CSE 102 Programming Assignment 4

DUE

December 6, 2022, 23:55

Description

- This is an individual assignment. Please do not collaborate
- If you think that this document does not clearly describe the assignment, ask questions before its too late.
- This assignment is about simple file read/write, array usage and string operations. You can use pointers and functions. You cannot use dynamical memory allocation. You cannot use recursion. You can use global variables. You can use standard string library operations.

This is a `C` Programming assignment. You will write a `C` program according to the following description.

- You are given a file(`input1.txt`) which contains a list of words(one word at a line).
- You are given a file(`input2.txt`) which contains lines of character strings of various sizes.
- Your program should read the given files (`input1.txt` and `input2.txt`) and for each word in `input1.txt`, find the location and the orientation in `input2.txt`.
- Strings can appear vertically or horizontally. Once all the strings are found, your program should write their head position and the orientation to `output.txt` file.
- The order of words in `input1.txt` and `output.txt` should match. (Ignore the missing words)
- Some words may be missing in `input2.txt`. `output.txt` should not include the missing words.

Example

Contents of `input1.txt`:

```
Xanthos
Patara
Myra
Adana
Arycanda
Phaselis
```

Contents of `input2.txt`:

```
askjdkdanmowflfkadkjkaksdklskksadllasd
fkfklsal;sdlslfsdkdfkMyraslfsfeokdlasdfdf
wfewfPsdflsdfklsdosfkeowkfweofkwoefkkfsf
sdfsdakjewofksdfoiwefjowefks;dfPhaselissdfkefowefjksdlfjslfjsldfj
oefkwtifjslflsdfkjeifjsl;asldkiefjiefjlskfdjeifjksk
sdlfkawioAjksdlfkeifsflkefjiwfejsllfjeighghj;wp
dfkjerwfjr;s;dlskfjfiefwflwekfjwlefiehgwefjiw
sdfkkafwXanthosdkfjlsAfdjiefeiwowefwefiefjowiefehgl;lewofjefw
iieoefnFGSAiejfiwefjhrwefiiwejfiwjefihhwefklskdf
skkfjwifjwiiwifieifwfy[pwpfjwifwefijijwhi]
sdfijwiefjiwhiwiejfiwcifjweofi;aiajdijeifwuiefhwoeifjowehi
iiwefiwefjwiefjwiefjiajiefiiwejflkajsdlkjfiewhfiwefjowiefhhaskjlskdfu
iwjefijefmmiewifjweifnklksjdiaifwefpwefijfiwfOWOEFIEWfnweiuf
jfnwefiwfpwfjiejfiwefdjwflljfwiefjiwefwflijijfiewfhu
kkfiefiewpfkfjfjhewfoawefkwoefoejwfowoefkowefkowo
```

`output.txt`:

```
Xanthos (8,9) E
Patara (3,6) S
Myra (2,22) E
Arycanda (8,22) S
Phaselis (4,32) E
```

Search Directions

There are 8 possible directions. Labels are:

- N: North
- S: South
- E: East
- W: West
- NE: North East
- NW: North West
- SE: South East
- SW: South West

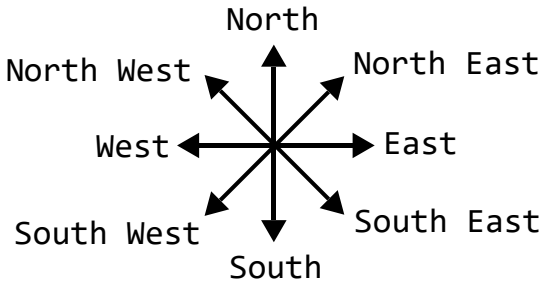

Figure 1: Directions

Bonus Part: 50pts

When searching in any direction, you stop when you see a null character. In some cases, one or many rows can be shorter than the rows above or below. In such cases, there will be sub sections(regions) in the matrix which requires the search to start over.

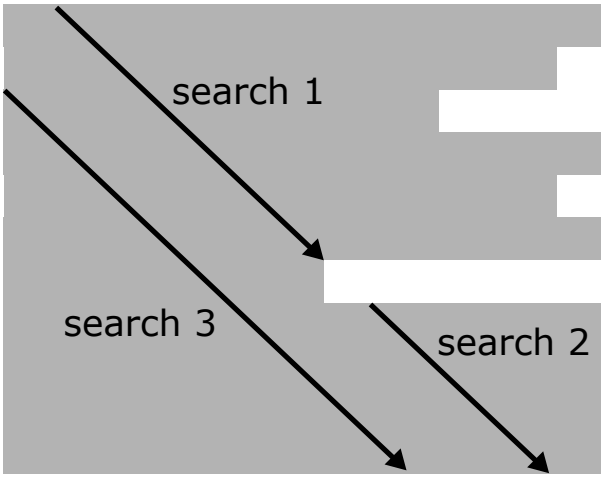

Figure 2: A case which needs multiple starts of search on the same diagonal

In this figure, you can see that, `search 1` will end when it sees a null character. You have to do `search 2` in order to search the rest of the data in that direction. If you implement this part, you will receive `50pts` bonus.

Beware of the fact that you can see such cases when you search in any direction except in the `East` direction. You have to handle all of them in order to get the bonus points.

Another thing about this special case is that, you can have multiple sub regions in the same direction. You have to handle them all.

Remarks:

- For `input2.txt`, strings which correspond to rows are separated by whitespace.
- For `input2.txt`, Maximum length of each row is `250`
- For `input2.txt`, Maximum number of rows is `100`
- Position indices start from `1`.
- For `input1.txt`, there is no limit on the number of words.
- For `input1.txt`, maximum word length is `10` characters.
- You don't have to do error checking on the input file. You can safely assume that you will be given a proper input file which doesn't violate the described format.
- Be careful with the size of the array you allocate in program stack. Large arrays may not fit in program stack(stack size may be smaller on the test machine) and your program crashes.
- Make sure you can read input files with or without a tailing newline at the end. (If you are using a windows machine, newline is `CRLF`, on unix it is `LF`). You can alter this using advanced editors (i.e. Visual Studio Code). Test your code for every possible combination.
- Do not print anything other than the expected output. (For this assignment, your program prints NOTHING).
- You cannot use anything which is not covered in class.
- Do not submit any of the files you used for testing.
- Do not submit your output file.

Turn in:

- Source code of a complete C program. Name of the file should be in this format: `<full_name>_PA4.c`.
- Example: `okay_temiz_PA4.c`. Please do not use any Turkish special characters.
- You don't need to use an IDE for this assignment. Your code will be compiled and run in a command window.
- Your code will be compiled and tested on a Linux machine(Ubuntu). GCC will be used.
- Make sure you don't get compile errors when you issue this command : `gcc <full_name>_PA4.c`.
- A script will be used in order to check the correctness of your results. So, be careful not to violate the expected output format.
- Provide comments unless you are not interested in partial credit. (If I cannot easily understand your design, you may loose points.)
- You may not get full credit if your implementation contradicts with the statements in this document.

Late Submission

- Late submission is NOT accepted.

Grading (Tentative)

- `Max Grade` : `100`.

All of the followings are possible deductions from `Max Grade`.

- No submission: `-100`. (be consistent in doing this and your overall grade will converge to `N/A`) (To be specific: if you miss `3` assignments you'll get `N/A`)
- Compile errors: `-100`.
- Irrelevant code: `-100`.
- Major parts are missing: `-100`.
- Unnecessarily long code: `-30`.
- Using language elements and libraries which are not allowed: `-100`.
- Not caring about the structure and efficiency: `-30`. (avoid using hard-coded values).
- Significant number of compiler warnings: `-10`.
- Not commented enough: `-5`. (Comments are in English).
- Source code encoding is not `UTF-8` and characters are not properly displayed: `-5`. (You can use 'Visual Studio Code', 'Sublime Text', 'Atom' etc... Check the character encoding of your text editor and set it to `UTF-8`).
- Fails at reading the file: `-30`.
- Fails during file write: `-30`.
- Order of the word list is wrong: `-20`.

- Coordinates of horizontally placed words are wrong: `-10` (each).
- Coordinates of vertically placed words are wrong: `-10` (each).
- Missing words: `-20`.
- Incorrectly matching words: `-20`.
- Output format is wrong: `-30`. (Be careful with spacing)
- Infinite loop: `-90`.
- Prints anything extra: `-30`.
- Unwanted chars and spaces in `output.txt`: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-200`.
- IF YOU DON'T FOLLOW THE FILE NAMING CONVENTIONS YOU WILL GET `0`.

Note: Some of these items are not independent. So, you cannot expect isolation of many of them. For example, if you cannot read input file correctly, you will fail to produce the correct output file. Partial grading is not guaranteed.