Project 2

Maze Runner

Due: 8th November 2017

Overview

In this project you will implement the logic of a maze solver in C. In the course of this project, you will get more practice with all of the fundamental programmatic structures of C as well as experience using external library code. Finally, you will need to create your first '.h' file to provide an interface to your code.

Submission Instructions

Submit only the .c and .h files *that you created* along with your Makefile, but do not zip them!

Additionally, be sure your makefile produces an executable called "maze_runner" so that my test script can run it.

Technical Description and Instructions

In this project, you will not write a full program. I have already written most of a program to solve randomly generated mazes. The parts that I have written generate the maze and provide functions through which the maze can be interacted with. Additionally, the parts that I have written rely on calling some functions that will solve the maze. Implementing those functions is your job!

Your Two Files

You must create two files for this project: "runner.c" and "runner.h". Runner ".h" should expose two functions to the rest of the program called `runner_solve()` and `runner_init()`. The first function will utilize the maze library functions I've provided to solve a maze[1]. The second function simply performs any setup necessary for the `runner_solve()` function to run. These two functions should be implemented in the "runner.c" file.

You may create as many helper functions as necessary to support your `runner_solve()` function[2]. Likewise, you may use any reasonable algorithm to actually solve the maze. Your runner must leave behind a trail of 'breadcrumbs' as it moves through the maze (see Program Output for details).

Program Output

Program output is actually handled for you on this one! The maze_runner.c file I've provided for you will call `print_maze()` for you to show both the unsolved maze and the maze after your runner has finished.

However, you must accurately show the path that was taken by your algorithm to solve the maze. This is done by using the `maze_set_char()` function as your 'runner' moves through the maze. When the runner crosses an empty square, it should leave a '.' behind. When it crosses a '.', it should leave an 'o' behind.

[1] Descriptions of these maze library functions are in the comments of the mazelib.h file provided.

[2] My implementation has around five.

When an 'o' is crossed, an 'O'[3] should result. Finally, when an 'O' is crossed, an '@' should be left in its place.

Things to Remember and Helpful Hints:

Make sure you read all of the comments in the "mazelib.h" header file! These comments will tell you all you need to know about interacting with the maze.

Don't forget to include "-std=c11"[4] in your `CFLAGS` variable in your Makefile. This makes all the latest features of C available to you.

A good way to start this project would simply be to create your .h and .c files and fill in a skeleton for the necessary functions. They can just do nothing and return, but this will let you test your Makefile by compiling the project. After that, you can run the program to make sure everything is working and then begin to implement your solver logic.

Grading Specification

For your submission to receive a grade of 'pass', it must fulfill the following criteria:

- **It must be submitted correctly**.
- I must be able to compile your program simply by invoking "make" in a directory containing your code and Makefile.
- The program must compile with no warnings or errors.
- The program should run with no errors and output a legally solved version of the generated maze.

[3] Capital 'o', not a zero!
[4] If your compiler doesn't support the c11 standard, use c99 instead.