Lab 3 - CPSC 2311

Due: 2/6/23 at 11:59 pm

Overview:

This assignment is designed to provide practice with the following:

- ★ Linked List
- ★ Reading comma-delimited files using scansets
- ★ 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.

Requirements:

Scanset conversion:

In class, you have looked at how to read and write information using scanf/printf or fscanf/fprintf. For this assignment, you will read data from a comma-delimited file using the concept of scanset conversion. In class, we discussed examples of scanset conversion and you are required to use scanset conversion to read ALL of the data. Below are several links that may help you understand scanset:

https://www.geeksforgeeks.org/scansets-in-c/
https://www.knowprogram.com/c-programming/read-and-display-the-string-in-c-programming/
http://www.cplusplus.com/reference/cstdio/fscanf/
https://www.tutorialspoint.com/scansets-in-c

So, what is the data that you will read? In an email, I asked you to complete a Google form with questions about yourself. The form asked for your first and last name, birthday, major, and class standing. This is the data you will read from a file. I will give you an input file

(Lab6Input.csv) to test your code. The autograder will test your code on additional input files. You will have to think of corner cases and test your code.

Structs:
Each set of data will be stored in a linked list. You must use two structs; one for the birthday and one that will be used as a node for the linked list. You must typedef both of these structs.

Linked List:

You all should have seen linked lists in previous classes. If you need a refresher, here's a youtube tutorial: https://youtu.be/VOpjAHCee7c

You will define a struct for the linked list that will have the following data members:

- ★ Character arrays for the first name(size 50), last name(size 50), major(size 50), and class standing (size 50).
- ★ An instance of the struct that represents the birthday
- ★ A pointer keeping track of the next node in the list

Files:

For this lab, you will read in data from an input file using scanset, store that data into a linked list, and check for duplicate entries. 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. In this file, you should

- ★ Use assert to check for the correct number of command line arguments and that the files opened correctly
- ★ Open your input and output files
- ★ Call createList to create and fill your linked list with input data
- ★ Call printList to print out the linked list
- ★ Free memory from the linked list by calling deleteList

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 “LAB3”. 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 :

./LAB3 <autograderinput>.csv <studentout>.txt

If make does not compile your code or your executable is not named LAB3, 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. The specs for the node struct are above. The struct storing birthday information should contain ints for the day and year and a char array for the month.
You will also have your function definitions in this file. Header guards are required for this file.

functions.c

This file will implement the following functions:

void add(node_t **node, node_t **head);
This function will add a node to a linked list. You will take in two parameters, a double pointer to the node you want to add and a double pointer to the head of the list. You should check if the list is empty and add the node to the list. You will have to print out the data in order so you may want to add the node to the end of your linked list.

node_t* readNodeInfo(FILE* input);
This function will read the data from the input file, returning a pointer to the populated node. Use malloc to allocate the memory for the node that will eventually be added to the linked list. Using scanset conversion, read the data and store it in the node allocated. (You must use scanset conversion to read **ALL** the data, not just part of the data.)

node_t* createList(FILE*, node_t);**

This function will be called in your driver and will create the linked list. The first argument will be a file pointer to your input file and the second will be a double pointer to the head of your list. Use a loop to read from your input file, calling **readNodeInfo** for each node and then calling **add** to add that node to the list. After all of the information from the input file has been added to the list, return a pointer to the head of the list.

void printList(FILE*, node_t*);
This function will print out your linked list. The first argument is a file pointer to your output file and the second is a pointer to the head of your list. If the list is empty, print "LIST IS EMPTY\n" to stderr. Else, your output should look like the following:
****

List Info:
Name: Eliza Sorber
Date of Birth: March 12, 2002
Major: CS/BS
Year: Junior

Name: Logan Baker
Date of Birth: February 8, 2003
Major: CS/BA
Year: Sophomore

****

AKA:
printBorder()
\n
List Info:\n
Name:\t<firstname> <lastname>\n
Date of Birth:\t<month> <day>, <year>\n
Major:\t<major>\n
Year:\t<class standing>\n
\n
printBorder()

There is exactly one tab character between each of the labels and the data and a single space in between data such as first and last names. The autograder will strip extraneous new lines so

you will not lose points for having extra new line characters, however, the rest of your output should match exactly.

You need to print out the data in the order that you read it in. You will lose points if you print out the data backward.

void printBorder(FILE*);
This function prints out 80 stars. I've implemented this function for you.

void deleteList(node_t);**
This function will deallocate the memory used in your linked list to avoid any memory leaks. You will take in one parameter, a double pointer to the head of your linked list.

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.

You will also need to check for duplicate entries. If there are two identical entries in your input file, you should only print out the first one. There are a few different ways to do this. The simplest method is to exhaustively search your linked list and check to see if there are any matching entries. You can do this check when adding to the linked list or when printing to the output file. If you've taken 2120, you could also use a data structure like a hash table or a trie to keep track of duplicates (this is **not required**, just a fun option). You may implement any additional functions to perform the check.

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

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 function that is not documented.
- ★ 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.