AU 2020 CSE 2421 LAB 4

Assigned: Saturday, September 26th
Early Due Date: Tuesday, October 5th by noon
Due: Wednesday, October 6th , by 11:30 p.m.

IMPORTANT NOTE: YOU HAVE BEEN GIVEN 10 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 and C library string functions
- Linked Lists
- Function Pointers
- Passing Command Line Arguments into main()
- Header files and Make files

REMINDERS and GRADING CRITERIA:

- This is an individual lab.
- Not starting to work on this lab immediately would be an unbelievably bad idea.

- **Effort has been made to ensure that this lab description is complete and consistent. That does not mean that you will not find errors or inconsistency. If you do, please ask for clarification.**

- **Every lab requires a Readme file** (for this lab, it must be called **lab4Readme – use this name.** This file must include the following:

  · Required Header:

  BY SUBMITTING THIS FILE TO CARMEN, I CERTIFY THAT I STRICTLY ADHERED TO THE TENURES OF THE OHIO STATE UNIVERSITY'S ACADEMIC INTEGRITY POLICY.
  THIS IS THE README FILE FOR LAB 4.

  · 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, with 4-5 sentences, how you used gdb to find a bug in your program while debugging it. Or, if you had no bugs in your program, how you used gdb to verify that your program was working correctly. 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 may use ddd instead of gdb, if you wish.

- 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), you will receive 75% of your earned points for the designated grade as long as the assignment is submitted by 11:30 pm the following day, based on the due date given above. If you are more than 24 hours late, you will receive a zero for the assignment and your assignment will not be graded at all.

- Any lab submitted that does not compile – without errors or warnings - 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 without errors or warnings) on stdlinux and execute on stdlinux without crashing, the gcc command must use the **–ansi** and **-pedantic** options.

Since a Makefile is required for this lab, you must create the appropriate compile statements to create a .o file for every .c file you create, which would then create a lab4 executable. Graders will be downloading your lab4.zip file from Carmen, unzipping it, and then executing **make** from a linux command line prompt. Your program must compile –without errors or warnings – via commands within the Makefile. Note that the graders will be checking your Makefile to verify that you are using -ansi -pedantic with your compile statements. **Given valid input, your program must also run without having a seg fault or other abnormal termination.**

- You are welcome to do more than what is required by the assignment if it is clear what you are doing, and it does not interfere with the mandatory requirements. Obviously, a "buy item" option could be implemented specifying retail price received per item and how many items were purchased which would modify inventory info for a particular stock number (make sure you take care not to purchase more items than are in stock). Or, an option could be implemented that would indicated an inventory item had been replenished including how many items have been brought in to inventory of a particular stock number. If there is a price (either wholesale or retail) change, then the item would have a different stock number. Or, additional option could be implemented such that options 1-6 below gave data for a particular department rather than the whole store. Any new options must use the function pointer array unless they could potentially change list head. If you choose to implement any of these "suggestions" or another of your own design AND THEY WORK CORRECTLY, you will receive 10 points in the extra credit category shown in Carmen for each option implemented. If you choose to do any of these or another additional option of your own (approved in writing from your instructor), please put additional information into your lab4Readme file describing what you have implemented so that the graders will know to check it. No more than 30 extra credit points are available for this lab. This is over and above the early deadline bonus.

LAB DESCRIPTION

PART 1. GROCERY STORE INVENTORY SYSTEM:

Mandatory filenames: 1) **lab4main.c** – will hold main()
2) **lab4.h** – will hold any local typedefs or struct definitions, function prototypes, etc. NO VARIABLES can be declared here.
3)) you must have an additional file name for **every function that you write**. (e.g. insert_node.c, delete_node.c, option1.c, option2.c, etc.)

Mandatory executable name: lab4

PROBLEM:
You will write a program to store and process data in order to manage inventory and to keep business records for a grocery store.

The program will execute in the following format:

lab4 filename1 filename2

where lab4 is the name of the executable
filename1 is the data file that contains current inventory information to read in
filename2 is a data file that your program will create with the output inventory

This means that you will have to read two parameters from the command line. The two filename parameters can be the same filename, so you will have to close filename1 before opening filename2 to ensure that you do not end up with corrupted data.

You will be supplied with an initial inventory file called **store_inventory**. All of the data for the grocery items to be stored in the inventory system will be in this file. Once the program reads in the 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 grocery items which will be in the file, so your code must be able to deal with an indeterminate number of them (up to the limits of memory).

First, your program must open the file specified by the filename1 parameter and read in the initial inventory. There will be options for adding or deleting grocery items 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:

- Grocery Item (Assume there will be a single **line** possibly containing white spaces; check the initial **store_inventory** file on Piazza)
- Grocery Store Department (Assume there will be a single **line** possibly containing white spaces; see the sample data posted on Piazza)

- stock number (1 - 10000)

- Wholesale price of the item in dollars (a floating pointvalue)
- Retail price of the item in dollars (a floating point value)
- Wholesale quantity purchased by this store (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 grocery item will not be separated from the data for the following item; that is, the first line of input for the next grocery item will immediately follow the item which precedes it on the following line. The end of the input for the items will be indicated when an fscanf() function returns EOF. There will be data about at least one grocery item in the file. The linked list that your program builds for the store must create a linked list based on increasing stock number; the grocery items will not necessarily appear in the input in this order. You program **MUST** order all items as it inserts them into the linked list. [This means that you **CANNOT** read all items in with no concern for order, then, subsequently, order the linked list. **If you choose to do this, you will receive 0 points for the lab**.]

After reading the input data, and constructing the linked list, your program must:

a) Tell the user that the inventory has been read in from the file and how many grocery item records were read in.

b) Prompt the user to select from the following options for processing data. (**REQUIRED:**

Use an Array of Function Pointers to call the User's Choice of Function for options 1 through 9.

1. Determine and print total revenue (from all grocery items sold): This is calculated as the sum of (retail price * retail quantity) for all grocery items;

2. Determine and print total wholesale cost (for all grocery items): This is calculated as the sum of (wholesale price * wholesale quantity) for all grocery items;

3. Determine and print the current investment in grocery items: This is calculated as the sum of (wholesale price * (wholesale quantity – retail quantity));

4. Determine and print total profit (for all grocery items): 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 grocery items sold): This is calculated as the sum of the retail quantities for all grocery items;

6. Determine and print average profit per sale: This is total profit (#4) divided by total number of sales (#5);

7. Print grocery items in stock: This function must print each grocery item on a "Inventory List" where (Wholesale quantity – Retail quantity >0). The output would be a "Title Line" that says something like "Grocery items in Stock" then on subsequent lines, the number of grocery items currently in stock (Wholesale – Retail) followed by a tab character then the stock number followed by a tab character, then the Grocery store department followed by the tab character then the name of the grocery item;

8. Print grocery items out of stock: This function must print each item on the inventory list where (Wholesale quantity – Retail quantity == 0). The output would be a "Title Line" that says something like "Grocery items out of Stock" then on subsequent lines the stock number followed by a tab character, then the Grocery store department followed by the tab character then the name of the grocery item;

9. Print all grocery items from a specific department. After prompt for a (sub)string of a particular Department Name, this function must print each item on the inventory list where the Department Name contains that string. For example, if the Department Name is "DAIRY DEPARTMENT" and the substring read is "DAIRY" or "dairy", then all "DAIRY DEPARTMENT" inventory items must be printed out with appropriate headings along with the current number of items in stock.

10. Add a new grocery item (prompt the user to enter the data for *a single grocery item*, in the format given above from the keyboard;

11. Delete grocery item (prompt the user to enter a stock number to delete from inventory: if the item is not found in the linked list, print a message indicating an error, or if the list is empty, print an error indicatingthat);

12. Write the current inventory in the linked list out to disk using filename2 from the command line, free all dynamically allocated memory and exit the program. (The format of the file written to didsk must be such that it can be used as input to the program the next time it runs.)

The user will enter a choice of one of these twelve options by entering 1 - 12 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 grocery stock ID number which is not in the list of grocery items. You must write a separate function to do the necessary computations and print output for each of

these options. Some functions should call other functions you've already written. 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 must write a separate function to read the data for a single grocery item 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 grocery items in stock list or grocery items out of stock list, where each pair must be printed on separate line).

REQUIREMENTS

- All function prototypes must be included in your lab4.h file.
- Declare the structs shown below in your lab4.h file:

```
struct Cost {
        float wholesalePrice;
        float retailPrice;
        int wholesaleQuantity;
        int retailQuantity;
};
struct Data {
        char item[50];
        char department[30];
        int stockNumber;
        struct Cost pricing;
};

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


- **You can assume that there will be no grocery item duplicates in the inventory file you read from disk. You cannot assume that a user won't "fat-finger" a stock number that is already in use when trying to add a new item.**
- **You MUST write and debug the following functions first:**
  - ✓ main (adding items as needed)
  - ✓ the program to read in the inventory data from disk
  - ✓ 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 from the classslides.
- DO NOT WRITE THE CODE FOR OTHER FUNCTIONS UNTIL THE FUNCTIONS ABOVE ARE WORKING!!!
- **Create functions that do work for you that is repeatable.** Perhaps a function that finds the

Node * for a particular stock number within your linked list would be helpful to you. Perhaps a function that prints a particular item's stock number, department, the number of items on hand and item name, might help you. (Writing out 0 as items on hand for the out-of-stock items report would be acceptable). Perhaps a function that tells you whether or not a particular stock number is already in your linked list might help you. Perhaps a function that traverses the linked list and calculates one of the values needed in options 1 through 6 depending upon a parameter value might help you. Perhaps a function that, if given a **Node *** address, returns a **Cost *** address from within that node. These are a brainstorming list that I came up with as I wrote this lab description, not a requirement. This is just a bullet item to get you to think about how to minimize the amount of code you have to write while still getting the job done.

- **Simplify data structure pointers as much as you can.** Think about whether you can create an address to a specific element, assign that a value and access structure elements more simply. You might be able to create a function that is passed the address to a specific struct Data item rather than passing in the address to the specific Node. Again, this is just a bullet item to help you think about efficient, less complicated was to write your program.

- If the output for either print statement is empty (e.g. the print out-of-stock list has no grocery items to print, you must print a statement to that effect. Similarly, if you are asked to delete a grocery item 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(){
        return;
}
```


You can change the return value and the number and type of parameters passed when you have a better idea of what it should include later in your development cycle. **IF you do this, then make will work for you from the beginning and you won't have to do extra work late in your development cycle to split everything out into separate files.**

- 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 grocery items in the list.)

- You can assume that the user will always input a valid option (e.g. number between 1 and 12) so that you could use a switch statement for option#. This would allow you to have a specific case statement for option 10, 11 and 12 and could use the default case for your function pointer array statement. Keep in mind that you can pass parameters to a function that the function never chooses to use. ;-)

- Prior to exiting the program, you must free() all dynamically allocated memory. You can determine whether you have managed to accomplish this with the **valgrind** tool.

CONSTRAINTS

· **Source code files (.c files) submitted to Carmen as a part of this program must include the following at the top of each file:**

```
/* BY SUBMITTING THIS FILE TO CARMEN, I CERTIFY THAT I HAVE
** STRICTLY ADHERED TO THE TENURES OF THE
** OHIO STATE UNIVERSITY'S ACADEMIC INTEGRITY POLICY.
*/
```

If you choose not to put the above comment in your file, you may receive no points for the lab.

– You must comment your code

You must create and submit a Makefile that will be used to create your .zip file and your executable, **lab4**. This Makefile must define at least each of the following targets: **all**, **clean, lab4**, **lab4.zip**, **lab4main.o**, **and any other .o file needed to create your lab4 executable. Creating this file at the beginning of your development process and using it as you work, will allow you to test your Makefile for proper operation.**

- Separate your code into different functions so that your main() is uncluttered and easy to understand.
- **The file lab4.h must hold the prototypes for all functions you create in addition to the structure and typedef info from above.**
- No variables can be declared outside via a block. That is, no File Scope variables are permitted.
- The grocery data must be stored in a singly-linked list, where each node in the list contains the data for one grocery item. The head of the list cannot be a **NODE**, it must be a **NODE ***.
- You program MUST order all items by ascending stockNumber as it inserts them into the linked list. [This means that you **CANNOT** read all items in with no concern for order, then, subsequently, order the linked list. **If you choose to do this, you will receive 0 points for the lab**.]
- You are not permitted to declare any variables at file scope; you must, however, declare the Node type used to store the data for the linked list at file scope in a file called lab4.h (but no variables can be declared as part of this declaration). See the description of the declaration of the Node type above. There will be a 50% penalty for file scope variables.
- You may use the following I/O C library functions: scanf(), printf(), fscanf(), fprintf(), fopen(), fclose(). NO OTHERS unless specifically approved by your instructor in writing.
- All floating-point data will be entered with two digits of precision and must be output with two digits ofprecision and using monetary symbols and commas where appropriate.
- You must allocate the storage for each grocery item node structure dynamically.
- The grocery item name and department name must each be stored in strings declared to be of type ***array of char*** of size 50 and 30, respectively. (See the description of the declaration of the Node type above.)
- If a grocery item is deleted, you must call free() to free the storage for the node for the item.
- **You must use an Array of Function Pointers to call the User's Choice of Function for options 1 through 9.**

- You must write the inventory list to disk (filename2) and then free the space for the individual nodes that still contain data before the program terminates when the user selects option 12.
- See the text file posted on Piazza with sample input. This file will be posted shortly, with example screen output to follow. While waiting you can work on main/insert/delete.
- You must create a Makefile that creates your executable **lab4**. Insure the Makefile contains all appropriate dependencies to create lab4 and lab4.zip.
- the **valgrind** program must say that your program has no memory leaks possible

LAB SUBMISSION

Always be sure your linux prompt reflects the correct directory or folder where all of your files to be submitted reside. If you are not in the directory with your files, the following will not work correctly.

You must submit all your lab assignments electronically to Carmen in .zip file format. The format of zip command is as follows:

[jones.5684@fl1 lab4] $ **zip <zip_filename> <files-to-submit>**

where **<zip_filename>** is the name of the file you want zip to add all of your files to and **<files-to-submit>** is a list of the file(s) that make up the lab. Since the names of all the .c files can be different for each individual, I cannot create a specific list of files you will need to put in you zip file. Do not forget to include your readme file and your lab4.h file.

You must create your zip file from within your Makefile.

I highly recommend testing your .zip submission you've created with the procedures supplied in the Lab 1 Description Document under the title of **TEST TO CONFIRM THAT YOUR .zip FILE HAS EVERYTHING IT NEEDS.**

NOTE:

• Your programs MUST be submitted in source code form. Make sure that you zip all the required .c files for the current lab (and .h files when necessary), and any other files specified in the assignment description. Do NOT submit the object files (.o) and/or the executable. The grader will not use executables that you submit anyway. She or he will always build/compile your code using your Makefile after inspecting it to insure it compiles using **gcc -ansi –pedantic** and then test the executable generated by that command.

• It is YOUR responsibility to make sure your code can compile and run on CSE department server **stdlinux.cse.ohio-state.edu,** using **gcc -ansi –pedantic** without generating any errors or warnings or segmentation faults, etc. Any program that generates errors or warnings when compiled or does not run without system errors will receive 0 points. No exceptions!