

Programming Project 3

Due: Monday, 2/22/15 at 11:59 pm

Maze Solving

For this lab, write a C program that will find its way through a maze using the depth-first search algorithm. This program takes input from a file where the filename is specified in the command line argument that contains two integer values per line of input:

- The first valid line gives the size of the 2-D maze (the number of rows given by the first number, then the number of columns given by the second number), valid values are ≥ 1
- The second valid line gives the coordinates of the starting position in the maze
- The third valid line gives the coordinates of the ending position in the maze
- The remaining valid lines in the file give the coordinates of blocked positions in the maze

If the command line arguments do not contain a valid filename, you should print an error message to STANDARD ERROR and quit the program. This could be because there was not a name given or the name given did not match that of a valid file. If the command line arguments contains multiple names, we leave it up to you to come up with a “good solution”. Ideas on such a “good solution” could be:

- Use one of the names as the input file and proceed (pick which name makes most sense to use).
- Give an error message stating that too many names were given and quit.
- Run the program to solve the maze given with each filename.

The following shows an example of such an input file. The coordinates are given with the row listed first and the column listed second. A maze of $N \times M$ has rows numbered from 1 to N and columns number from 1 to M .

```
10 20
1 1
10 20
5 1
4 2
3 3
1 10
2 9
3 8
4 7
5 6
6 5
7 4
8 3
```

This input creates the following maze with 11 blocked positions:

```
size: 10, 20
start: 1, 1
end: 10, 20
*****
```


- Mark all unblocked positions in the maze as "UNVISITED"
- push the start position's coordinates on the stack
- mark the start position as visited
- While (stack is not empty and end has not been found)
 - if the coordinate at the Top of the Stack is the end position
 - then end has been found
 - if the coordinate at the Top of the Stack has an unvisited (and unblocked) neighbor
 - push the coordinates of the unvisited neighbor on the stack
 - mark the unvisited neighbor as visited
 - else
 - pop the coordinate at the Top of the Stack
- If the stack is empty
 - The maze has no solution
- else
 - The items on the stack contain the coordinates of the solution from the end of the maze to the start of the maze.

When referring to neighbors, those positions will be the ones above, below, left or right of the current position (not diagonal). So for position x,y its neighbors are at:

- x+1, y
- x-1, y
- x, y+1
- x, y-1

Your program is to first output the size of the maze, the start and ending coordinates and an ASCII drawing of the maze. The code `maze.c` does this for any maze of size 30X30 or less. The `maze.c` program uses a static sized 2-D array; **however, your program MUST use a dynamic 2-D array sized to reflect the maze size given in the input file.** You must also dynamically deallocate this array at the end of your program. The `maze.c` program also does not do any error checking for invalid input, your program MUST check for invalid input.

Once the maze solving algorithm is run, you must then print out a message stating either:

- the maze has no solution

or

- listing the coordinates of the locations of the path in the maze from the start of the maze to the end of the maze that was found by the algorithm. Note this means printing out the contents of the stack in reverse order.

The stack MUST use a linked list of coordinate values. **The head of the linked list MUST NOT be global. It must be declared as a local variable in main() or some other function.** You may create a structure to contain the head of the stack if you desire but again the initial instance of the structure must be a local variable. The code for each stack operation MUST be done in its own function where the head is passed in as a parameter.

Command Line Argument: Debug Mode

Your program must be able to take one optional command line argument, the `-d` flag. When this flag is given, your program is to run in "debug" mode. When in this mode, your program is to display the coordinates of the maze positions as they are popped off the stack if the Top of Stack coordinate does not have an unvisited (and unblocked) neighbor. When the flag is not given, this debugging information should not be displayed.

Since the input file for the maze also comes from the command line arguments, you may not assume which order in which the command line arguments are given. Thus the command line arguments may be given as:

- `./a.out mazeInput.txt`
- `./a.out mazeInput.txt -d`
- `./a.out -d mazeInput.txt`

One simple way to set up a "debugging" mode is to use a boolean variable which is set to true when debugging mode is turned on but false otherwise. Then using a simple if statement controls whether information should be output or not.

```
if ( debugMode == TRUE )  
    printf (" Debugging Information \n");
```

Optional ways for writing this code can be seen at:

- <https://www.cs.uic.edu/pub/CS211/LectureNotesF12/debugmode.c>
- <https://www.cs.uic.edu/pub/CS211/LectureNotesF12/debugmacro.c>

Program Submission

You are to submit the programs for this lab via the Assignments Page in [Blackboard](#).

To help the TA, name your file with **your own net-id** and the assignment name, like:

- `ptroy1LabX.c`