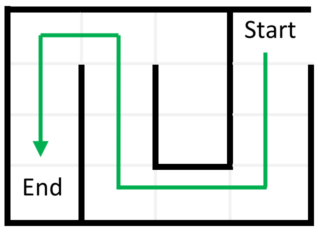
PROJECT 2: BACKTRACKING

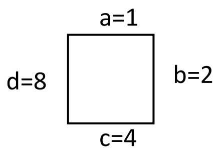
OVERVIEW

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.



THE PROGRAM

Write a C program `backtracking` 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 wall.

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

```
+---+---+---+---+---+---+
| O   O   O   O   O | X |
+   +---+---+---+   +   +
| O | X |           S | X |
+   +   +   +---+---+   +
| O | X |           F   O |
+   +   +---+---+---+   +
| O   O   O   O   O   O |
+---+---+---+---+---+---+
```

Additional specifications for your program are:

1. You must include a recursive function called `step()` which accomplishes the task of taking "one step" in the maze. Otherwise, break down your code into multiple functions as you see fit.
2. Your code must handle any size of maze up to and including 10x10.
3. Your code must display the board each time it is updated. Do not wait until a solution is found to display the board. I want to see the actual "stepping" and "backtracking" take place.
4. Your code must pause each time a cell on the board is updated so that it is easily visible on the monitor. Use the `getc()` function to make the pause happen.
5. Use S, F, O, and X in your display function as indicated above to describe the start and end location, successful and unsuccessful paths.
6. The name of the input text file must be entered by the user on the command line when the program is executed.
7. If the maze has no solution, the program must display an appropriate message.
8. Include appropriate documentation at the beginning of the program and before each function.
9. Do NOT use any global variables.

SEARCH OPTIONS

The algorithm to find a path from a start to an end location in the maze can be easily extended to provide alternative solutions. Use `-all` and `-short` as input options on the command line to give the user the following two exclusive options:

1. With the `-all` option entered on the command line, display all solutions to the maze in a single step.
2. With the `-short` option entered on the command line display the shortest solution to the maze in a single step.

For both options, you will need additional data structures to store and evaluate the paths that the backtracking algorithm identifies.

File Format for Maze Descriptions

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     c0,0 c0,1 c0,2 …. c0,M-1
5     c1,0 c1,1 c1,2 …. c1,M-1
…     …
N+3   cN-1,0 cN-1,1 cN-1,2 …. cN-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
```

Project Outcomes

- To use backtracking for solving a search problem.
- To use matrices for representing spaces and search paths.
- To use a Makefile for compiling source code into an executable.

Implementation Suggestions

Your program must be started as follows:

```
backtracking maze.txt [-all | -short]
```

The parameter `maze.txt` indicates the name of the text file that contains the maze description, `-all` and `-short` indicate optional parameters for handling the special cases of displaying *all* solutions or displaying the *shortest* solution. The notation `[ ... ]` indicates that the values may or may not be present and `|` indicates either or but not both.

Use a second matrix of characters to represent cells not-yet-visited (using ‘ ’), visited (using ‘X’ or ‘O’ as described above), and the start (‘S’) and end (‘F’) cell for the search.

Some sample code to get started on this project is available in *eLearning*. I would strongly encourage you to review and use this code to develop your solution.

Demonstration & Deliverables

Submit your complete solution as a single zip file (only standard zip files will be accepted) containing A) source code, B) a single Makefile to compile the program, and C) a README file (if applicable) to the corresponding Dropbox in *eLearning*. In addition, submit the source code and the Makefile as individual files to your own directory on the shared drive under `/shared/dropbox/treichherzer/cop5990`. Be sure to follow the Project Submission Instructions and Code Documentation and Structuring Guidelines posted by your instructor in *eLearning* under Course Materials to avoid unnecessary point deductions on your project grade.

Finally, be sure that your code is well organized. Functions should be refactored so that they can be easily reviewed and maintained. This means that any large function (functions with a body of about 20 or more lines of code) must be broken down into several smaller functions. You should refactor code such as the body of a loop or an if-else-statement when it exceeds a large number of lines of code or when the statement contains additional nested if-else or loop statements!

Interesting Fact

In mathematics, the dimensions of a matrix are specified in term of the number of rows and columns written as m x n where m is the number of rows and n is the number of columns. However, for most graphic displays, the size of the display is given as width x height where the width is the number of columns and height is the number of rows in the corresponding pixel matrix.

Due Date

The project is due on the date shown in the syllabus and *eLearning* calendar. Late submissions will not be accepted and the grader will not accept any emailed solutions. The syllabus contains details in regards to late submissions.

Important Notes:

1. Projects will be graded on whether they correctly solve the problem and whether they adhere to good programming practices.
2. Projects must be submitted by the time specified on the due date. Projects submitted after that time will get a grade of zero.
3. Please review UWF's academic conduct policy that was described in the syllabus. Note that viewing another student's solution, whether in whole or in part, is considered academic dishonesty. Also note that submitting code obtained through the Internet or other sources, whether in whole or in part, is considered academic dishonesty. All programs submitted will be reviewed for evidence of academic dishonesty, and all violations will be handled accordingly.

GRADING

This project is worth 100 points in total. The rubric used by the grader is included below. Keep in mind that there will be a substantial deduction if your code does not compile.

Project Submission	**Perfect**	**Deficient**		
eLearning	5 points individual files have been uploaded	0 points files are missing		
shared drive	5 points individual files have been uploaded	0 points files are missing		
Compilation	**Perfect**	**Good**	**Attempted**	**Deficient**
Makefile	5 points make file works; includes clean rule	3 points missing clean rule	2 points missing rules; doesn't compile project	0 points make file is missing
compilation	10 points no errors	7 points some warnings	3 points some errors	0 points many errors
Documentation & Program Structure	**Perfect**	**Good**	**Attempted**	**Deficient**
documentation & program structure	5 points follows documentation and code structure guidelines	3 points follows mostly documentation and code structure guidelines; minor deviations	2 points some documentation and/or code structure lacks consistency	0 points missing or insufficient documentation and/or code structure is poor; review sample code and guidelines
Backtracking	**Perfect**	**Good**	**Attempted**	**Deficient**
reads maze description from file	10 points correct, completed	7 points minor errors	3 points incomplete	0 points missing
implements search function using matrix representation for maze as described in the project description	15 points correct, completed	11 points minor errors	4 points incomplete	0 points missing
displays maze step-by-step until a solution is found	15 points correct, completed	11 points minor errors	4 points incomplete	0 points missing
handles the –all option for finding all solutions	15 points correct, completed	11 points minor errors	4 points incomplete	0 points missing
handles the -short option for finding the shortest path	15 points correct, completed	11 points minor errors	4 points incomplete	0 points missing

I will evaluate your solution as attempted or insufficient if your code does not compile. This means, if you submit your solution according to my instructions, document and structure your code properly, provide a makefile but the submitted code does not compile or crashes immediately you can expect at most 49 out of 100 points. So be sure your code compiles and executes.