CX 4010 / CSE 6010
Partner Assignment 2
Word Chains
Note extra requirements for graduate students only!

Submission Due Date: 11:59pm on Thursday, October 21; 48-hour grace period applies
note no initial submission
Submit a single zipfile as described herein to Canvas!

In this assignment, you will write a C program to automate solving a type of word puzzle we will call a "word chain." The idea behind a word chain is to progressively transform one specified word to another specified word of the same length through a series of steps where only one letter is changed at a time. In addition, each transformation must result in another word.

Here are a couple of examples. (Note that example words used here are illustrative and may not all be present in the word file you will use.)

Change PASS to TEST:

PASS → PAST → PEST → TEST

Change YOUR to BEST:

YOUR → POUR → POUT → POST → PEST → BEST

Here are a couple of examples of transformation steps that are not allowed:

TAIL → TALE (more than one letter changed in a single transformation; you'd need multiple transformations like TAIL → TALL → TALE)

WORD → WZRD (not a valid word)

The transformation process can be viewed as a graph problem. Each word can be represented as a node, and there is an edge between two nodes if exactly three of the letters of the corresponding words are the same (positions must be the same as well). Then, a graph algorithm can be used to determine a shortest path through the graph from a specified source word to a specified destination word, and the path can be printed. In this problem, we will assume that a relatively small number of words with four letters specified in an input file are the only allowed words in the chain.

Division of labor

You will work in teams of two (occasionally 3 when the total number of students in a section is odd) within your section (CX4010 or CSE6010). Your code should be modular, such that the main function is relatively short and easy to read, with the bulk of the detailed work performed in functions.

One person should write the code to read in the provided file, which has 500 words with four letters, then generate the graph by forming connections. The source and destination words should be specified as two **command-line arguments** using integer indices corresponding to the word's position in the file (e.g., the first word is node 0, …, the last word is node 499). Note that there are many ways to read in and store words; for example, you may use `fscanf` to read a word as a string to a temporary variable at a given address, followed by `strcpy` or `strncpy` to copy the word from the temporary variable to its ultimate location (you may need to look up syntax for the string functions if you choose to use them, but they are not hard to learn). Graduate students should allow an additional transformation whereby two adjacent letters in a word may be swapped provided this results in a legal word. For example, a direct connection between LIEN and LINE would be allowed and would need to be represented as an additional edge in the graph. Use of this option should be specified by an additional command-line argument specified in the README file. This argument should be the last in the list, and the default case with only two command-line argument should construct the graph without allowing this additional "swap" transformation.

The other person should use the generated graph to find and print out the length of the shortest path along with the shortest path found. The printed path should only show the word chain (no node indices). If there is no path, the program should print a suitable message. Graduate students should also print out the total number of words that are "reachable" from the specified source word via the allowed transformations and the average number of transformations needed to reach the reachable words (this will be one less than the number of words in the chain if the source and destination words are both counted—so the number of arrows in the example chains above).

You and your partner(s) may work together to define needed data structures, work with the input file and command-line arguments, and produce the slides, and for other tasks you deem necessary.

If you are assigned to a group of three, two of you should develop separate implementations for one of the tasks described above such that each can be combined with the work of the third group member. We recommend only one student construct the graph and the other two write different implementations to find the shortest paths. Overall, each team of three then will have two complete implementations to solve the problem, each with one portion developed by a single student and combined with the other portion developed by one of the other two students.

Undergraduate students may use either an adjacency matrix representation or an adjacency list representation of the graph. Graduate students must use an adjacency list. Regardless of the graph representation used, the program should be modular with a fairly short main function and no global variables.

Group assignments will be available in Canvas, with an initial sign-up period for those who would like to choose their partners. After the sign-up period, partner assignments will be made randomly, but assuming a preference to work with a different partner if possible.

Submission: You should submit to Canvas a single zipfile that is named according to the Georgia Tech login of someone in your group—the part that precedes @gatech.edu in your GT email address. For example, I would name my zipfile echerry30.zip. To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

The zipfile should include the following files:

(1) your code, with a main file named **wordchains.c** and additional .c and/or .h files as needed/desired. (Groups of three should name their main files wordchains1.c and wordchains2.c.) If you are using linux or Mac OS, we recommend you use a makefile to compile and run your program, and you should include it if so. **It is expected that your code will compile and run on the COC-ICE cluster.**

(2) a README text file (not formatted in a word processor, for example) that includes the compiler and operating system you used for compiling and running your code along with instructions on how to compile and run your program. If you are in a group of three, you should include instructions for both complete implementations in the same README.

(3) a series of slides composed in PowerPoint or similar software, saved either in PowerPoint or as a PDF and named slides.pptx or slides.pdf, structured as follows:

- 1 slide: your names and a description of the division of labor across the team for this assignment (who did what) with specific references to the code (e.g., files, functions, data structures, etc.).
- 1 slide: a brief explanation of your approach for representing the graph and generating its connections, how well you think your approach works, and any ideas you may have for future improvements. (Groups of 3 may need 2 versions of this slide, although we recommend 2 versions of the shortest path component.)
- 1 slide: a brief explanation of your approach for calculating and outputting shortest path, how well you think your approach works, and any ideas you may have for future improvements. (Groups of 3 may need 2 versions of this slide.)