CS/SE 2337 – Homework 2 – Word Search

Dr. Doug DeGroot's C++ and Data Structures Class

Due: Sunday, October 1, 2023, by midnight

How to submit: Upload

1. your source code file (.cpp) and
2. a copy of your program's output in PDF or TXT format. (Copy your console output and place it in a text file or create a PDF file with it.)

Your program must compile on either Codeblocks or MS Visual Studio 2015 (or later).
Name your cpp file as follows:

HW2-CS2337-John-Doe.cpp

Maximum Number of Points: 100

Objective:

Write a program that searches for specified words hidden within a matrix of letters.

1. Ask the user for the name and location of a file to read; this file will contain the Word Search puzzle. Read the file. The file may contain comments linankes (they will start with a # symbol), data lines, and blank lines. The first non-comment/non-blank data line in the file will specify the dimensions of the matrix (e.g., 20 30 or 20 x 30). These two numbers specify the number of rows in the matrix and then the number of columns in the matrix.
2. Please embed the required smaller matrix within a larger 40x40 matrix at this point. We will see much better ways of handling this issue once we study points and memory management, but for now, let's practice using a subset of a larger memory element.
3. Following the two array dimension numbers, there will be a number of rows of letters. These letters should be read into your matrix (a 2D array of chars) row by row, column by column.
4. Following the rows of letters, there will be some number of words/phrases that might or might not be hidden within the matrix. You are to search for these words/phrases and report 1) whether or not a given word/phrase is found within the matrix, and if so, 2) the starting coordinates of the word/phrase and the direction in which you found it, e.g., N, NW, SE, etc. Report the starting locations using 1-based numbering rather than 0-based numbering.
5. Search for each word (name, title, phrase, etc.) one at a time (i.e., read a word/phrase, search for it within the hidden-word matrix, report the results, and then read the next word/phrase from the file). Continue until all words/phrases have been searched for and reported.
6. I'm going to upload several versions of our hidden words puzzle to eLearning for your use. I suggest starting with the Comedy Movies puzzle called ***ComedyMovies-PARTIAL.txt***. Once you can process this puzzle correctly, you should try to process the file ***ComedyMovies-Final.txt***.
7. Provide for fault-tolerance when trying to open the input file. If it doesn't exist or can't be found, tell the user. Also, tell the user the name of the file that can't be found/opened. Then give the user a chance to enter another file name and try to open that file. Continue until you either find and open the file or until the user types something like "quit" or "exit." If you feel comfortable doing so, use the *perror* function to report the system's error message; then report your own error message as well. (There's an uploaded document on perror() on eLearning.) Make sure your program can handle bad file names and/or non-existent files. Don't forget that you can also use directory navigation as well.
8. Don't report success or failure for each word as you go. Instead, report only words that are found, their locations, and their directions; but for words that are not found, shove them into a *vector of*

strings and wait until you've searched for all words before reporting the words that weren't found and then finally quitting. Thus, you will have all successfully found words appear first -- along with their coordinates and their direction -- and then the words that weren't found will be listed at the end.

9. Report the indices where movies are found in human-speak, not in geek-speak. (See the example output at the end of this file. Don't report coordinates using 0-based index numbers.)
10. You could add a Boolean variable that controls whether you search for multiple or just single occurrences within the matrix. This is optional, but please consider adding it.
11. **For extra fun**, consider adding an interactive capability to your program that asks the user to enter additional words or strings to search for in the matrix once you've searched for all words in the data file. Search for a few and report the results. (Your search string doesn't have to be a movie; it can be pure gibberish. It just needs to be some string that your program can search for, preferably strings that you can spot visually in the input matrix.)

Annotations for the code:

1. The main function can be at either the beginning or the end of the program. I don't care which.
2. Add comments as the top three lines of your program file to include your name, the name of the program, and the date you created the program.
3. Then include comments stating any relevant notes on how your design works when executed. This is a requirement.
4. Add a change log in your comments as you go that lists the major changes you make to your program and when – nothing too terribly detailed, but a list of breadcrumbs that will remind you and others of what you've done as your program becomes more sophisticated and/or nearly complete.
5. Point out (in the comments at the top of your program) any special features or techniques you added using a comment saying something like "**// Special Features:**" or "**// Notes:**"
6. Comment your code effectively, as we discussed in class. Don't use redundant comments, but instead make your program self-documenting. Use descriptive variable names everywhere so that the code becomes as self-documenting as possible. Use additional commentary only to improve readability and comprehensibility by other people. No fluff!
7. You absolutely MUST use consistent indentation and coding styles throughout the program. Failure to do so will result in a loss of 5 points.
8. If the program does not work at all, or works incorrectly, at least 10 points will be deducted. So you must find every word that is in the matrix and must report all words that are NOT in the matrix. The starting coordinates and direction must also be correct.
9. Please meet the deadline.

Note:
We will need to discuss the **peek()** function in class, the STL Vector container, and fault tolerant file opening. Don't worry; they're fun. However, you can and should begin designing and coding your program now.

Note: I am uploading two data files for you to try.

ComedyMovies-PARTIAL.txt
and
ComedyMovies-Final.txt

To Submit:
Copy the output that your program writes and include it in your homework submission as a separate file. The output should look something like the following:

```
Oops, can't open the specified input file: No such file or directory
The file name used was: I_Dont_Exist.txt
Enter another file name to use (or quit): ComedyMovies-Final.txt
--- The new file name is: ComedyMovies-Final.txt

Nbr Rows: 18;  Nbr Cols: 18
SCFRBOBROBERTSLOKL
TNLESREKCILSYTICRL
AOEPOBLRUAECHBUDEA
RITOTHEYOMTOLBATPH
LTCMWRGIFMLBEEIBEE
LCHAIBAONILLAURGEI
AEKNLULDDGCISLHKLN
MLCTILLAINTAGOLHSN
AEUDALYCUNZHSYLOSA
CRRURDBHSAGTEOPLUS
LOTCLULHLLBPCRAPLA
UMSKIROPOULALPEAAK
EHNSAHNNSTLASACCEH
LSOORADTEHSHBKCROA
EUOUFMEMECOHETHEER
SRMPARTRBTAROSAASV
SLEISTOOTHERUTLESE
NATTAHNAMSEYSLSSMY

ALL OF ME found at 9, 5: (direction = NE)
ANNIE HALL found at 9, 18: (direction = N)
BABE found at 3, 14: (direction = SE)
.
.
.
UNCLE BUCK found at 9, 9: (direction = NE)
YES MAN found at 18, 12: (direction = W)

Couldn't find these movies:
Raising Arizona
Zzzz

[and maybe others!]
```