# CS-231 C Language Lab 4

### Spring 2019

## Overview

Write spelling checker programs to work together via pipes and signals. Each unique word of input in an input file will be checked against a file of words (zipped dictionary file attached as part of this assignment) and printed with a message stating it is/is not spelled correctly. This is a 60 point program.

# Specifics

You will write lex.c, compare.c, and spellCheck.c and compile them into lex.out, compare.out, and spellCheck.out. All communication will be via pipes for this assignment. Specifically, you will use the pipe function multiple times to create pipes. Remember that pipes are one way communication channels. You will also be using the sort and uniq commands, making sure that the words input from a file are sorted in alphabetical order and each word appears once. Your programs will ignore case, that is, the words 'At' and 'at' will be the same word in your code. **When you write your test file, include multiple words on each line, and include some words which start with an upper case letter and some that start with a lower case letter. All words beginning with either 'a' or 'A' will be output before all words beginning with either 'b' or 'B', etc.**

Your program lex.out will take as a command line argument the name of a file. All words in that file will be written to stdout, one word per line. A word is defined as a sequence of alphabetic characters. Note that this means every character which is not alphabetic is not part of a word, so each sequence of nonalphabetic characters will be skipped over. Any letter which is alphabetic will be output as is, no change in case is allowed.

The program compare.out will take two files of input. One file of input is stdin, which will be attached to the pipe used for output by uniq. The other file of input is the dictionary file which is part of this assignment. You will write compare.c so that each input file is read once, which is possible since both files are in alphabetical order. Only one word of each input file will be in your program's memory at a time. Note that a word from uniq will be considered the same as a word from the dictionary file if they differ only in capitalization.

The program spellCheck.out is the controller program. You will run spellCheck.out with the name of a file as the first command line argument. This file will be checked against the dictionary, whose name will be specified as the second command line argument. Each unique word in the input file will be checked against the dictionary, and each word will be printed with the message that it is/is not spelled correctly. This output will be sent to stdout. The way spellCheck works is this: it will use fork, exec, wait, and create pipes so that the input file will be processed by lex.out, the output of lex.out will be processed by sort, to do an

alphabetic sort ignoring capitalization, the output of sort will be input to uniq to remove duplicate words again ignoring capitalization. The output of uniq and the the dictionary file will both be input into compare.out, which will determine if the words from uniq are in the dictionary file.

The program spellCheck.out will work in the following way:

- spellCheck.out will track each child process it creates, that is, it will store the result of each call to fork()
- spellCheck.out will enter a loop which will end after all the child processes have terminated
- spellCheck.out will call waitpid() in a signal handler for the signal SIGCHLD. This call to waitpid() will not block. The signal handler will print a message stating which child process terminated (both by id and by name, such as lex.out) to the file "spellCheck.log" which spellCheck.out will create.

# Notes

- You will need to write spellCheck.c to fork child processes to exec lex.out, compare.out, sort, and uniq. Do not attempt to write all of spellCheck.c at one time because if there is a problem with your communication paths, you may not be able to discover which path is at fault. I highly suggest that you write spellCheck.c to communicate only with lex.out. Once that is done, save the working copy of spellCheck.c in some other file (spellCheckLex.c for example) and then modify spellCheck.c to communicate with sort. Continue this process until the entire program is complete.
- Create a directory for this lab and place the C files (spellCheck.c, lex.c, and compare.c) and your input data file into this directory (do not include the dictionary). Have only these 4 files in the directory. Then issue the command

  ```
  tar cvf lab4.tar *
  ```

  This will create the file lab4.tar, and its contents are the archived c files and data file. Submit lab4.tar via Blackboard. Be sure to test that the files are properly tar'ed before submitting them. You can do this by copying (cp) your tar file to another directory and issuing the command

  ```
  tar xvf lab4.tar
  ```

- When writing this program be very sure that each of your programs behaves correctly. I have discovered that if lex.out does not close its files (including pipes) and then terminates normally that the entire system of programs will fail. This is true for each of your programs. Be sure to close all files and pipes when their use is completed, and terminate each program normally.
- The number one occurring error in logic in this type of program is not testing for words with different cases of letters. The letter 'Z' has a smaller ASCII and UNICODE value

than the letter 'a', and a simple string comparison will give the wrong results. Note that lex is not allowed to change case of letters in words, neither is compare. The words will be output exactly as input (except that if two input words differed in case, only one of them will be output). Note that the sort and uniq programs in linux have arguments which let them ignore case for the purposes of their comparisons.

- When calling exec to run sort and uniq, call a version of exec which allows the program name instead of a complete path name to sort or uniq. This means that sort and uniq will be discovered using the PATH environment variable. I am quite likely to have sort and/or uniq in a directory on my machine which is different from the path you enter in the exec command.

# Deliverables

Your source code files and input data file (4 files total, tar'ed into one submitted file) must be submitted via Blackboard in the file lab4.tar no later than 11:59:59 p.m. Friday May 10.