Lab 5 - CPSC 2311

Due: 2/20/23 at 11:59 pm

Overview:

This assignment is designed to provide practice with the following:

- ★ Linked List (sorting)
- ★ Function pointers

Read the entire document to ensure you are aware of all requirements.
This is an individual lab. You are not allowed to receive help from anyone other than the 2311 lab TAs.

There are many ways to complete this lab, but to ensure that you learn how to use several new functions/concepts in "C", I am going to be very specific on many of the instructions. I strongly suggest you read the entire document to ensure you understand what you are required to do. Points will be deducted for not following directions.

This lab will be built off of your linked list code from Lab 3. While you will not have to check for duplicate entries, your code should be able to read from an input file, store data in a linked list, and print output to a file. We will be adding several additional functions to Lab 3.

Requirements:

Function Pointers:
In class, you should have gone over function pointers. In this lab, you will be sorting a linked list either by first name or last name. You should only have one sortList function, but you will pass a function pointer to one of two compare functions to specify how sortList should sort the linked list.

Here are some additional resources about function pointers:
https://www.cprogramming.com/tutorial/function-pointers.html
https://www.geeksforgeeks.org/function-pointer-in-c/
https://youtu.be/axngwDJ79GY

Date Validation:
You will have to validate that the date stored in a node is valid. To do this, you will have to check that the day is not negative and also not higher than the number of days in the month. You will also need to check for leap years.

Sorting:
In this lab, you will take in an additional command line argument which will determine whether you should sort your list by first name or by last name. You can use any sorting algorithm (or make up your own!) to sort your list as long as you keep the function specs the same.

Files:

For this lab, you will check to make sure that a valid date is being read in from the input file. You are then going to sort the linked list (sort is determined by a command line argument). Your solution will consist of the following files:

driver.c

This file will have minimal code. Most of your code should be in your functions.c file. You will have to change the number of arguments you check for because you will be taking in an additional command line argument. In this file, you should keep the same code as in lab 3 but add:

- ★ Take in an additional command line argument, either "1" or "2". If ran with 1, then you should sort the list by first name. If 2, then sort by last name.
- ★ Call sortList to sort the linked list (different calls depending on last name or first name)

sortList should be called after createList, but before printList.

Your driver shouldn't have more code than necessary. DO NOT include functions.c in your driver.c file. As a rule, you should not include ".c" files. In this lab, doing so will screw up the autograder.

makefile

You are required to have a makefile for this lab to compile your code. You can have any targets that you like as long as the "make" command compiles your code (You can have other functionalities like "make run", "make test", or "make clean", but I will only be testing whether or not your code compiles). The autograder will run the "make" command, which should compile and link your files into an executable. Name your executable "LAB5". You will also have to use gcc as your compiler. Compiling with clang will not work for the autograder.

The **autograder** will call make and then attempt to run :

./LAB5 <autograderinput>.csv <studentout>.txt 1
OR
./LAB5 <autograderinput>.csv <studentout>.txt 2

If make does not compile your code or your executable is not named LAB5, you will lose points on the autograder. **File names are also case sensitive** so use “makefile” instead of “Makefile”. Do not include your executable file in your submission.

functions.h

This file will include the definitions for both of your typedef-ed structs. You will also have your function definitions in this file. Add the new functions you are implementing to this file. Header guards are required for this file.

You will also need to have a function pointer in this file. The functions pointed to by this function pointer should compare two nodes (take in two node*) and return an int:

0 -- if the nodes are equal
positive -- if the first node is "greater"
negative -- if the second node is "greater"

You will have two functions in functions.c that fit these specifications: compare_by_lastname() and compare_by_firstname().

functions.c

In addition to the functions from lab 3, you will have to implement the following functions:

bool checkInvalidDate(node_t *node);

This function should take in a node ptr and check that the node being pointed to contains a valid date. We will not give you incorrectly spelled months, but you must check that the day is in the month. To do this, you will also need to check for leap years (I did this in a helper function).

If a node has an invalid date, don’t add it to the list (or don’t print it out). However you chose to implement, it shouldn’t be in your output file.

void sortList(node_t **head, <function pointer> comp);

This function takes in two arguments. The first is the head of your linked list and the second is a function pointer to the compare function you want to use in your sort. This will be either compare_by_lastname() or compare_by_firstname().

You can sort your list however you like. If you’ve taken 2120, you may want to use a different sorting algorithm. However, exhaustively sorting will be the simplest method and we won’t test on input files too large for you to handle in the 10 minute autograder time out.

As far as algorithms go, I recommend using bubble sort for those totally new to sorting (here's a helpful guide: https://www.geeksforgeeks.org/bubble-sort/) and merge sort for those who have taken 2120 or want a challenge (https://www.geeksforgeeks.org/merge-sort/). Both of the links show how to use these sorts with arrays; linked lists follow similar logic.

Sorting your list is largely a choose your own adventure. As long as you keep the given functions the same, you can use any sorting algorithm and any helper functions you would like. I recommend adding a helper function called swapNode that takes in two node pointers (node*) and switches those two nodes in the linked list.

You should use the compare function that you passed in to compare nodes. ***You should not have a separate sort function for last name and for first name.*** You can use the same logic and just call a different compare function.

int compare_by_lastname(node_t *a, node_t *b);
This function should take in two node pointers and compare the two nodes by last name. You should return 0 if the strings are the same, a positive number if the first node is "greater", and a negative number if the second node is "greater".
HINT: read the man page for strcmp, especially the section on return values ;)

int compare_by_firstname(node_t *a, node_t *b);
This function should take in two node pointers and compare the two nodes by first name. You should return 0 if the strings are the same, a positive number if the first node is "greater", and a negative number if the second node is "greater".

The two compare functions should be virtually identical.

These functions are required but you are free to implement helper functions as you see fit. As long as the required functions (with the correct parameters and return values) are present, the implementation is flexible.

Make sure to remove all warnings from your code (compile with -Wall to view warnings). If your code has warnings when compiling, you will lose points.

Submission Information:

You will submit to gradescope. You should submit the following files. **DO NOT** tar your submission files. **DO NOT** submit a folder.

driver.c
functions.c
functions.h
makefile

NOTE: This week, the autograder will be checking for memory leaks (make sure your free function works). To test this, run valgrind on the SoC machines.

For any autograder-specific questions: email Eliza (esorber@clemson.edu). I usually respond faster to a Teams DM than an email.

You should have a header in each of your files that contains the following information. If you neglect to place a header in a file, you will receive a 5-point deduction for each missing header.

```
/*************************
 *Your name                *
 *CPSC 2310 Spring 23      *
 *UserName:                *
 *Instructor:  Dr. Yvon Feaster  *
 *************************/
```

- ★ Your code should be well documented. If you do not provide the appropriate documentation, a minimum of 5 points will be deducted for each un-documented function.
- ★ Your code should not contain “magic numbers”. For example:
  for(int i = 0; i <89; i++){}
  89 is a magic number; we don’t know what it represents. Use a global variable, local variable, or a #define for any magic numbers in your program.
- ★ There should be no line of code longer than 80 characters. There will be a minimum of a 5-point deduction for this violation. The more lines of code over 80 characters the higher the deductions. You must use proper and consistent indention. There will be a minimum of a 5-point deduction if your code is not neat with proper and consistent indention.

Failure to do any of the above items may result in a deduction of points for each offense.