AU 2016

CSE 2421

LAB 3

Due Date: Monday, October 3 by 11:30:00 p.m. **NO LATE LABS WILL BE ACCEPTED FOR THIS LAB!**

IMPORTANT NOTE: READ THE INSTRUCTIONS HERE CAREFULLY, AND FOLLOW THEM. ALSO REFER TO THE CLASS SLIDES ON STRUCTURES, LINKED LISTS, AND FUNCTION POINTERS. IF YOU DO, YOU CAN FINISH THIS LAB IN MUCH LESS TIME. IF YOU DO NOT, YOU WILL SPEND MUCH MORE TIME ON THE LAB, AND YOU MAY STILL NOT FINISH ON TIME!

Objectives:

- Structures and Structure Pointers
- Linked Lists
- Function Pointers

REMINDERS and GRADING CRITERIA:

- You are allowed to work in pairs for this assignment, and you are encouraged to, but you are not required to do so.

- Every lab requires a Readme file (for this lab, it must be called lab3Readme – the name is important, and case matters! DO NOT use any "extension" on the file name; Linux can identify the file type without any extension). This file should include the following:

-Student name(s) - to avoid the 10% deduction (as explained in the course syllabus) if working with a partner;

-Effort distribution for each contributor (assumed to be 100% if you are not working with a partner);

-Total amount of time to complete the entire lab;

-Short description of concerns, interesting problems or discoveries encountered, or comments about the contents of the lab. Do not omit this or write something very general, or you will lose points.

– You should aim to always hand an assignment in on time. If you are late (even by a minute - or heaven forbid, less than a minute late), based on the due date given above, you will receive no credit for this lab. NO LATE LABS WILL BE ACCEPTED for this assignment, because code will be posted right after the lab is due to help students study for the midterm exam. I encourage you strongly to start early – no more than a few days after the lab is assigned. The most common reason for failing to submit on time is waiting too long to start!

– Make yourself aware of the Coding Style in C file posted on: https://piazza.com/osu/autumn2016/cse2421/resources, under Labs. Be sure to follow the coding style information in that document, as the grader can, *and usually will*, deduct points for code that fails to follow the style requirements there.

– Any lab submitted that does not compile and run WILL RECEIVE AN AUTOMATIC GRADE OF ZERO.

No exceptions will be made for this rule - to achieve even a single point on a lab, your code must minimally build (compile to an executable, using the command shown below) on stdlinux and execute on stdlinux without crashing, freezing, or causing segmentation faults. You MUST use the following commands to build your source code to an executable (the grader will use these commands):

For part 1: %gcc -ansi -pedantic lab3p1.c -o lab3p1
For part 2: %gcc -ansi -pedantic lab3p2.c -o lab3p2

– You are welcome to do more than what is required by the assignment as long as it is clear what you are doing and it does not interfere with the mandatory requirements.

LAB DESCRIPTION

PART 1. CLASSIC BOOKSTORE INVENTORY SYSTEM (Approximately 75%): Mandatory file name: lab3p1.c (This is the required file name. Do not change it!)

For this part of the lab, you will write a program to store and process data in order to manage inventory and to keep business records for a bookstore which deals in classic book titles. The user will enter data about the book titles to be stored in the inventory system, and will select options related to what to do with the data. You do not know the number of different book titles which will be entered by the user, so your code must be able to deal with an indeterminate number of different books (up to the limits of memory).

First, your program should prompt the user to enter data about the books in the initial inventory (There will be options for adding or deleting books provided to the user later by the program after the initial data is read from input and used to build the linked list with the initial inventory). The data will be entered in the following format, with each item of data listed entered on a different line:

Book title (We will assume the title may possibly contain white spaces; see the sample data posted on Piazza)

Author's name (We will assume there will be a single author for each book, with the name possibly containing white spaces; see the sample data posted on Piazza)

Book stock number (1 - 10000)

Wholesale price in dollars (a floating point value)

Retail price in dollars (a floating point value)

Wholesale quantity purchased by bookstore (a whole number greater than or equal to zero)

Retail quantity purchased by customers (a whole number greater than or equal to zero)

Data about each book will not be separated from the data for the following book; that is, the first line of input for the next book will immediately follow the book which precedes it on the following line. The end of the input for the books will be marked by the string END_DATA, which will appear on the next line after the end of the data for the last book. There will be data about at least one book in the input. Please see the sample input data posted in a text file on Piazza. You should copy this sample data into a file in your lab 3 directory and use it to test your program using redirection of the input, as described in

class, and in the directions for lab 2. The linked list which your program builds for the books should store them in order of increasing book stock number, but the books will not necessarily appear in the input in this order. After reading the input data about the books, and constructing the linked list, your program should prompt the user to select from the following options for processing of the data (Write a separate function to prompt the user, get a user choice, and call the appropriate function):

1. Determine and print total revenue (from all books): This is the sum of (retail price * retail quantity) for all books;
2. Determine and print total wholesale cost (for all books):This is the sum of (wholesale price * wholesale quantity) for all books;
3. Determine and print total profit (for all books): This is total revenue (above) minus total wholesale cost (above);
4. Determine and print total number of sales (total number of books sold): This is the sum of the retail quantities for all books;
5. Determine and print average profit per sale: This is total profit (above) divided by total number of sales (above);
6. Print book list: This function should print “Book list:” on one line, and then the names of all books, each on a separate line;
7. Add a single book (prompt the user to enter the book data for *a single book*, in the format given above, but there will be no END_DATA string after the book input);
8. Delete book (prompt the user to enter the book stock number: if the book is not found in the linked list, print a message indicating an error, or if the list is empty, print an error indicating that);
9. Exit the program

The user will enter a choice of one of these nine options by entering a single digit, 1 - 9, immediately followed by enter (new line). You can assume that all user input will be correct, except that the user may inadvertently attempt to delete a book which has not been added to the list of books. You should write a separate function to do the necessary computations and print output for each of these options. Some of these functions may call some of the other functions. The goal is to make main() as small and succinct as possible, while creating functions that make it as easy as possible to monitor, modify and execute the code. You should also write a separate function to read the data for a single book into a node after allocating storage for the node.

Be sure, for each of the functions above which is required to print output, to print a label which describes the output, along with the appropriate output on the same line (except for the function to print the book list, where all the titles should be printed on separate lines).

GUIDANCE

- Declare the struct node type at file scope (before main), as follows:

```
struct Data {
        char title[40];
        char author[40];
        int stockNumber;
        float wholesalePrice;
        float retailPrice;
        int wholesaleQuantity;
        int retailQuantity;
};
typedef struct Node {
        struct Data book;
        struct Node *next;
} Node;
```

- Code is posted in a text file on Piazza for a function to allocate memory for a node, and then call a function to read data for one node from input, and store it in the memory allocated for the node.
- You should write and debug the following functions first:
  - main
  - a function to print the nodes in the list; see, and carefully follow, the sketch of the algorithm in the class slides on linked lists.
  - a function insert, to add a node to the list; see, and carefully follow, the sketch of the algorithm in the class slides.
- DO NOT WRITE THE CODE FOR OTHER FUNCTIONS UNTIL THE FUNCTIONS ABOVE ARE WORKING!!!
- Until you are ready to write the code for the other functions, you can write functions with empty parameters and empty blocks for the other functions in the program; if these functions are called in the program, they will do nothing, so this will create no problems for testing and debugging.
- After the functions above are working, write a function delete, to delete a single node from the list; see, and carefully follow, the sketch of the algorithm in the class slides.
- Then, write the remaining functions (they are similar to the function to print the books in the list).

CONSTRAINTS

– The book data must be stored in a singly-linked list, with each node in the list containing the data for one book title.

– You are not permitted to declare any variables at file scope; you should, however, declare the Node type used to store the data for the linked list at file scope (but no

variables can be declared as part of this declaration). See the description of the declaration of the Node type above.

– All floating point data will be entered with two digits of precision, and should also be output with two digits of precision.

– You must allocate the storage for each book node structure dynamically.

– The book name and author's name should each be stored in strings declared to be of type ***array of char*** of size 40 (See the description of the declaration of the Node type above; these members cannot be declared to be of type char *, or the code becomes more complex!).

– If a book is deleted, you should call free() to free the storage for the node for the book.

– You do not need to free the space for the individual nodes before the program terminates. Although usually we would write a separate function to do this which can be called after the user selects option 9, to reduce the amount of work for the lab, we will omit this here (but it would usually be done in production environments).

See the text file posted on Piazza with sample input for Part 1.

PART 2. USE AN ARRAY OF FUNCTION POINTERS TO CALL THE USER'S CHOICE OF FUNCTION (Approximately 25%): Mandatory filename: lab3p2.c (Do not change this name!)

This program should prompt the user to enter a choice of one of four numbered functions, and output a message which prints the number of the function the user chose, unless the user chooses function 4, which should terminate the program. For example, if the user chooses function 1, the program should print:

You chose function 1!

The output for functions 2 or 3 should be the same, except for the difference in the number of the function. For function 4, no output should be produced, but the program should terminate.

Be sure to error catch the following:

– Incorrect function number (i.e. not 1, 2, 3, or 4): If the user enters a choice of function number other than 1, 2, 3, or 4, print a message indicating the incorrect choice:

Your choice of function number is not one of the possible options. Choose 1, 2, 3 or 4!

CONSTRAINTS

– Be sure to output your result to the screen with messages like the ones given above.

– You should declare an array of function pointers, and you MUST call the function for each of the four choices with one of the function pointers in this array.

– The program should keep running and repeatedly prompting the user for a choice of function until the user chooses function 4.

See the instructions on submission of the lab on the next page.

LAB SUBMISSION

You should submit all your lab assignments electronically using the submit command. Be sure that all the files you are submitting are in the directory from which you run submit (you should create a lab3 directory, and put all files for this lab there)!!!!! Here are the submit commands for the lab3 assignment:

9:10 %submit c2421ab lab3 lab3Readme lab3p1.c lab3p2.c

10:20 %submit c2421ad lab3 lab3Readme lab3p1.c lab3p2.c

1:50 %submit c2421aa lab3 lab3Readme lab3p1.c lab3p2.c

Reminders:

– You must submit all required files with a single submit command; if the line for the command wraps, allow it to do so. Do not hit enter/return in the middle of the command!

– The files that will be received by the teaching staff will be all the files you submitted ***the last time you run submit***; files from any earlier submission are deleted every time submit is run. Therefore, if you resubmit, ***be sure to submit all the required files***!

For more information about the submit command, review that section in your prelab practice problem.

Do not submit executable versions of your program, but only source code files and the Readme file!