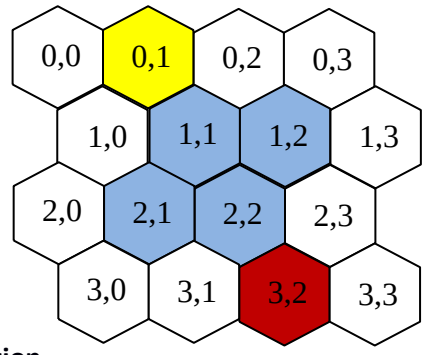
# CS 305: HW 2 Hexmaze path finder
# Fall 2019

**During the semester free tutoring is available to you for CS 305, please see tutors if you need to. If you have hard time in the class, e-mail the tutor fellow: stepup@up.edu for an appointment.**

**Due Date: NLT Sunday**, October 13th 2019 at Midnight (I very strongly suggest finishing up by October 7th). Your code files (.h and .c files), makefile, 10 sample inputs that represent different hexmaze boards you used for testing. In your archive also includes the answer questions in a pdf or docx file. Submit your HW as a single zip file to Moodle.

This assignment is meant to give you practice with recursion and working with files.

This assignment is to be completed **individually**. Advice: start as soon as the homework assignment is assigned, so you have plenty of time to get the code to compile, have time to test, have time to debug, have time to re-test, and have time to complete the summary report.




**Specification**

You will implement a path finding solution for an avatar that moves on a hexagonal grid. You will read the board configuration from a .txt file, implement a recursive path finding algorithm, and error check for different kinds of inputs and output scenarios. Your solution has to be in C. The above figure shows a sample 4x4 input board that has the avatar in the yellow starting cell (row: 0, column: 1), the goal to reach is the red cell (3,2) and the blue cells are the obstacles that the avatar cannot cross and cannot be included in the path. If a path exists from the source to the goal, show the board configuration at the end of the search, if the path does not exist print error message and show the board with the partial path found by the algorithm.

The input file will encode the above input board as follows:
4 4 0 1 3 2
0 0 0 0
0 -1 -1 0
0 -1 -1 0
0 0 0 0

The input file structure is as the following:
board rows, board columns, starting row, starting column, goal row, goal column
space separated 2D array of 0: available cell, -1 obstacle

If file's starting or ending coordinates are out of board's width and height, if the board is smaller than 1x1, if the 2D grid value is other than 0 or -1 print appropriate error.

Your solution has to be scalable to allow for any board size (as long as it can be loaded into memory).

1. The final code should include the following three files:

- `main.c` – test program (completed for you, included in starter code)
- `hexmaxe.h` – declares the `board` struct and the function prototypes that operate on the board
- `hexmaze.c` – defines following functions (you must create a new file for this)
  - `int find_path(char*);`
  - `int start_search(board*);`
  - `int search(board*, int, int);`
  - `int check_board(board*);`
  - `int is_safe(int, int, int, int);`
  - `void print_board(board*);`
  - `void free_board(board*);`
- `load.h` – declares the function prototype (completed for you, included in the starter code)
- `load.c` – defines the function
  - `hexboard* file_load(char*);`
- `cell.h` – declares the `cell` struct and two enumerated types (completed for you, included in the starter code)

2. You have to create a `makefile` that defines targets, so that the following commands work:

```
make find_path
make clean
```

`make maze` should compile the files correctly using gcc and create an executable called `find_path`
`make clean` should remove the executable `find_path` and all .o files
`make` has to have targets to make all intermediate object files: `main.o, load.o, hexmaze.o` and `maze`

3. Then, once `find_path` is created, the user runs it by:

```
find_path <file name with the board configuration>
```

or

```
./find_path <file name with the board configuration>
```

where `board.txt` is the filename of the hexmaze board to be tested

**GET STARTED**

1. Open `hexmaze.h and hexcell.h` both have the structs.

board
---
max_row:int
max_col:int
start_col:int
start_row:int
end_col:int
end_row:int
hexcells: hexcell** hexcells;
(2D array of hexcell structs, malloc'ed in load_file)

hexcell
---
color: colors
obstacle: obstacles

The board object has row and col size stored in row_max and col_max. All user clicks are entered as a cell number starting with cell number 1:1 to row_max:col_max , but the rest of your assignment will reference the cells as a 0 based offset index.

You do not need to edit the header files `hexmaze.h or hexcell.h or load.h`.

2. Open `main.c`. This file defines the main program. Checks the arguments and calls load_file for reading in the board.

You do not need to edit `main.c`.

3. Create a new file called `load.c`.

At the top, include "`load.h`", "`hexmaze.h`" and "`hexcell.h`".

Implement the following function (can copy and paste function prototypes from `load.h`):

**load.c: file_load**

args: unchecked file name to be opened for reading
out: returns allocated and value instantiated board* object or NULL if processing the input file failed
allocate gameboard object, check if successful allocation
open file for reading, check if successful
read the first line with 6 decimals separated by space that stand for: board's height (rows), board's width (cols), starting row, starting column, goal row, goal column. Save these variables into the gameboard object's respective variables row_max and col_max start_row, start_col, end_row and end_col
next create a hexboard object instance on a heap and sets max_row and max_col read from a file
creates (hexcell ** ) 2D array of hexcells as follows
    create a row array of hexcell pointers
    for each row entry create a col array of hexcells

read in the subsequent rows and columns from the file that contain a space delimited hexboard states and save these into the board object: -1 is a obstacle, 0 no obstacle

initialize the field color of the 2D hexcells to white
initialize the field obstacle of the 2D hexcells to the value read from the file
close the file

**hexmaze.c:print_board**

args: pointer to the current board struct
out: void
prints the hexboard board:
  if hexcell color is path, print 'P'
  if hexcell has an obstacle, print 'X'

otherwise print '0'

**hexmaze.c:find_path**
args: f_name -- unchecked file name that contains board configuration
out: int return board status 1 if path found 0 otherwise
creates the board instance
prints the board
call start_search with the loaded board struct
print the result of path search
print the resulting board

**hexmaze.c:search**
recursive implementation of the path search
args: board - current board state of row and col
out: return 1 if a valid path found and 0 otherwise

recursive algorithm will
check if current row and col coordinates are legal
check if current row and col are teh goal
check if current row and col have an obstacle
check if current row and col cell already belongs to path
mark current cell as potential path
process each spatially adjacent hexcell as an attempt to find a path
unmark the cell if neibors do not lead to a goal

**hexmaze.c:start_search**
args: current board state
out: returns 1 if search found path and 0 otherwise

Algorithm, this is a wrapper function for search method.
if the start or end cell coordinates are obstacles return 0,
otherwise call search with starting cell coordinates

**hexmaze.c:free_board**
arg: current board struct
out: void
deallocates all heap memory -- all rows, all columns, all hexcells

*Error-checking*
Error checking: if incorrect board dimensions or hexcell value are read from a file, gracefully exit. As mentioned, check the file opening and the object allocations.

*Documentation*
The code should be well-commented, well indented, use consistent bracing, and include enough whitespace to be readable. The .c files should have a block comment at the top that includes information about the author and included code.

*Test cases*

I provided 5 boards for testing. You should create at least 10 more new boards for testing your program. Include these files with your final submission

**Example Output**

sample output can be generated using provided compiled solution and the test boards.

**Additional Enrichment**

If you have time, you may add the code to find the shortest path, create a hexmaze board generator etc., please save a copy of the original homework and put it in a folder called `orig`. Then, copy the files for the enrichment into a separate folder called `enrich`.

Please document the additional features in your code and in your written summary. This is not an extra credit per se, but a chance for you to do some exploration.

**Logistics**

1. Download the starter code located on Moodle. If using Linux, you can unzip it with the unzip <name of zip file> command. You may use any IDE that you can find for C or just a text editor (emacs) to write your code. However, your code *must compile* with gcc on UP's Linux VDI
2. When you are finished and ready to submit:
    1. Create a new folder called username_HW2.
        1. (For example, Pluto Disney's would be disney_HW2.)
    2. Put all .h, .c, makefile and your 10 test boards you used for additional testing in your folder.
    3. Copy your username_HW2_summary.pdf (or .docx) file into this folder.
    4. Zip the folder by right-clicking and "compress"
3. **What to turn in:** Submit your **username_HW2.zip** file to Moodle before the due date and time. After logging into learning.up.edu, navigate to CS 305. You will find a link to submit this file. You do not need to print anything for this homework submission.

**Grading Guidelines (total of 20 points)**

Your files will be graded in three separate categories:

- (5 points) Code Quality: Design, commenting, whitespace, readability, proper indentation, consistent naming conventions, good variable names
- (10 points) Code Operation: Does code do what is listed in the specification? Note: code that does NOT compile will receive 0 points for code operation.
    - 1 points file_load: create game object and load file into it
    - 5 points find_path, search_start, search, print_board
    - 2 points to correctly find path
    - 1 point free_board
    - 1 point makefile (works for compiling and cleaning)
- (5 points) Summary report: Completeness, correctness, and clarity

**HW 2 Report Guidelines and Format – use the template provided below. Include the questions in your write-up.**

**Name:** **CS 305 HW 2 Report**

**1. Questions (include these in your write-up):**

1a. (.5 point) If your program does not meet the specifications, please note the differences and anything that is not working correctly. If it works, write “Meets all specifications.”

1b. (.5 point) Copy and paste your terminal window contents from running the find_path hexmaze on your 10 testing boards (similar to what you would get by running my solution on the test boards provided). Make your boards simple enough to illustrate program’s operation – both checking for the correct and error cases.

2c. (1 point) Complete the table below about the new test boards that you created.

**Table 1: Testing of new boards**

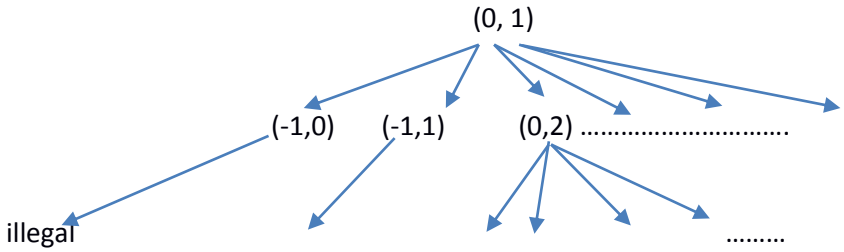
| Filename | What is wrong (or correct) about the file? | What your program did when running on this file |
|---|---|---|
| board6.txt | | |
| board7.txt | | |
| board8.txt | | |
| board9.txt | | |
| board10.txt | | |
| board11.txt | | |
| board12.txt | | |
| board13.txt | | |
| board14.txt | | |
| board15.txt | | |

2d. (.5 point) Do you have test cases for different possibilities of incorrect board configurations in the file? If not, state the cases that you did not test.

3. (2 points) Suppose the following small hexboard is used as input:

```
4 4 0 1 3 2
0   0   0 0
0 -1 -1 0
0 -1 -1 0
0   0   0 0
```

Draw a recursive tree of stack calls that are made when finding a path from the start cell (0,1) to the goal cell (3,2). Complete the tree. If a recursive call ends in false, you may quit that branch. You can save writing by using (2,1) to stand for the stack frame with the coordinates row 2 column 1(2,1).




(continue completing this chart)

3a. (.25 pt) How much time did you spend in total on this homework assignment (including the report)?

3b. (.25 pt) What was the most challenging part for you to complete in this assignment?

**Appendix A: (copy this statement if it applies to you)** I verify that the code and this write-up were authored by me. I have documented the help I have received in comments in the code files.

**Appendix B**: Copy and paste your hexmaze.c, load.c, files and makefile here (use Courier New 8pt font so the characters line up correctly)