PORTLAND STATE UNIVERSITY
CS201: COMPUTER SYSTEMS PROGRAMMING (SECTION 5)
FALL 2017

HOMEWORK 1
DUE: OCTOBER 9, 11:00 AM

1. Goals and Overview

1) In this programming assignment we will exercise and further develop practical knowledge of the C Programming Language.
2) You will use the standard C library to manipulate files, strings and arrays
3) You will properly allocate and deallocate dynamic memory
4) You will get familiar accessing multidimensional arrays and pointers

2. Development Setup

For this programming assignment you can work individually or in groups of 2 students. You only need one submission per group. You will work in the CS Linux Lab (linuxlab.cs.pdx.edu) located in FAB 88-09 and 88-10. If you don't already have an account, go to http://www.cat.pdx.edu/students.html for instructions.

For this Homework you must use ANSI C and Make. Please refrain from using any language extensions or 3rd party libraries that are not part of the standard ANSI C. Do not use any C++ specific language constructs. We will use the GNU C Compiler (gcc) already installed in the development machines in the lab. Do not use g++!

Grading will be done in the setup mentioned above so please make sure that your code works under these conditions.

3. Introduction

In this Homework we will design and develop a maze solving application. For this problem we will use the Wall Follower algorithm. This algorithm assumes that all the walls of the maze are connected together or to the maze's outer boundary, therefore, following the wall at some point will lead you to the exit.

The Wall Follower algorithm can be implemented using a right hand policy or a left hand policy. That means either that the algorithm follows the wall on the right hand size or the wall on the left hand side. You are free to implement either one. In fact the changes between one and the other are trivial. In the rest of this document we will assume a right hand policy.

4. Problem Description

4.1 Maze Representation

A maze will be represented as a 2-dimensional array of characters of any dimension. Your code should use dynamic memory to properly handle this requirement. The maze will be provided in a text file also specifying the maze entry and exit coordinates within the array.

Your first task will be to open the file specified by the user as a command line parameter as follows:

./hw1 mazefile.txt

The first line of the maze file will contain the number of rows and columns of the maze as a comma separated value. The first value will be number of columns (X-axis) and the second value will be the number of rows (Y-value).

The second line in the file will contain the X, Y coordinate within the array that specifies the entry to the maze. Your maze traversal algorithm should use this coordinate as starting point.

The third line in the file will contain the X, Y coordinate within the array that specifies the exit to the maze. **Therefore the goal of your maze traversal algorithm is to find the path between the starting point and the exit point.**

You can assume that the first 3 lines of the file will follow the specification mentioned above and that these lines have a length that will not exceed 20 characters each.

The rest of the file contains the maze array specified as an array of characters. The 'X' character indicates a wall and the '0' indicates a walkable tile within the maze. At the end of each row the file will contain a line feed character ("\n"). **Do not change the format of the provided files in any way as different files will be used to grade your code.**

You must read the maze array from the file and copy it to a dynamically allocated multidimensional array. **You cannot assume an upper bound on the number of columns or the number of rows in the maze.**

You can assume that the actual number of rows and columns specified in the first line always match the actual maze array specified in the file. Also you can assume that the entry and exit points are always valid in the maze array.

Finally, you can assume that the maze specified always contain a path between the entry and the exit points specified in the file and that such path is solvable by the Wall Follower algorithm. **We will assume that it is not possible to walk diagonally**.

At this point is important to note that the Wall Follower algorithm provides a solution that does not guarantee to be the shortest path. This is fine, the purpose of this assignment is to familiarize with the particularities of the C language.

As part of this document we have provided 3 input files to use as test cases. However you should use additional test cases to test your code as the grading process will involve a more rigorous testing process with additional test cases not provided.

4.2 Wall Follower Algorithm

The next step involves the implementation of the Wall Follower Algorithm. The algorithm is based on 4 movement rules. Also, the algorithm requires you to keep track of the direction the person is facing. As this rules are always with respect to the direction the person is facing. The rules are as follow:

1) If there is no Wall at the Right of the person: Turn Right and walk one block
2) Else, If there is no wall forwards: Walk one block
3) Else, if there is no Wall at the Left of the person: Turn Left and walk one block
4) Else Turn 180 degrees and walk one block

4.3 Program Output

After finding the path between the entry and the exit point, your program must print to the console the structure of the maze showing the path between the entry and the exit. We will use ASCII characters to draw the maze on the screen. It is fine to show loops in your solution created by the Wall Follower algorithm after a dead end. Below is an example of a possible sample output for the solution of a simple maze in which X represents a Wall, 0 represents a Walking tile not part of the solution and W represents the solution:

```
XWXXXXXXXXX
XWXX0000000X
XWWXXX0XXX0X
XXWWWWWWWX0X
XXXWXXXXWXXX
XXXXXXXXWXXX
```

5. Makefile

As part of your program your solution you should implement a Makefile that automatically compiles your code. Your Makefile should include at least an entry to build your application (make hw1) and a cleanup entry (make clean) to delete all the files generated by the compilation process (i.e. object files, executables, libraries, etc.)

Furthermore, your Makefile should include the following GCC compilation flags: -O0 -ansi –pedantic –Wall. Also you must avoid using any other flag that interferes or circumvents the flags above. This flags above will ensure your code is ANSI C compliant and it will also provide you additional warnings about possible ambiguous or undefined behaviors in your code. The –O0 flag disables all optimizations and therefore it removes erratic behaviors in the debugger caused by some optimizations.

6. Software Engineering

You should include comments where appropriate. It is not a good idea to repeat what the function does using pseudo-code, but instead, provide a high-level overview of the function including function parameters, return values and assumptions. Some functions, might have as few as one line comments, while others might have a longer paragraph.

Also, your code must split into small functions, even if these functions contain no parameters. Code split into functions is usually much easier to debug as the problem can be easily isolated.

7. Hand-In

You should submit only one assignment per group. For submission, you should provide only source code (*.c, *.h) and a Makefile script as outlined in Section 5. You should also include a README file briefly describing how your code works and how to run your program. Please pack your files into a TAR file with the following filename structure CS201_odinId_HW1.tar. Please replace “odinid” with your ODIN id.

We will use the D2L system for submission (https://d2l.pdx.edu/). Please remember that there is no late policy.

8. Rubric

Grading will be done according to the following Rubric with partial credit given for features partially implemented.

Feature	Possible Points
File is opened and correctly parsed	25
Dynamic array is used to represent the maze	15
The wall follower algorithm correctly traverses the maze	30
The maze structure and the solution are presented on the console	15
The code has no memory leaks and no segmentation faults	5
Readme file contains the brief description of the code and instructions to run it	5
Code follows good engineering principles	5
Total	***100***

9. References

Homework 1 Overview Class Slides - http://www.raoulrivas.net/content/upload/02b-Hw1.pdf

Animation of a Wall Follower program - https://www.youtube.com/watch?v=U4N7bvGnByQ

GNU Make Manual (Look at Section 2.2 for a simple Makefile example) - http://www.gnu.org/software/make/manual/make.html

Kernighan & Ritchie, “The C Programming Language” 2nd edition, Chapters 1-5, 7.