Lab 6 - CPSC 2311

Due: 2/27/23 at 11:59 pm

Overview:

This assignment is designed to provide practice with the following:

- ★ Linked List
- ★ Structs

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 and Lab 5. You will not have to use your sorting or check duplicates implementation. You should be able to read in data from an input file and store it in a linked list.

In this lab, you will be iterating through a linked list and finding the oldest person in the list, the youngest person in the list, and the number of birthdays in each month of the year. You will store these stats in a struct.

Requirements:

Files:

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 can remove the calls to sort list used in Lab 5.

- ★ Create a instance of your list stats struct and set it using getListStats (continued in the functions.h section below)

- ★ Instead of calling your printList function, which would print the list to your passed in output file, you will instead call printStats, which will print the stats to that output file.

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. **Do not have your "make" command run or clean your code.** Name your executable "LAB6". 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 :

./LAB6 <autograderinput>.csv <studentout>.txt

If make does not compile your code or your executable is not named LAB6, 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 from Lab 3. You will also have your function definitions in this file. Add the new function definitions and list_stats_t struct you are implementing to this file. Header guards are required for this file.

In this file, you will be adding a new struct. You should use typedef for this struct. I typedefed my struct as <list_stats_t>, but you can call yours something else if you want. This struct should include the following data members:

- ★ A node pointer pointing at the oldest person in the list
- ★ A node pointer pointing at the youngest person in the list
- ★ An int array holding the number of birthdays in each month (your array should have 12 elements where 0 is January and 11 is December)

functions.c

In addition to the functions from lab 3 and lab 5, you will have to implement the following functions:

list_stats_t getListStats(node_t *head);
This function will take in a pointer to the head of your list and return a populated list_stats struct.

This function should iterate through the linked list and find the oldest person in the list, the youngest person in the list, and the number of birthdays per month. All months will be spelled correctly and capitalized (January, February, March **not** january, february, march).

It is possible to get all these stats with one iteration of the list, however if you want to implement three separate helper functions, one for finding the oldest, one for finding the youngest, and one for counting the number of birthdays, that works as well. Just have your getListStats function call those helpers.

I recommend writing a compareBirthdays helper function that will take in two birthdays and determine which one is older and which is younger. This will help you keep getListStats from becoming too large and overwhelming.

void printStats(FILE* , list_stats_t *;
This function will take in two parameters, a file pointer pointing at the output file to print to and a pointer to the struct containing your stats.

This function should print out your results like the following:

```
********************************************************************************
List Statistics:
----------Oldest Person----------
Name: John Smith
Date of Birth:  May 11, 1993
Major:  CS/BA
Year:   Senior

----------Youngest Person----------
Name: Mary Baker
Date of Birth:  January 30, 2004
Major:  CS/BA
Year:   Freshman
```

----------Birthday Counts----------
January: 0
February: 4
March: 3
April: 2
May: 2
June: 2
July: 0
August: 3
September: 5
October: 1
November: 1
December: 0
****

Or if you prefer the more pseudocode way of showing it:

printBorder()
<ten dashes>Oldest Person<ten dashes>
Name:\t%s %s\n
Date of Birth:\t%s %d, %d\n
Major:\t%s\n
Year:\t%s\n\n
<ten dashes>Youngest Person<ten dashes>
Name:\t%s %s\n
Date of Birth:\t%s %d, %d\n
Major:\t%s\n
Year:\t%s\n\n
<ten dashes>Birthday Counts<ten dashes>
January: #
February: #
March: #
April: #
May: #
June: #
July: #
August: #
September: #

October: #
November: #
December: #
printBorder()

If the list is empty, print out “None\n\n”.

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      *
 *Email:                   *
 *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.