Lab #4 Assignment

CSE 1320-003

Fall 2013

Design Document Due Date: **Tuesday, Dec. 3, 2013 at 11:59am**

Testing Plan Due Date: **Tuesday, Dec. 3, 2013 at 11:59am**

Lab Due Date:  **Wednesday, Dec 11, 2013 at 11:59am**

**Extra credit for early submission as follows:**

**+ 9 points by Tuesday, Dec 10, 2013 at 11:59 PM**

**+ 20 points by Monday, Dec 9, 2013 at 11:59 PM**

**+ 33 points by Sunday, Dec 8, 2013 at 11:59 PM**

(see instructions on website for how to turn this in - “lab submission info”)

Grade value: 10% out of 100% for all grades

Objective: The goal for this lab is to provide an opportunity for practice of C program development and the C programming topics covered in class. It is assumed that the student is familiar with the programming concepts but not the C syntax.

Every lab assignment allows students to practice program development, debugging, and testing. All of these skills are crucial to success in Dr. T’s class. The format of this assignment is similar to the assignments that will be required for the rest of the semester so a student who is unsure of their skills can use this assignment to refresh them and to ask the instructor or TA about any concepts they are unsure of.

Topics:

Recursion Algorithms

Pointers Multi-dimensional Arrays

Linked lists Strings functions

Control structures Input and output,

File input Sorting

Searching Program design

Modular programming structure Error checking

Programming style Compiling code

Executing code Debugging and testing.

Plan: Dr. T gives an overall problem that students work on all semester. Each lab assignment implements specific concepts. Succeeding assignments will modify and extend previous assignments.

Overview: For the end of the semester you will be trying to determine the staff details of what it would cost to start and then run a small garden store and then some methods to sort and search the data. You will be developing an estimate based on a number of factors including the size of the store, the kind of stock, the start up costs, staff costs, and other factors. (If you really own a business, you will quickly realize that this is not going to be a very realistic example, even later in the semester, but the idea is for you to practice your programming skills. ;) You will also make various other calculations as required. You are going to get information from the user and store that information in a certain way. We’ll call this the data-entry phase of the program. Once all the input data is entered, your program will allow another user to perform various tasks on the data by choosing tasks from a menu. We’ll call this the menu-driven phase of the program. The user can continue to choose tasks from the menu as long as desired and one of the menu choices must be to end the program. The program ends correctly when the user chooses that option.

You are also required to design your program in advance before you begin writing code. You will document your design and turn in the design document at least a week before the lab assignment is due. The goal of the design document is to assist you in developing the actual program.

This assignment has an overview section, a large data description section, a task description section, an implementation requirements section, a grading scale, and a deductions section. If there is additional info needed it will be in a miscellaneous section at the end of this lab assignment. Read ALL of the assignment before you start trying to develop the program.

Be sure to check the DEDUCTIONS section at the end of this assignment to avoid penalties. You may NOT use global variables, the exit command, goto, break (except in a switch), or continue. You should do input and output using printf, scanf and getline (you are allowed to use getchar and putchar if desired.)

Assumptions for the lab #4:

1) Strings will have a maximum length of 100 chars and they may have blanks. Use constants to define the max value.

2) If your instructions tell the user to enter numbers, then you do not have to check that the input might be a letter instead of a number unless specifically instructed to do so for some particular input. If the user does not follow your instructions to enter a number when required then any data type input error is the USER’s fault and you will not be penalized in this lab.

3) You will have a maximum number of 25 rooms (MAXROOMS)

4) You will have NO maximum number of employees (also called staff) (MAXSTAFF) for Lab #4.

5) For date checking in Lab #4, you may make the simplifying assumption that February has 30 days. All other months should be used with their actual count of days.

Problem: The data in your database will be the garden store information needed to create the estimate such as rental space rates, stock information, and so on. As part of this assignment you will have to find real data about some items to use as test data for your program. When the user runs your program, they must first enter data. The data they entering will be the garden store information that you have researched to use as your test data. After (or while) getting all the input data from the user (and storing it in arrays) you must also check that all the input they give is valid, i.e. there will not be a negative number of work hours. This will complete the data-entry phase of the program. Then you will allow the user to begin the menu-driven part of the program and do the following tasks:

1) Using the data in the arrays,

A) Print all or part of the data in a table

B) Calculate

i. Total estimated first year costs

ii. Staff costs using the enhanced staff data in the linked list

C) List

i. Rooms in order by size in square feet

ii. Rooms with size equal to a given square footage

D) Determine scenarios: None required for Lab #4

E) Search for

i. Garden section/room in the store with a given size using binary search

ii Highest paid employee

F) Sort

i. Garden section names

ii. Garden section/room sizes using merge sort

2) End the program

Data Description: For lab #4, you will be storing the input data in some single arrays, some multi-dimensional arrays, some arrays of structs, and a linked list of structs.

The data elements and descriptions are given below. Following this first list, the data elements are described by type.

a) Start up costs – a money amount composed of the cost of a business license, the total cost of the utility connection fees (both water and power), the cost of a business permit, and an amount to have as a start up fund. This set of data will also include an opening date for the garden store.

b) Type and amount of flowers and plants to stock represented by a single LETTER with A meaning annuals, H meaning high water use perennials, X meaning xeriscape perennials, N meaning native shrub, T meaning native tree, and O meaning other plant. The user shall give an integer (the count of types of stock available), then letters (representing each kind of available stock), then more integers representing amounts). This information will be formatted as an integer of either 1, 2, 3, 4, 5, or 6 followed by that many letters of either A,H, X, N, T, or O then followed by that many numbers indicating amounts of each type of stock. Your garden store must offer at least annuals and native shrubs at a minimum. The number represents how many types of stock types are available and the letter code(s) indicate which kinds of stock they are.

Ex: 4 X A N O 65 500 150 325

c) Stock type average cost per item – same number of values as previous info giving a list of cost of the average item of each kind of stock. Ex: for example above, 35.50 2.00 22.00 7.25

d) Desired profit amount (markup) per stock type – same number of values as previous info giving a list of values that indicates a percentage of markup to be used on that stock type. (Markup can be 0.)

e) Staff data – there will be four types of staff members - manager, owner, horticulturist, and sales staff.

Each staff member will have a pay rate per hour, a number of hours worked per month, and an ID number. A total yearly staff cost will be calculated from this info.

f) Type of rooms in the store including outdoor garden section "rooms". The garden store will have some indoor/interior rooms for plants as well as some outdoor garden section "rooms" for plants. The types of rooms are Indoor sales room, Loading area, outdoor Sales room, Bathroom, Office. The input is an integer of either 1, 2, 3, 4, or 5 followed by that many letters of either I, L, S, B, and/or O. Must have indoor sales room (I), loading area (L), and bathroom (B) at a minimum. The number represents how many types of rooms are in the store and the letter code(s) indicate which kind of room they are. Ex: 4 S I B L

g) Amount of rooms of each type – same number of values as previous info giving a list of numbers representing how many rooms there are of each type. Must have a minimum of two indoor sales rooms, one loading area, and one bathroom. Ex: for example above, 3 1 2 3

h) Size of each room – this is a set of pairs of values, one pair for each room counted in the amount of rooms above. The values represent length and width of the room in fractional feet (ex: 3.5). For the example just above there are 3+1+2+3=9 rooms, so there will be 18 values used to give the sizes.

i) Square foot rental rates per year – two floating point money amounts that are the amounts that one square foot of indoor space and one square foot of outdoor space rents for. Ex: 120.00 75.50 would represent $120.00 per indoor square foot per year and $75.50 per outdoor square foot per year.

j) Total yearly expense – This money value will be calculated by the program

k) Total yearly profit - This money value will be calculated by the program

l) First year profit or loss - This money value will be calculated by the program

m) Garden section number – This will be assigned by the program but stored as a value in the garden section array to allow for sorting and re-sorting

o) Sales garden section name – a phrase less than 100 characters long that gives a name for each of the garden section rooms. In lab #3 we are using strings for the name, i.e. "English Cottage Garden Section" is one phrase and can be used as a garden section room name. After you read in this name string, you will need to append (concatenate) the garden section number to the end of the string. See sample code on website for assistance.

p) Square footage of each garden section – This amount will be calculated by the program and stored in the garden section array

r) Profits per garden section – For each garden section room in the store that has stock, this array will hold the gross profit and net profit for the garden section room. These values are calculated by the program based on the data input for each garden section room, the stock types and amounts in each garden section room, the AVGPRICE for each stock item (which is the item's cost to the store), and the MARKUP amount for each stock item.

The specific detailed format for each piece of data is listed below.

General data storage requirements for lab #4:

Overall you are required to use some multi-dimensional arrays (data content and sizes will be specified for each one) and at least two one-dimensional arrays of structs. You may also use additional one-dimensional and multi-dimensional arrays to hold the remaining data as needed. [OK, this isn't as confusing as it seems. ☺]

As in Lab #3 we will be using structs to contain data for different components in our program and then we will store these structs inside arrays or in linked lists.

As an example, let's look at the room data for a garden store. You will have store information about what kind of room it is (room type), how many rooms there are of this type, and then for each room of that type you need its length and width. We will use a one-dimensional array of structs to store data about the rooms. Each struct will be one room. Each member of the struct will represent different data about that room in that struct. Since we will also be naming the garden sections in the rooms, then there will also be a string array, the name of each room/ section, in each struct.

You will have a maximum number of 25 rooms (MAXROOMS) for Lab #4.

This information then dictates the sizes of the various types of arrays you will need. These arrays are specified in detail below. We will also modify the data storage for some of the data we already have.

The first array will hold information about the stock, *stockinfo*. This will be a floating point array of 6 rows and 3 columns. Each row will represent one of the 6 types of stock that might be in the store. You should declare the following constants to use with this array: ANNUAL = 0, WATER\_PERRENIAL = 1, XERIC\_PERRENIAL = 2, NATIVE\_SHRUB = 3, NATIVE\_TREE = 4, OTHER = 5. The three columns in the array will represent AMOUNT = 0, AVGPRICE = 1, and MARKUP = 2. The AMOUNT column will be used to represent whether a particular kind of stock is available at that store or not by holding the amount of that stock in the store. So if the store sells xeric perennials, then *stockinfo*[XERIC\_PERRENIAL][AMOUNT] will be store the amount 65 which represents the amount of xeriscape perennials (from the earlier example). If native trees are not sold at this store, then *stockinfo*[NATIVE\_TREE][AVAILABLE] will be set to 0.0 to represent 0 items in stock. If the AMOUNT column is greater than 0 (or true), then values should be stored in the AVGPRICE and MARKUP columns. If the AMOUNT column is 0 (or false), the AVGPRICE and MARKUP columns should have the value 0.0. {HINT: If you initialize your array to 0, then you will have the correct values in the columns you don’t need.} For Lab #4 you may continue to use the stock arrays as you did in lab #3 or you may move the stock data into a struct and have one array of stock structs.

The second array will hold amount information about the numbers of types of staff members. The array will be a one-dimensional integer array. Declare constants to use for the indices, ex. MANAGER = 0, OWNER = 1, etc. for the four staff types. Inside the array, store the amount (number of) items associated with that type. These amounts are requested from the user. For example, if the user says there are two managers then the value for *stafftype*[MANAGER] should be set to 2. Be sure to declare your constants correctly for the array you are using.

The third array will hold amount information about the numbers of each room type. The array will be a one-dimensional integer array. Declare constants to use for the indices, ex. INDOOR = 0, LOADING = 1, etc. for the five room types. Inside the arrays, store the amount (number of) items associated with that type. These amounts are requested from the user. For example, if the user says there are three outdoor sales rooms then the value for *roomtype*[OUTDOOR] should be set to 3. Be sure to declare your constants correctly for the array you are using. Error check the values that are stored in the array so that the user does not exceed the maximum number of rooms.

The next array will be an array of room structs that will contain all the data that was previously in the roomsize array and the sectionname array. It will also include data for each room including the room area in square footage, how much stock of each type is in each room, the gross profit from that room assuming 100% sales, the net profit from that room and a room number as a value also.

So the room struct type you create needs to have a room type, a room number, a room/section name, a length and a width, an area (that is calculated from length and width), the amount of each type of stock (so this is 6 amounts) in the room, a gross profit value, and a net profit value.

* The room type will use the constants already declared, like LOADING.
* The room number would be a copy of the original counter which will be the room number as counted from 0 to *roomtype*[index]-1.  You are storing this number so that you will be able to sort and then re-sort the array.
* The string array inside the struct will have 100 columns. Both indoor sales rooms and outdoor sales rooms can have names. Make sure to declare all your constants (such as max string length) before you try to use.
* The values for length and width stored in this struct should be measurements in fractional feet of the length and width of the current room.  Therefore if the 2nd office is 8’6” by 9’4”, then the *width* should be 8.5 and the *length* should be 9.33.   The area of the room should be calculated automatically be the program and stored in the *area* member of the struct for that room.
* There will be 6 stock amounts to correspond to the six types of stock that could be in the room.
* The gross profit is all the money paid for stock by customers, and the net profit is the gross profit minus the actual cost (the AVGPRICE) for each stock item. All of the profit data comes from information the user enters in other places.  After the user has entered all the input data, your program must calculate these profits and costs and fill the structs in the array with data.  I would recommend writing a function to find stock types, amounts and costs for each room, calculate the profits, and store it all in this array.  The gross profit for a room is calculated as:

            stock1\_amount \* (stock1\_avgprice \* (1+stock1\_markup)) + …

+ stock6\_amount \* (stock6\_avgprice \* (1+stock6\_markup)) = gross\_profit;

Then net profit is :

            stocktype1\_amount \* stocktype1\_avgprice

+ stock2\_amount \* stock2\_avgprice + …

+ stock6\_amount \* stock6\_avgprice = actual\_cost;

            gross\_profit – actual\_cost = net\_profit

The last major data structure will be a linked list of structs representing the staff members. The staffmember struct will contain an ID number (long int) for a staff member, a staff type (MANAGER, OWNER, etc.), an hourly pay rate, and a number of hours worked per month (assume each staff person works the same number of hours every month). The linked list will be created by reading in data for each individual staff member and storing it into the linked list.

In addition to the arrays described above you may use more arrays if desired or you may add columns to the arrays already described in order to represent the remaining data.

In Lab #4 you will continue to use a struct type to use for storing the start up cost information and a struct type to hold date information.

Detailed descriptions of each data value. The data values below must be stored as described. You may also store additional values into these arrays if desired. In the descriptions below *number* usually represents the current room number that is being used.

It is NOT necessary to have the user enter the data in exactly the order listed. You may get data from the user in whatever order makes sense. However, you must implement the data input method described later in the document for "file input method" and if you enter the data in an order different than the example below, then you must specify the order for your user.

*Start up costs:*

Declare a struct type that can contain all of the start up information including opening day. Store all the start up information in a variable of this struct type.

*Business license cost* – floating point money amount.  Must be 0 or positive cost. This should be error checked and then stored as a floating variable in a struct.

*Total cost of the utility connection fees* - floating point money amount.  Must be 0 or positive cost. This should be error checked and then stored as a floating variable in a struct.

*Cost of a business permit* - floating point money amount.  Must be 0 or positive cost. This should be error checked and then stored as a floating variable in a struct.

*Start up fund amount* -  floating point money amount.  Must be 0 or positive cost. This should be error checked and then stored as a floating variable in a struct.

*Opening day* – three integers that give the target opening date for the store.  These values should be error checked and then stored in a date struct.

*Stock data:*

*Number of types of stock* – a single integer from 1 to 6 which indicates how many different kinds of stock there are in this store.

*Stock type* – a single character or a group of characters representing the kind of stock that will be sold. The total number of these characters will be the same as *number of types of stock.* Each character must match one of the valid stock type codes and the codes that are used must be entered in the same fixed order as the list of stock types. For each valid code, a one (1) should initially be stored in the array *stockinfo*[*type*][AMOUNT] where *type* is the constant stock type value that matches the entered code. Ex. If user enters N as one of the characters, then 1 should be stored in *stockinfo*[NATIVE\_SHRUB][AMOUNT].

*Stock amount* - an amount that is the number of items available of the current type. This value is stored in the floating point multi-dimensional *stockinfo* array at the [*type*][AMOUNT]where *type* is the constant stock type value that matches the current type and where the value of AMOUNT was 1 from the previous step.

*Stock type average cost per item* - a floating point money amount that is the average cost for one item of the current type. This value is stored in the floating point multi-dimensional *stockinfo* array at the [*type*][AVGPRICE]where *type* is the constant stock type value that matches the current type.

*Desired profit amount (markup) per stock type* - a floating point number between 0 and 3 inclusive. This limits the profit to 300% of the cost, i.e. if the average cost of this stock item is $5.99 then the item can be priced not more $23.96 which pays the cost of the item and then returns 300% profit. This value is stored in *stockinfo* array at the [*type*][MARKUP]where *type* is the constant stock type value that matches the current type.

*Staff data:*

*Staff member data –* a set of data to go in a staff member data struct which an employee ID number, a staff type code, an hourly pay amount, and an hourly amount per month worked. Ex. 80043346 M 32.50 175 . The struct for a staff member should also contain a forward pointer and a backward pointer to allow hooking up the linked list.

The staff data struct will have a link pointer included. A simple name to use would be *nextstaff* for the staff data struct pointer. The link pointer must be a pointer to the data type of the struct. Therefore, inside the *staffdata* struct type you will be adding a struct member of *struct staffdata \*nextstaff*.

Additionally, you will need to declare some external pointers in the functions to use in traversing the linked list. For example, you will probably want something like:

*struct staffdata \*start, \*curr, \*prev,* and *\*follow* for the external pointers for the staff linked list.

*Room data:*

*Room type and amount* - a single character or a group of characters representing the kind of rooms there are at the store. They must match one of the valid room type codes. For each valid code, the user should be asked for an amount of rooms of that type. This value should be stored in the array for room data at *roomtype*[*type*] where *type* is the constant room type value that matches the entered code. Ex. If user enters L as one of the characters, then the user should be asked for how many loading areas there are and this value should be stored in the room array at *roomtype*[LOADING]. Ex. 4 S I B L 3 1 2 3

*Square foot rental rates per year –* two floating point values that will be used with the roomsize to work out the yearly rental cost for the indoor store space and the outdoor store space to determine the whole store rent cost. Ex. 120.00 75.50

After getting the room types, amounts and indoor and outdoor rental rates, you will use one room struct for each room of data to be stored. The room structs should be stored in a one dimensional array. [You may store the structs in a two dimensional array if desired but make sure you know how to search your two dimensional array if you use it.] A single struct will hold all the data for a single room. If you have all your structs in a one dimensional array called *rooms* then you might use *rooms[i].width* to store the width and *rooms[i].xeric\_perennials* to store the amount of xeric perennial flowers in the same room *[i]*.

*Room length and width* – this is a pair of floating point values for each room in the store representing length and width of the room in fractional feet (ex: 3'6" = 3.5'). You must have the user enter feet and inches and then convert this value to fractional feet. Regardless of how the user enters the data, the data MUST be stored as a floating point number representing a fractional measurement. This same process will be needed for each type of room that is in the store and for all the rooms of that type. For example, if there are 2 baths, 3 indoor sales rooms and 1 loading area, then there are 6 different rooms to get length and width for. Since the possible number of rooms is large (ROOMMAX is 25), this data entry could take a long time. [HINT: start with a small number of rooms and then after testing it, increase the numbers for your final test runs to script.] Once you have gotten the length and width for a room, you should calculate the area and also store that in the correct struct member.

*Room stock content* – this is an integer value giving the amount of a given stock type in a single room/section of the store.  This data is stored in the room struct.   For all the types of items sold in the garden store, some items of each of those types might be in any section or room.  Therefore for each room, you will need to ask the user how much stock of each type is in that particular room and store the answer.  Ex:  If the garden store sells ANNUALs and NATIVE\_SHRUB, then each room might have some stock of that type in it. You would need to ask the user, room by room (just like for length and width), how many items of each type are in that room (of the types that are sold.)  The result is that if the 3rd indoor sales room has 125 native shrubs in it, then you will get and store 125 in the native\_shrub member of the room struct that represents the current room such as *rooms[k].native\_shrubs = 125*.

*Room number* – this is an integer counter value that is stored in the struct for the rooms of the current type.  When you are getting the data from the user, you will be asking them about the data for INDOOR room 3, for example.  In addition to getting the length, width, and stock data for the room, make sure you also store the number of the room in the array, in this case 3.  This will later allow you to sort the array by amount of ANNUAL stock, for example, and then be able to re-sort back to room number order.

*Sales room name* – a string will less than 100 characters that gives a name for each indoor and outdoor sales room that has one.  The string will be read in with the getline command into a temporary string.  The length of the string should be checked using the *strlen* command. If the string is less that 1000 chars in length then it should be copied to the name member of the current struct. Ex: The name of indoor sales room 5 which is the hardy shrub room might be the string "Hardy Shrub" and it would be stored in *rooms*[5].*name* which is the string array. [HINT: make sure you know how to read a string in, check its length, and then copy it in order to make this work.]

*Gross profit* and *net profit per room*– Is calculated from other data that has been input.

*Total yearly expense* for the store – This money value will be calculated by the program as the business costs plus a year's rent plus the cost of all of the initial stock.

You are allowed to make assumptions about rooms with no stock (like BATHROOM or OFFICE) but make sure to write comments about this assumption in your code and to fill in the struct member or array element appropriately, with 0's, for those rooms.  If some of your rooms do not have any stock, then you do not need to ask the user about stock in those room types but you do need to make sure there are 0's in those values in the array.  You may also choose to ask about the size and the stock for one room all at the same time.  This means that you could ask the user about indoor sales room 3 and get length, width, amount of annuals, amount of xeric perennials, and other stock, then ask for the room name. You would already know and have stored the information that the room was type INDOOR and had room number 3 and you would calculate the area, gross profit, and net profit to store those values. After getting all the data for INDOOR sales room 3, then you could go on to ask about sales room 4.

# Data entry phase

Input Implementation: For Lab #4 the user must enter at least 3 rooms (see earlier requirements) and no more than 25 rooms. The user must enter at least 7 staff members and there is no maximum number. The user must have at least two kinds of products (stock) in their garden store as previously defined. As this data is entered by the user, the program should check to make sure these counts are appropriate. [Note for development: start with smaller numbers then increase to the required minimum when program is working well.] If the user’s number exceeds the max, the program should inform the user of the maximum number of inputs allowed and have them reenter the count.

Once you get to input for the rooms, your program must loop for each room to read and store input for the length and width. You may collect all the room information in this same loop or you may loop through the rooms more times in order to get stock amounts and other data.

Make sure your program begins with a brief intro to the user so that they know what the program is for.

For Lab #4, you must implement the following method of input.

Your program must allow the user to enter data from a file. The format of the data in the file must be the same format as the data when entered on one line from the keyboard. For Lab #4, you must name your input file "CFall13Lab4input.txt".

If your program implements both line of data input method and file input method then the program must ask the user how they want to input the data. Once the user has chosen the method of input or if only line of data input method is implemented, then all the input will be done that way for the current run of the program.

File input method:

[Data definitions were given in Lab #2 and Lab #3 in the Individual data input method descriptions. Continue to use those data definitions unless new definitions are specifically given in this assignment.]

The file input method is to have the necessary data entered by the user on just a few lines with multiple pieces of information per line. The input would be all the pieces of data listed above. You must tell the user exactly how to enter the line of data. For Lab #4, use the line of data method with inputs in exactly the order specified below.

The data for the garden store will be entered by the user as values on two (or more) lines as follows {there must be one space only between the values}. The expanded staff data will be read at the bottom. Here's an example:

prompt>500.00 250.00 69.99 50.00 15000.00 9 1 2014 4 X A N O 65 500 150 325 35.50 2.00 22.00 7.25 .75 .90 1.00 .90 3 M H S 2 1 5 120 48.8 76.4 27.50 4 S I B L 3 3 2 3 120.00 75.50

0 0 20 25 40.0 60.0 Hardy\_Shrubs

0 0 100 10 20.0 20.0 Native\_Shrubs

0 0 0 65 20.0 20.0 Hardy\_Trees

0 400 0 75 55.5 20.0 Annuals

65 0 0 10 10.0 19.3 Perennials

0 0 0 140 45.5 35.3 Flowers

10.5 10.0

9.5 10.0

15.0 10.0

10.0 5.0

20.0 10.0

45634543 S 20.90 173

12387665 H 25.50 200

17668532 S 18.35 177

59366820 S 15.00 169

23685669 M 31.09 161

76543987 S 12.78 140

43211223 M 29.60 165

65647355 S 13.11 155

which represents a business license cost of $500.00, a power utility connection cost of $250, a water utility connection fee of $69.99, a business permit cost of $50, and a start up fund of $15,000. The opening day will be 9-1-2014 (September 1, 2014).

The store will have four types of stock – xeric perennials, annuals, native shrubs and other with the amounts – 65 xeric perennials, 500 annuals, 150 native shrubs and 325 other plants. The average costs of these stock types is $35.50 for xeric perennials, $2.00 for annuals, $22.00 for native shrubs and $7.25 for other plants. The markup is 75% for xeric perennials, 90% for annuals, 100% for native shrubs, and 90% for other plants. There are three kinds of staff members – managers, horticulturists and sales staff. There are two managers, 1 horticulturist, and 5 sales staff.

There are 4 types of rooms in the store and the following amounts – outdoor sales (3), indoor sales (3), bathrooms (2), and loading areas (3). The rental rate per indoor square foot per year is $120.00. The rental rate per outdoor square foot per year is $75.50. [For this assignment, I assume that bathrooms and loading areas do not contain stock items.] All the preceding data is typed with single spaces between them all on one line. (As long as you only type in single spaces, it will not matter if the line wraps on the screen when you look at it like above. It will still be one line to C.)

The next data is one line for each room. The order of the lines matches the order of the rooms given so 3 outdoor sales rooms, 3 indoor sales room, 2 bathrooms, and 3 loading areas is 11 lines. For sales rooms there are stock amounts in the same order as the stock types were input, then length, width, and room name. In the example above, the first outdoor sales room has 0 xeric perennials, 0 annuals, 20 native shrubs and 25 other plants and is 40' by 60', and is named "Hardy\_Shrubs". The second outdoor sales room has 0 xeric perennials, 0 annuals, 100 native shrubs and 10 other plants, is 20' x 20' and is called "Native\_Shrubs." Likewise for the third outdoor sales room called "Hardy\_Trees". The stock amounts and length and widths are given for the fourth and fifth rooms - indoor sales rooms, "Annuals" and "Perennials". The sixth room is the 45.5' x 35.3' indoor sales room names "Flowers". The 7th room is the first bathroom of size 10.5' by 10' and the second bath is 9.5' x 10'. Three loading areas are 15' x 10', 10' x 5' and 20' x 10' in size.

Following the room data will be data for each staff member. Each line has the data for one employee with four pieces of data; an employee ID number, a staff type represented by a letter, the hourly wage for that employee, and the average number of hours that employee works in a month.

To use the file input data, your program will read the first number and store it in the correct location for the business license data, then read and store the next number into its corresponding location, then read the third number, and so on. Your program should try read all the numbers, chars, and the strings that are on a single line within a single input command as much as possible, i.e. do not have a scanf statement for every single value.

Your program should read in one line for the first set of garden store data and then as many lines of input for rooms specified by the counts given up to the maximum of twenty-five more lines of input. After the room data lines then there are employee lines of data. Your program will know that you have reached the employee data when the value of the first number on the line exceeds 9999999. You will not typically know how many employees there are. You should read employee lines until you reach the end of the file. The employee data should be stored in structs that are put into a linked list.

File input method :

This input method is to have all the necessary data entered by the user from a file. The input would be all the pieces of data listed above and would use the format specified. Be sure to tell the user the name for the data file so that they can run the program with their own data file.

You should make two test files with different data to test with. Copy the test file to the specified file name and then test with it. Do this for both your test files.

File input method additional information::

Your program needs a FILE \* pointer which is connected to the file CFall13Lab4input.txt (the default file name) with *fopen*. Ex:

FILE \*infile;

infile = fopen("CFall13Lab4input.txt", "r");

/\* If your user is entering the file name you would use the code below \*/

int filenm = 100;

char \*filename;

FILE \*infile;

printf("Please enter the complete name of your data file: ");

getline(&filename, &filenm, stdin);

if (filename != NULL)

infile = fopen(filename, "r");

Input verification:

When the user has entered all the store, room, and staff data, print out all the input data in an easily readable form, ex. use a table with headings, or columns with headings or rows with labels. It is strongly suggested that this printing be written as a separate function that can be called at any time in the program to print the current contents of the arrays and variables. It is also suggested that a small function be written which will print the name of a sales room given the type of room (INDOOR or OUTDOOR) and room number (Ex. outdoor sales room 1, 2, or 3) as input. The function could have a switch or a bunch of if statements in it. You may wish to write small output functions for printing staff data, stock data, start up data and room data if desired. These could be combined to print the data required for input verification.

Task Description: (*Menu-driven phase*)

Once all the data is read into the arrays your program should give the user a main menu with the following choices. This menu is just a listing of the choices NOT a new window or any other sort of graphics based menu. The user should select a choice by entering a letter of number at a prompt at the bottom of the menu. The menu choices are: (use any number scheme you wish)

1) Using the input and calculated data,

A) Print all the data about the store including total expenses, gross profits, and net profits for a year

B) Calculate staff costs using the enhanced staff data in the linked list

C) Calculate total estimated first year costs

D) List all rooms in order by square footage

E) List all rooms with square footage equal to a given amount

F) Search for a garden section/room in the store with a given size by using binary search

G) Search for highest paid employee

H) Sort by the garden section names

I) Sort the garden rooms/sections by sizes of area using merge sort

J) End the program

The print function A lets the user select to print one set of store data or all the data. These functions should call the function(s) that were discussed in the input verification section above.

The calculate-staff-costs function B has the following steps:

For each staff member in the linked list

Multiply hourly wage

times hours per month

Sum this amount for all employees

to get total monthly staff cost

Multiply this value times months of the year to get yearly staff cost

The calculate-first-year-costs function C has the following steps:

Sum all the startup costs

Find the yearly staff cost (this should use function B)

Calculate the total stock cost (we'll assume this is the first month's stock) as:

For each type of stock in the store

Multiply amount of items of stock type

Times average cost for that kind of stock item

Sum these stock costs for total monthly stock cost

Multiply this value times months of the year to get yearly stock cost

Calculate store rental cost as:

Total store square footage times outdoor percentage

Times outdoor square foot rental rate per year and

Total store square footage times indoor percentage

Times indoor square foot rental rate per year

Sum the indoor costs and outdoor costs for store rental cost

Sum the costs for startup, yearly staff, total stock, and store rental to get the first year cost

The list-all-rooms-in-order function D should first sort the rooms array by size (using function I) then should print out the list in order.

The list-all-rooms-equal-to function E should ask the user to enter the square footage of a room then go through every room and print the complete information for all rooms that have an square footage equal to the user’s entered value.

The garden-section-room-size search function F has the following steps:

Ask the user to enter the square footage of a room

Use **recursive binary search** to search the area amounts of all rooms in the garden store to find a match

Print the info for the first matching room including room type, number, name, length and width

The highest paid employee search function G has the following steps:

Use any search method discussed in class to

Search all the staff structs in the linked list

to find the largest monthly pay for an employee (hourly wage \* hours)

Print the information for the highest paid employee

The sort-by-name function H has the following steps:

Use any sort function discussed in class to

Sort the section / rooms by name alphabetically

For rooms with no name, put them at the end of the list in any order

Print the sorted list

The sort-by-area function I has the following steps:

Use the **recursive** **merge sort** function discussed in class to

Sort the section / rooms by area from largest to smallest

Print the sorted list

When the user chooses "End the program" from the main menu, print a concluding message and then gracefully end the program.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

You should design your program in advance before you begin writing code. In this lab and all other labs you will document your design and turn in the design document at least a week before the lab assignment is due. The goal of the design document is to assist you in developing the actual program.

Read ALL of the assignment before you start trying to develop the program.

Be sure to check the DEDUCTIONS section at the end of this assignment to avoid penalties. You may NOT use global variables, the exit command, goto, break (except in a switch), or continue.

Develop your program in small steps. Test and debug as you go.

Implementation Requirements:

For Lab #4 WRITE A DESIGN DOCUMENT FIRST. The design must include

a) all the functions you expect to write,

b) brief (one line) descriptions of each function, and

c) some indication of which function calls what other functions.

d) an indication of what order you plan to work on the functions in your design.

When you indicate the development order for the functions, explain why you chose to do that function in that order. You explanation should reference things like the point value of that task, the ease of making changes, the amount of work required, whether the function is similar to another so you will save it until later, whether a function does something completely new, whether you think you will even have time to work on the function, and other reasons.

The design document may be written as lines of text or as diagrams (such as a diagram that start with the main function at the top and all others below it, a tree, UML, etc.) or as some combination of those, but it must include the information listed for a), b), c), and d) above. Each function should accomplish one main purpose and each function at a lower level should have a more specific purpose than the function that calls it. Be sure to include all the functions that are described in this lab assignment. This design documentation will normally be turned in for the lab and a design will be required to be turned in for all labs after Lab 1. See the website or this assignment for the DESIGN DOCUMENT due date. It is usually ONE WEEK PRIOR to the lab due date.

A Lab #4 Design Document must be turned in in order for your Lab #4 assignment to be graded.

For Lab #4 you must also write and turn in a TESTING PLAN along with your design document. The testing plan should include at least five (5) sets of test data values that will test at least the following elements of your code:

Show the validation of file input initial costs and correction of invalid cost

Demonstrate correct calculation of staff costs

Demonstrate correct searching of the linked list

Demonstrate a search that does NOT find a matching value

Deomonstrate sorting the array by name and then resorting the array by area

Give your set of test values for each test element above and indicate what the correct result of the test should be and how to demonstrate that this result has occurred, i.e. will the user have to ask for the data to be printed out or will that happen automatically?

A Lab #4 Testing Plan must be turned in in order for your Lab #4 assignment to be graded.

The program should use the following data structures:

User defined struct types and pointers

Multi-dimensional arrays, single-dimensional arrays, arrays of struct, linked lists

Global CONSTANTS for specific and/or maximum values given in this assignment.

Constants can be done with #define or with the const declaration. Examples:

#define EVENTMAX 10

const int EVENTMAX = 10;

The program should use the following control structures:

Function calls to perform tasks

A selection structure to choose input style

A while or for loop to read the input data

If, if-else, nested ifs, or switches to error check and implement the menu options

The program should NOT use:

global variables

goto

exit

break (except in a switch)

continue

any topic not covered in class before the lab DUE date unless approved by the instructor

Programming practices:

Comment your code! Use headers as described below, use line comments and use block comments. Line and block comments should explain the meaning of the code. As an example compare the following examples of code with comments:

Example 1:

/\* Find A by multiplying L times W \*/

A = L \* W; // A is the product

Example 2:

/\* Find the area of the rectangle by multiplying length times width of the sides \*/

A = L \* W; // A is area, L is length of one side, W is length of perpendicular side attached to L

While Example 1 has comments, the comments do not tell us anything extra about the code. The code itself tell us that A is the product of L\*W. In Example 2, the comments are meaningful and explain the goal of the code and the meaning of the variables. Make your comments like Example 2 not Example 1.

The program should have a program header which gives, at least, your name, the number of the lab assignment, your class and section, the assignment date, the due date, and a description of the program. If multiple files are used, each file should contain a similar header.

Each programmer-defined function, i.e. each function you write, should have a function header. This header should include at least the function name, the purpose of the function (which can be the one line brief description of this function from the design document), and its inputs and outputs.

The program should be implemented as a set of functions with a main routine and at least one function for menu operations, one for getting input (file input), one for printing, and one for each of the functions listed for the user choices. This is the required minimum number of functions. You may use many more functions than this but you must use at least this many. The purpose of functions is to divide the problem into small tasks, each one assigned to its own function and then to call the functions from main() or from another function when appropriate. Do not code the entire program in main!

For Lab #4, you must split your program into at least three separate program files as follows:

All constants and type definitions in one file,

The main routine and the functions prototypes in one file, and

The remaining functions in one or more physical files.

Compile them all together with #include statements or with gcc to create the output a.out

The program should generally perform the following actions in the given order:

Declare and initialize the variables

Print a welcome screen for the user that introduces the system

Get the needed input values

Print the appropriate outputs

Let the user make additional choices until the user indicates that they are finished.

Output requirements:

**The program must be run and the output recorded in a script file from OMEGA using the gcc compiler.**

No Exceptions! If you do not know how to create a script file, it is your responsibility to ask the TA, look for help on the class website, or OIT how to use this function.

You must come up with information to use as test data for 3 kinds of stock, 2 kinds of staff, and 3 rooms as a minimum. See the Miscellaneous section below for other useful info.

Testing:

**This program must be run with two different sets of test data for the input data. This is what gets recorded in the script file. Your test runs should cover all the tests described in your testing plan.**

You must create two different data sets and run your program with both of them. I will give you a small amount of sample data at the end of this lab for you to use as a model. You may run your program two times within a single execution or you may execute the program two different times so that you have a total of two different data sets. The sample data sets that you create must meet the guidelines given in the problem definition. Your test runs should demonstrate all the choices that are available in your program.

NOTE ABOUT ERRORS:

Programs turned in for credit MUST compile and run WITHOUT any compilation errors or runtime errors using the gcc compiler on omega. No other compiler may be used for the compilation for credit. No other operating system may be used for the compilation and/or execution for credit.

*Compilation errors* occur while the program is being developed and they prevent the program from compiling correctly. Programs compile correctly when running

>gcc myprogram.c

gives no error or warning messages.

Programs may be turned in for credit when they are partially complete but all completed functions must run without any errors. A program completes without runtime errors if it ends only when the user selects for the program to end and it correctly prints the exit message. Partially complete programs that accept input and that run correctly for each implemented menu choice will receive partial credit.

Any other type of ending is a *runtime error* or a *"crash"*. A program has a runtime error if it compiles and runs but then crashes in any situation, i.e. if there is any set of choices the user can make that will make the program crash. It is your responsibility to test all possible choices in your program to make sure that none of them cause a runtime error. The goal of creating input test data and running your program with it should be to test all of the various choices in your program to make sure all of them terminate correctly.

**Labs which have errors in them and do not terminate normally will receive an overall grade of 0 (zero)**

Grading Scale:

Code: (57%)

Program header and function headers for all functions and

Comments (line comments and block comments) and

Style (indentation, consistency, meaningful identifiers, lateral separation of

code from line comments, etc.) (6 points)

Modularity (division of the problem into small tasks, each one assigned to its own function and

called from main() or from another function when appropriate

--do not code the entire program in main!) (3 points)

Correct use of multiple program files for main, struct definitions, and other functions (3 pts)

Correct definition and manipulation of the struct type with pointers for staff (7 pts)

Correct manipulation of the linked list of staff structs (6 pts)

Correct recursive binary search algorithms using array of structs (6 pts)

Correct recursive merge sort algorithms using array of structs (6 pts)

Correct staff pay calculation using data in linked list (6 pts)

Correct modification/rewriting of menu functions to handle linked lists of structs (3 pts)

Correct function structure as required (3 points)

Correct implementation of other algorithms as given (3 points)

Proper implementation of file data input and input error checking (5 points)

Output: (43%)

Input file name and contents correctly defined and used (3 points)

List tasks perform correctly (4 points)

Calculate pay tasks perform correctly (6 points)

Binary search tasks perform correctly (5 points)

Merge sort tasks perform correctly (5 points)

Other functions perform correctly (3 points)

All tests from testing plan shown in output (3 points)

Input verification shows valid values and list of inputs correctly saved and printed (4 points)

Output gives clear information to explain the values to the user (5 points)

Output contains all the given test data and one additional data set (5 points)

Extra Credit:

Design document development order is well thought out (2 pts XC on lab)

Grading Deductions:

Use of global variables will result in an overall grade of 0 (zero) [-100 deduction]

Use of the exit, break (outside a switch), continue command, or

other non-allowed items will result in an overall grade of 0 (zero) [-100 deduction]

Labs which cannot be compiled or do compile with warnings

will receive an overall grade of 0 (zero) [-100 deduction]

Labs which have errors in them and do not terminate normally

will receive an overall grade of 0 (zero) [-100 deduction]

Missing design document will result in an overall grade of 0 (zero) [-100 deduction]

Missing test plan document will result in 50 point deduction [-50]

Late submission of softcopy of code and/or script file to appropriate TA

will result in an overall grade of 0 (zero)

UNLESS student has obtained prior instructor approval [-100 deduction]

Use of C language elements not yet discussed in class by the lab due date

will result in potential deduction of points - discuss with instructor before using.

Miscellaneous:

Dr. T's Sample input:

500.00 250.00 69.99 50.00 15000.00 9 1 2014 4 X A N O 65 500 150 325 35.50 2.00 22.00 7.25 .75 .90 1.00 .90 3 M H S 2 1 5 120 48.8 76.4 27.50 4 S I B L 3 3 2 3 120.00 75.50

0 0 20 25 40.0 60.0 Hardy\_Shrubs

0 0 100 10 20.0 20.0 Native\_Shrubs

0 0 0 65 20.0 20.0 Hardy\_Trees

0 400 0 75 55.5 20.0 Annuals

65 0 0 10 10.0 19.3 Perennials

0 0 0 140 45.5 35.3 Flowers

10.5 10.0

9.5 10.0

15.0 10.0

10.0 5.0

20.0 10.0

45634543 S 20.90 173

12387665 H 25.50 200

17668532 S 18.35 177

59366820 S 15.00 169

23685669 M 31.09 161

76543987 S 12.78 140

43211223 M 29.60 165

65647355 S 13.11 155

Your test data should be drawn from real data, i.e. a real garden store.