Project 01: Word Search Puzzle
CSE-031-01 **Points: 100**

Overview

In this project, we will use all the topics we have learned about C to write a program to solve Word Search Puzzles.

Description of `wordsearch.c`

The main program (`wordsearch.c`) is given to you. Your primary task is to implement the `searchPuzzle()` function to complete the program. When the program starts, it will read the puzzle grid from a text file and save it as a 2-D character array. The first number in the text file containing the puzzle grid indicates the size of the grid. For instance, a 5 means the puzzle is a 5 × 5 grid. The program will then ask the user for the word to search and store it in the variable `word`. The program should then print the original puzzle grid and search for the word in the puzzle. The program should finally print whether the search word is found, and if so, the search path as described below.

Your Tasks

`searchPuzzle(char**, char*)` – This function is the core of solving the puzzle. It takes in the puzzle grid to find the search word input by the user (the 2nd argument) and prints the phrase `Word found!` and the path(s) if the word is found. If the word is not found, then it prints `Word not found`. **The search will happen *in a case insensitive manner*** and ***all directions are allowed.*** The letters in the found word (i.e., the path) must be **one index away from each other** either in row or column or both (or in simpler terms, adjacent to each other) as shown in the sample runs.

- **YOU MUST NOT USE ANY ARRAY NOTATION ([]) IN THIS PROGRAM!**
- **YOU MUST NOT USE ANY LIBRARY FUNCTIONS TO CONVERT CHARACTERS INTO LOWER/UPPER CASE!**
- **YOUR OUTPUT FORMATTING MUST EXACTLY MATCH THE SAMPLE RUN IN TERMS OF SPACING, WORDING OF PROMPTS AND NEWLINES!**

Feel free to create any helper functions and additional arrays to simplify your `searchPuzzle` function. To help further explain the expected behavior of your code, some examples follow along with explanation of the output:

Sample Run (user input shown in blue, with each run separated by a dashed line):

```
----------------------------------------------------SAMPLE RUN 1 (./wordsearch puzzle1.txt)
Enter the word to search: HelLo

Printing puzzle before search:
W E B M O
I L H L L
M L Z E L
M Y E K O
A O A B A

Word found!
Printing the search path:
0       0       0       0       5
0       0       1       3       4
0       0       0       2       0
0       0       0       0       0
0       0       0       0       0
```

```
----------------------------------------------------SAMPLE RUN 2 (./wordsearch puzzle2.txt)
Enter the word to search: bAnANa

Printing puzzle before search:
J Z I M O B
O T H N G A
E R B Q P W
M A E K O Z
A T N R A E
E N O B T K

Word found!
Printing the search path:
0       0       0       0       0       0
0       0       0       0       0       0
0       0       1       0       0       0
0       642     0       0       0       0
0       0       53      0       0       0
0       0       0       0       0       0
----------------------------------------------------SAMPLE RUN 3 (./wordsearch puzzle3.txt)
Enter the word to search: deTeR

Printing puzzle before search:
J Z I D O
E T H N E
E R Z T R
M D P K O
A T F R A

Word found!
Printing the search path:
0       0       0       1       0
4       3       0       0       42
2       5       0       3       5
0       1       0       0       0
0       0       0       0       0
----------------------------------------------------SAMPLE RUN 4 (./wordsearch puzzle3.txt)
Enter the word to search: color

Printing puzzle before search:
J Z I D O
E T H N E
E R Z T R
M D P K O
A T F R A

Word not found!
```

Explanation of Sample Runs

Please note that for the **first sample run** shown above, the path output is not unique, but your solution needs to output only one of the many viable paths *if those paths overlap in the first letter* (`'H'`) – see explanation of third sample run when the first letter in the multiple paths found are at different locations in the puzzle. For example, alternative solutions to this puzzle include the following paths and your solution should output any *one* of these, including the example provided in the sample output:

```
0       0       0       0       5                       0       0       0       0       0
0       0       1       0       4                       0       0       1       0       3
0       0       0       2       3       or              0       0       0       2       4
0       0       0       0       0                       0       0       0       0       5
0       0       0       0       0                       0       0       0       0       0
```

For the **second sample run**, finding the word in the puzzle involves *backtracking*. Your solution must print the path as shown. Here `642` denotes that the character at this location is the 2nd, 4th and 6th in the search path. Your program may also output `246` instead of `642` at this location (and subsequently, `35` instead of `53` at the other location shown in the output).

For the **third sample run**, the program produces two path outputs as *these paths do not overlap in the first letter* (`'D'`). **This is only a bonus feature and is worth 10 extra points (in addition to the 100 total points)**. For the basic solution, *your program need not produce multiple paths and a single one will suffice*. Note that there is one more possible solution that is not shown since one of the paths overlaps in the first letter of the path shown in the sample run output:

```
0       0       0       1       0
0       3       0       0       42
42      5       0       3       5
0       1       0       0       0
0       0       0       0       0
```

This sample run is an example of multiple occurrences of a word in a puzzle, which for this project, means that the first letter of each occurrence is at a different location (subsequent letters can coincide in location). However, it is also perfectly fine if your solution (assuming it implements this bonus feature) finds multiple occurrences of a word where the first letter is at the same location. In this case, the assumption would be that some of the subsequent letters for each occurrence are at different locations (e.g., the 3 occurrences of the word 'hello' in the first sample run).

Testing Your Program

After compiling `wordsearch.c`, run the program by typing `./wordsearch puzzle1.txt`, where `wordsearch` is the executable file, and `puzzle1.txt` is the text file containing the puzzle grid. Feel free to create your own text files for test cases.

Collaboration

You must credit anyone you worked with in any of the following three different ways:

1. Given help to
2. Gotten help from
3. Collaborated with and worked together

What to hand in

When you are done with this project assignment, submit all your work through CatCourses.

Before you submit, make sure you have done the following:

- Your code compiles and runs on a Linux machine (without the need for special libraries).
- Attached `wordsearch.c` and any additional test files, pseudocode, idea sketches, notes, etc.
- Filled in your collaborator's name (if any) in the "Comments…" text-box at the submission page.

Also, remember to demonstrate your code to the TA or instructor before the deadline.