AU 2018 CSE 2421 LAB 3

Due: Sunday, October 7th, by 11:30 p.m.

NOTE: NO LATE SUBMISSIONS will be accepted for this lab! If you do not submit by the deadline given above, your lab will not be accepted, and will not be graded.

IMPORTANT NOTE: READ THESE INSTRUCTIONS AND FOLLOW THEM CAREFULLY. ALSO, REFER TO THE CLASS SLIDES ON scanf() FORMAT OPTIONS, 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! YOU HAVE BEEN GIVEN 10 PLUS DAYS TO COMPLETE THIS LAB; IT WILL LIKELY TAKE SOME OF YOU EVERY BIT OF THAT TIME TO COMPLETE. DON'T PROCRASTINATE!

Objectives:

- Structures and Structure Pointers
- Strings
- Linked Lists
- Function Pointers
- Passing Command Line Arguments to main()
- Header files and Make files

REMINDERS and GRADING CRITERIA:

➢ This is an individual lab. No partners are permitted

➢ **Every lab requires a Readme file** (for this lab, it should be called **lab3Readme – use this name, without an extension or any other modification**). This file should include the following:
- Your name
- Total amount of time (effort) it took for you to complete the lab
- Short description of any concerns, interesting problems or discoveries encountered, or comments in general about the contents of the lab
- Describe how you used gdb to find a bug in your program while debugging it. Even if you did not use gdb, you still need to describe how you could use it to find a bug; this is required! Include how you set breakpoints, variables you printed out, what values they had, what you found that enabled you to fix the bug. If you didn't have to use gdb to find any bugs, then use gdb to print out the value of each address you received from a malloc()/calloc() call, then use a breakpoint for the free() system call to verify that each addressed is passed to free at the end of the program.

➢ You should aim to always hand an assignment in on time (or early!). If you are late (even by a minute – or heaven forbid, less than a minute late), ***your lab will not be accepted, and will not be graded.***

➢ Any lab submitted that does not compile/make – ***without errors*** - 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) on stdlinux and execute on stdlinux without crashing, using the following command for Part1:

```
% gcc -ansi -pedantic lab3p1.c -o lab3p1
```

And the following command for Part2

```
% make
```

In addition, if your code generates warnings from the compiler for either part, you will lose points for each warning, depending on the warning.

LAB DESCRIPTION

PART 1. BOOKSTORE INVENTORY SYSTEM:

Mandatory file name: lab3p1.c

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 (mostly) classic book titles.

The program will execute in the following format:

```
$ lab3 < filename1
```

where lab3 is the name of the executable, and filename1 is the input file that contains input to the program.

You will be supplied with a “test” input file named **lab3p1_input**. All of the data for the book titles to be stored in the inventory system will be in this file. Once the program has read in the book data, a user will be able to select options related to what to do with the data. You do not know the number of different book titles which will be in the file, so your code must be able to deal with an indeterminate number of different books (up to the limits of memory).

NOTE: You need to modify the input file to test the functions that your program is required to implement (described below).

First, your program should read input from the input file specified by the filename1 input file and read 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 in and used to build the linked list. 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; check the initial **lab3p1_input** input file on Carmen)
- Author's name (We will assume there will be a single author **line** for each book, with the name(s) possibly containing white spaces; see the sample input data in the input file **lab3p1_input** posted on Carmen)
- Book stock number (1 - 10000)
- Wholesale price of the book in dollars (a floating point value)

- Retail price of the book 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 indicated with the string END_DATA. There will be data about at least one book in the file. The linked list that your program builds for the books should store them in order of increasing book stock number; the books will not necessarily appear in the input in this order. Also, the linked list must be built in the way described in the class slides; specifically:

-The location of the "head node", or first node, must be stored in a pointer variable, and NOT in a structure.

-The next member of the last node must contain NULL, or, if the list is empty, the head pointer variable must be NULL.

-The structure for the nodes must be declared as shown below under GUIDANCE.

After reading the input data about the books, and constructing the linked list, your program should

a) Tell the user that the inventory of books has been read in from the file and how many book titles were read in.

b) Prompt the user to select from the following options for processing of the data. (**REQUIRED: Use a switch-case statement to call the User's Choice of Function for options 1 through 11).**

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 the current investment in books: This is the sum of (wholesale price * (wholesale quantity – retail quantity));
4. Determine and print total profit (for all books): This is total revenue (#1) minus total wholesale cost(#2) plus cost of current inventory (#3);
5. Determine and print total number of sales (total number of books sold): This is the sum of the retail quantities for all books;
6. Determine and print average profit per sale: This is total profit (#4) divided by total number of sales (#5);
7. Print books in stock: This function should print each book on "Book list" where (Wholesale quantity – Retail quantity >0). The output would be a "Title Line" that says something like "Books in Stock" then on each of the subsequent lines, for each book in stock, the number of books currently in stock (Wholesale – Retail) followed by a tab character then the book stock number followed by a tab character, then the author of the book followed by the tab character then

the title of the book;

8. Print books out of stock: This function should print each book on “Book list” where (Wholesale quantity – Retail quantity == 0). The output would be a “Title Line” that says something like “Books Out of Stock” then on each of the subsequent lines, for each book in stock, the book stock number followed by a tab character, then the author of the book followed by the tab character then the title of the book;
9. Add a single book (prompt the user to enter the book data for *a single book*, in the format given above from the keyboard (Note that there will be no END_DATA string marking the end of the book data);
10. Delete book (prompt the user to enter a book stock number to delete from inventory: 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 indicatingthat);
11. Exit the program. (This option would write the current inventory in the linked list out to disk using filename2 from the command line and would also free all dynamically allocated memory.)

The user will enter a choice of one of these eleven options by entering 1 - 11 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 from stdin 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 books in stock list or books out of stock list, where each title/author pair should be printed on separate line).

GUIDANCE

- Declare the struct Node shown below at file scope:

```
struct Data {
        char title[50];
        char author[50];
        int stockNumber;
        float wholesalePrice;
        float retailPrice;
        int wholesaleQuantity;
        int retailQuantity;
};

typedef struct Node {
        struct Data book;
        struct Node *next;
} Node;
```

- You should write and debug the following functions first:

- ✓ main (adding items as needed)
- ✓ the function to read in the inventory data from the input file
- ✓ the functions to print the nodes in the list (i.e. in-stock list and out-of-stock list); find 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; find and carefully follow the sketch of the algorithm in the class slides on Carmen.

- ➢ DO NOT WRITE THE CODE FOR OTHER FUNCTIONS UNTIL THE FUNCTIONS ABOVE ARE WORKING!!!

- ➢ If the output for either print statement is empty (e.g. the print out-of-stock list has no books to print, you should print a statement to that effect. Similarly, if you are asked to delete a book with a stock number that is not found in the list, print a statement that says that.

- ➢ Until you are ready to write the code for the other functions, you can write "stub" 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. For example, a stub function for delete_node() might look like this:

  ```
  void delete_node(){
  }
  ```

  You can change the return value and the number and type of parameters passed when you have a better idea of what the function should have later in your development cycle.

- ➢ After the functions above are working, write a function called delete, to delete a single node from the list; find 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 in or out of stock books in the list.)

CONSTRAINTS

- The book data must be stored in a singly-linked list, where each node in the list contains the data for one booktitle. The head of the list cannot be a NODE, it must be a list head (e.g.Node *)
- 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 ofprecision and using monetary symbols where appropriate.
- 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*** both of size 50. (See the description of the declaration of the Node type above.)
- If a book is deleted, you should call free() to free the storage for the node for the book.
- You must free the space for the individual nodes that still contain data before the program terminates when the user selects option 11.

- See the text file **lab3p1_input** posted on Carmen in the Labs folder with sample input for Part 1; you can start a firefox browser in stdlinux to download this file using the recommended command below:

  ```
  $ firefox https://carmen.osu.edu/#
  ```

  (If a different way of starting firefox has worked better in the past, use that command).

PART 2. Use command line parameters, function pointers and C library string functions, and create a Makefile

Mandatory file names: lab3p2.h lab3p2.c lab3p2f1.c lab3p2f2.c lab3p2f3.c Makefile

This program should execute the following command entered at the command line:

```
% lab3p2  func_type  string1  string2
```

where the three possible func_types are:
func_type == 1: Concatenate string1 with string2, separated by a single blank (white space)and print the concatenated string.
E.g., given the command line entry: % lab3p2 1 Go Bucks!
The output should be: Go Bucks!
func_type == 2: Determine if string1 and string2 are the same string (case matters).
E.g., given the command line entry: % lab3p2 window winnow
The output should be: string1 is not the same as string2
func_type == 3: Convert all the uppercase characters in string1 to lowercase, and vice-versa.
E.g., given the command line entry: % lab3p2 3 aWESOME! gREat
The output should be: string1 with inverted case is: Awesome!

Be sure to error-catch the following, and print an appropriate error message:
– Incorrect func_type number (i.e., not 1, 2, or 3).
– An invalid number of command line arguments (that is, greater than 3 or less than 3).

CONSTRAINTS
– You are required to write your own functions, but you can (and should) use string functions in the C library instead of writing your own code to perform functionality which they provide).
– Be sure to write output to the screen with messages similar to the ones given above.
– You should declare an array of function pointers, and use the array to call the appropriate function (1, 2, or 3) depending on what the user enters on the command line).
– Similar to the example in class, put each of the functions into its own .c file (see the required file names ot submit, and you should be able to determine which functions to put into which files), and create a lab3p2.h header file with function declarations. Finally, create a Makefile with dependencies, multiple tags, and a clean option, and make sure that your program builds successfully with make.

LAB SUBMISSION

LAB SUBMISSION

- You should submit all your lab assignments electronically to Carmen by starting a firefox web browser in stdlinux. You can start a firefox browser in stdlinux to upload your files using the recommended command below:

  ```
  $ firefox https://carmen.osu.edu/#
  ```

  (If a different way of starting firefox has worked better in the past, use that command).

  After starting firefox, go to Assignments, then Lab3, and upload your files. You can put all of your files into a zip (.zip) before uploading them. Be very sure to submit all of the mandatory files for Part 1, and also all the mandatory files for Part 2, and also your Readme file; be sure all of these files have the required names (any changes made by Carmen will not be penalized, though).

Reminders:

- You must submit all required files with a single upload; if you upload again later, Carmen will change your file names by adding a number, but you will not be penalized for any change in the file names made by Carmen!
- **Do not submit executable versions of your program**; only submit source code files, header files, your Makefile and the Readme file!