases (crush found, dead end) and the recursive step (pass it along to the next student).

If the crush has received the letter, you will need some way to signal to everyone else in the hallway that the letter has been delivered (maybe with a global variable?).
If a student passes the letter along, then they will need to mark who they passed the letter to with a '>'.
If a student gets the letter returned back to them then they should mark their position with a '.' indicating that the path was tried, and there is no crush at the end of it.

Once, Part A is successful, move on to Part B.

Part B

This part extends Part A by having a school with many hallways, some of them leading to dead ends. Use the existing code from Part A and it expand it for Part B.

You are given an ASCIIDatafile to mark the hallways of the school. All walls of the school are represented as '#'. The size of the maze (rows, col) is given as an integer pair i.e. (8,11). Below is a sample school.

```
8    11
###########
#         #
#### ######
#    #    #
#    #    #
# ###### ##
#         #
###########
```

Once, you run your letter delivery application a path is traced from the starting location S (in this case (1,1)) to your crush marked with an E, using ASCII characters. The starting and ending locations will be randomly generated.

```
###########
#S>>v     #
####v######
# v<v#  >E#
#v<^<#  ^.#
#v######^##
#>>>>>>>^ #
###########
```

With a layout like this, the algorithm used for a single hallway needs to be modified to account for students who could pass the letter down branching hallways. For students at a fork, they should try passing the letter down one hallway and if it gets returned back to them then they should try passing it down the other hallway and so on until either the letter gets delivered successfully or there are no more hallways to try, and they must then return the letter back to the person who gave it to them.

The new set of instructions on the envelope may look something like this:

If you are the crush, then keep the letter.
If you are at a dead end, then return the letter to whoever gave it to you.
If you are not the crush, then:
    Pass the letter down hallway 1 and tell them to read these instructions
If the letter has been returned to you after trying the last step then:
    Pass the letter down hallway 2 and tell them to read these instructions
If the letter has been returned to you after trying the last step then:
    Pass the letter down hallway 3 and tell them to read these instructions
If the letter has been returned to you after trying the last step then:
    Pass the letter down hallway 4 and tell them to read these instructions
If the letter has been returned to you after trying the last step then return the letter to whoever gave it to you.

For this scenario, the students can only move in four directions.

The Algorithm

The astute among you may recognize that this a maze walk. The premise is to try a direction and if successful keep trying that direction until the goal condition is met or a wall is encountered. This is repeated for all four directions. If a location has been exhausted, i.e. all four directions, then it is marked with a flag so that no further attempts are made at that location. This prevents infinite recursive calls. Consider the case where a location can be entered from multiple directions. The solution exemplified above shows one such case, and is dependent on order of the recursive calls.

You should extend the findPath(int x, int y) method from Part A with the following changes. Instead of having one recursive call for a single hallway, there will need to be four recursive calls, one for each direction. When the student passes the letter to another student, they will need to indicate which direction they passed the letter with ASCII ^v<> characters. Once all four directions have been exhausted and the letter has not been delivered, the student should mark their position with a '.' indicating that they are not part of the successful path to the crush.

Students who have already received the letter should be marked with something indicating that other students should not pass the letter back to them. You would not want a scenario where Student A passes the letter to Student B, B to C, C to D, and then D back to A. Once A gets the letter back and it was not from the student they gave it to (B), they will think this is a new letter and start the instructions again from the beginning resulting in an infinite recursion. Instead, consider that blank spaces in the school indicate students who have not yet received a letter and if there is an ASCII character in the maze that is not 'E' (ie. ^v<>.) then that is a student who has already received and passed the letter.

You (S) and your crush (E) are to be randomly placed in the maze. You may generate a random row and column within the bounds of the maze. If the location is not a wall accept the location and place E and then repeat and place S.

Output should be the starting location **S**; maze with both S and E indicated, followed by where E was found. Include the path through the maze using ASCII ^v<> characters. After the maze has been printed output the row and column where your crush has been found. E.g. of output expected

```
Starting location S is (x,y)

###########
#S>>v     #
####v######
# v<v#  >E#
#v<^<#  ^.#
#v######^##
#>>>>>>>^ #
###########

Crush found at location (x,y)
```

Little Hints

1) Start by making a method to read in an ASCIIDataFile that puts the school layout into a 2D char array and a separate method that prints this array.
2) When reading characters from the ASCIIDataFile use the `readC()` method. This method reads exactly one ASCII character.
3) At times you may have extra spaces at the end of some lines of text or simply wish to ignore the rest of the characters on the current line since hidden characters like line feeds are still characters but not part of the matrix. The method `readLine()` will read all characters from the current location to the end of line including the line feed. Allowing the next read to start at the beginning of the next line.
4) Have a method to set up the school hallways with the start and end location. For part A, hard code the start and end. For Part B, modify the same code to randomize these locations.
5) Start with Part A since it will make Part B much easier.
6) Included are 3 files hallway.txt, mz1.txt, and mz2.txt. These can be used as input maze maps.
7) Write output to an ASCIIDisplayer during the run. You should also output the solution as an ASCIIOutputFile. Completed solutions should look similar to the sample given in this text. When viewing the file use a mono space font like courier so that the maze appears properly.

8) Depending on the order in which you explore unvisited locations in the map, the final solution may vary slightly indicating different paths or tried locations.
9) The algorithm will find a solution, not necessarily the optimal solution.
10) As with most recursive algorithms the solution will be very few lines of code. If findPath seems to be getting very complicated and big, then chances are you are doing it wrong.
11) Note: There are multiple base cases, **E**, a student who has already received the letter, and a wall **#**.
12) Make sure you have read and understood the instructions of the assignment before beginning to code.

Submission

Only Part B of the assignment is necessary for the submission. In addition to your code, include an output of the large maze and a screenshot of a sample execution. Be sure to put proper commenting and ensure the layout and indentation of the code is neat.

Submit a zip file of your assignment via Sakai.
Rar submissions will be given a grade of 0.
Non recursive solutions will be given a grade of 0.
Late submissions will receive a penalty of 25%.