CS201 – Data Structures and Discrete Math I Fall 2014

Programming Project 2

Due: Tuesday, 10/21/14 at 11:59 pm

Maze Solving

For this assignment, you are to write a C program that will find a path in a given maze. The maze will consist of a 20x20 matrix. The path is to start at position 1,1 (upper left corner) of the maze and travel to some randomly determined ending position. The path can move to any open neighboring position in either a horizontal or vertical direction (but NOT diagonally).

The input for the maze will be given by a sequence of two integer values per line (separated by space characters). Each pair of values will be the coordinates of a position in the maze that is blocked (i.e. cannot be used in the path). The last pair of input values will be **1 1**. Since the start position cannot be blocked (unless you want a very boring maze), it makes a good terminal value for the input. The input is to be read from standard input, so it can be entered at runtime or redirected from an input file. Before the input is read in, print a message such as ***Reading blocked maze positions, (end input with value "1 1")...***.

If an input pair contains a value outside of the range of 1 to 20, print out an error message and ignore the input pair. After the input is finished being read in print out the maze in some readable form. Here is a sample data file. Note, there are three invalid position in the input.

After the maze has been printed you are to find the path to five different ending positions. These ending positions must be randomly determined by your program (use the **rand()** function) and must not be blocked positions. Each path must be found twice, once using a stack and once using a queue. The path found by using a stack will (in most cases) be different than the path found by the queue. It is possible that that a path could not be found (i.e. part of the maze is completely blocked off from the start position). In this case print a message that no path could be found from the starting position to the ending position.

To use the stack, you would first push the start position onto the stack and then loop until either the stack becomes empty (no path to the end) or you encounter the end position. During each cycle of the loop, you find a neighboring position to the position on the top of the stack that has not been visited. If such a position exists you push this neighboring position onto the stack. If such a position does not exist you pop the top position from the stack. Once the end position is found, the path from start to end are the positions currently on the stack. This algorithm is called a **Depth First Search**.

To use the queue, you would first add the start position to the queue and then loop until either the queue becomes empty (no path to the end) or you encounter the end position. During each loop cycle, you are to remove the first position from the queue. Then add all neighboring positions that have not been visited to the queue. Once the end position has been found, the path from the start to the end is found by backtracking from the end position to the beginning position. To do this, at each visited position you must remember which position added it to the queue. This algorithm is called a **Breadth First Search**.

Once a position is pushed onto the stack or added to the queue, that position has been "visited". A visited position cannot get "unvisited" while finding the current path.

Program Output Summary

1. The initial message. Such as:
   Reading blocked maze positions, (end input with value "1 1")...
2. Messages for any invalid input values.
3. The maze showing the blocked positions
4. For each of the 5 end positions,
   a. The end position
   b. The path found using the stack or message stating end position is not reachable.
   c. The path found using the queue or message stating end position is not reachable.

   The path can be shown by listing a sequence of positions in the maze (starting from position 1,1) or by showing the maze in a 20x20 grid which clearly showing the positions on the path and the blocked positions (a border around the maze is a good idea).

The following is an example of a drawn 10x20 maze (a 20x20 maze just took up to much space so it was 'reduced'). Asterisks are used to show blocked positions and the outside walls/border. Dots are used to show the open spaces in the maze. An 's' is used to show the starting position of 1,1.

```
**********************
*s........*..........*
*........*...*****...*
*..*....*............*
*.*....*.....*********
**....*........*.....*
*....*.........*.....*
*...*.....*....*.....*
*..*......*....*******
*......*..*..*.*.....*
*......*.....*...*...*
**********************
```

General Comments

The code for the queue and stack must be written by yourself and must use linked links with pointers. You will received zero points for this assignment if you use a library queue or stack (i.e. you don't write it yourself).

Your program must be written in good programming style. This includes (but is not limited to) :

- meaningful identifier names,
- a file header comment at the beginning of each source code file,
- a function header comment at the beginning of the function,
- proper use of blank lines,
- uniform indentation to aide in the reading of your code,
- explanatory "value-added" in-line comments,

Program Submission

You are to submit the programs for this lab via the Assignments Page in Blackboard.

To help the TA, name your file with your net-id and the assignment name, like:

- ptroy1Proj2.c