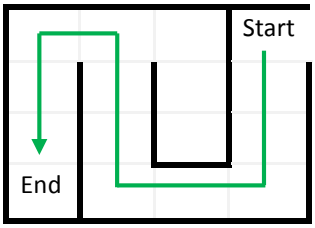
COP 3530 – Data Structures and Algorithms I

Project 5

Due: Friday, April 22 11:59 P.M. CST

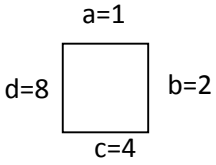
Objective:

This assignment focuses on backtracking that solves search problems by recursively evaluating alternative options in the search of a solution. The objective of this project is to find a path in a maze given a start and end location. Below is a 4x4 maze (light gray indicates cell boundaries, black indicates cell walls) with a green path to the end location from the starting location.



Problem Description:

Write a C program that implements an algorithm to find a path through a maze given a start and an end location. The maze is described by an NxM (rows by columns) matrix with each matrix cell describing the walls in the maze. Each cell can have 4 walls. The walls are encoded by numbers as specified below:



A single cell with 4 walls, labeled a, b, c, d and corresponding values for each

The matrix cell stores the combined value for walls that are present in a cell. For example, if the walls that are present in a single cell include a and d, meaning that b and c are open, then the value in the matrix for the corresponding cell is 1 + 8 = 9.

Applying the encoding method to the above maze generates the following matrix:

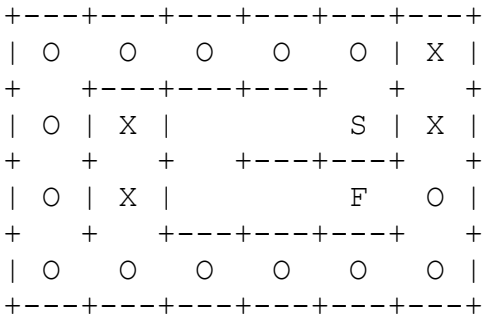
9	1	3	9
10	10	10	10
10	10	14	10
14	12	5	6

Assuming that the coordinate for the upper left corner of the matrix is (0,0) and the lower right corner is (3,3) then for the location (0,1) with value 1, the option to proceed through the maze is either going left to (0,0), right to (0, 2), or down to (1,1).

Your program must also be able to render the matrix into a display that illustrates the maze. For example, the following describes a 4x6 matrix of a maze with a possible rendering of the maze using ASCII characters.

- X indicates the location has been visited but led to an unsuccessful solution,
- O indicates the cell has been visited and lead to a successful solution,
- F indicating the end cell, and
- S indicates the starting cell.

9	5	5	5	3	11
10	11	9	5	6	10
10	10	12	5	5	2
12	4	5	5	5	6



The focus of this project is on properly handling the backtracking algorithm. For this reason, much of the surrounding code has been written for you. You will find the beginning code for this project in the zip file maze_code.zip online. The tasks remaining for completion are as follows:

1. Properly allocating and initializing both the matrix for storing the encodings as well as the matrix for storing the paths. You will need to read the encodings from a text input file. The matrix for storing the paths is called visits and should be initialized with the coordinates for the starting position and the ending position. All other cells in the matrix should be initialized to a blank space. In maze.h, you will find a list of defines that should be used when setting up your visits matrix (e.g., instead of setting a position equal to a blank space, set it equal to UNVISITED).
2. Freeing the memory for the maze when completed.
3. Implementing the backtracking algorithm for stepping through the maze. This function is labeled step and should display the maze after every step (in addition to when the final path is found). You will also want to pause after displaying the maze, you can do this by calling getchar() which forces the program to wait for a key press before continuing. While updating the board, remember to use the defined values from maze.h.
4. The name of the input text file must be entered by the user on the command line when the program is executed.
5. If the maze has no solution, the program must display an appropriate message.
6. You must create a makefile to compile your solution.

Input File Format

The specification of the maze and the start and end location must be read from a file. The format of the input file is shown below. Line numbers are given for illustration purposes only but are not included in the actual text file.

1	N M
2	x y
3	x y
4	$C_{0,0}$ $C_{0,1}$ $C_{0,2}$ …. $C_{0,M-1}$
5	$C_{1,0}$ $C_{1,1}$ $C_{1,2}$ …. $C_{1,M-1}$
…	…
N+3	$C_{N-1,0}$ $C_{N-1,1}$ $C_{N-1,2}$ …. $C_{N-1,M-1}$

In line 1, N is the number of rows, M the number of columns in maze matrix. Coordinates x and y specify the start location in line 2 and the end location in line 3. C_{ij} specifies the value for the walls in location i,j in the matrix.

The maze illustrated on page 1 would be represented in the file as follows:

```
4 4
3 0
0 3
9 1 3 9
10 10 10 10
10 10 14 10
14 12 5 6
```

You will also find an additional test file in the directory with the source code.

Running the Program

Example of how the program should be compiled and ran:

…$ make

…$ ./maze testMaze

```
+---+---+---+---+---+---+
|                       |
+   +---+---+   +   +   +
|   | S |       |   |   |
+   +   +   +---+---+   +
|   |           |       |
+   +   +---+---+---+   +
|   |   |       |       |
+   +   +---+   +---+---+
|   |   |       | F     |
+   +   +---+   +---+   +
|                       |
+---+---+---+---+---+---+
```

You should then be able to press any key to start stepping through the maze.

Steps for Completion:

There will be no additional tasks for this project.

1. Properly allocating and initializing both the matrix for storing the encodings as well as the matrix for storing the paths. (20 points)
2. Freeing the memory for the maze when completed. (5 points)
3. Implementing the backtracking algorithm for stepping through the maze. (50 points)
4. Displaying an appropriate message if the maze has no solution. (15 points)
5. Creating a makefile to compile your solution. (10 points)

In addition, **please add your name as a comment to the top of each file.**

Bonus Tasks:

The following are bonus tasks that everyone can complete:

- Displaying all paths through the maze (15 points)
- Displaying the steps to traverse the maze along with the path (i.e, for the maze displayed in this description, the steps would be to go Up Left Left Left Left Down Down Down Right Right Right Right Right Up Left) (15 points)

The next two tasks are bonus tasks that those who did not attend Codefest can complete (those who participated in Codefest have already earned these bonus points):

- Displaying the shortest path of all paths. (25 points)
- The display of the maze is a little simplistic. One simple way to update this is to use color. Look up how to use color codes. Keep the borders as the normal color, change the good paths to be colored green, the bad paths should be colored red, the start should be blue, and the finish should be colored yellow. Make sure to reset the colors back after finished. (25 points)

You should clearly distinguish the bonus steps from the rest of the program. You can do this by adding options to the command line after the name of the input file.

Bonus Options:

- No Option Specified – Run your program as normal
- All – Tells your program to print all the paths through the maze
- WSteps – Tells your program to run as normal but to include the steps as you go
- Shortest – Tells your program to print the shortest path

If you complete your program with the additional colors, then you should print with the new colors for all options.

An example run of program:

…$ ./maze testMaze All

Submission Instructions:

Make sure that before submitting your program that you have tested it on the ssh server (*ssh.cs.uwf.edu*). There is information for how to do this on eLearning.

For this project, submit a zip file containing all files required to run your project, these files will be named maze.c, maze.h, maze_client.c, testMaze, and makefile. If you used the scanner, you must also include scanner.c and scanner.h. Submit your project using the eLearning dropbox. Remember that all students who turn in their projects by the due date that receive a B or better will receive an added 0.5% on their final overall average.