Lab #3 Assignment

CSE 1320-002

Fall 2012

Design Document Due Date: **Tuesday, October 23, 2012 at noon (12pm)**

Lab Due Date: **Tuesday, October 30, 2012 at noon (12pm)**

(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 as covered in class and through work on previous labs. It is assumed that the student is familiar with the programming concepts but not the C syntax. This lab will introduce students to Dr. T’s approach to lab assignments and to reinforce and refresh previously learned material about C and programming.

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:

Algorithms

One-dimensional Array

Multi-dimensional Arrays

Simple structs

Strings

Control structures

Functions

Arithmetic and relational operators

Input and output,

Passing parameters

Data types

Global constants

Pre-processor directives

Searching

Sorting

Program design

Modular programming structure

File input

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 rest of the semester you will be trying to determine what it would cost to start a small brick-and-mortar bookstore. 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 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), continue, passing by reference (except arrays), or linked lists. 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 #3:

1) Strings may be single words or phrases with blanks and the maximum length will be 100 chars. Use constants to define the max value.

2) In general, 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. If the user does not follow your instructions to enter a number then any data type input error is the USER’s fault and you will not be penalized in this lab. There may be specific places where you ARE required to handle data checking but these are specified.

3) You will have a maximum number of 25 rooms (MAXROOMS) and 50 employees (also called staff) (MAXSTAFF) for Lab #3. These values will change in later labs.

4) For date checking in Lab #3, you may make the simplifying assumption that all months have 30 days. This will change in later labs.

Problem: The data in your database will be the bookstore 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 bookstore information that you have researched to use as your test data. After getting all the input data from the user (and storing it in arrays), you will allow the user to do the following:

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. Total estimated square footage of bookstore

iii. Staff costs only with given values

iv. Stock with greatest total amount in the store

v, Stock which will give the greatest profit if all sold

C) List

i. All the types of stock that will be sold with their average prices and starting amounts

ii. Types of stock with average prices less than a given value

iii. Types of stock with amounts greater than some given value

D) Determine scenarios:

i. Calculate total yearly expected profit on 100% sales (Scenario 1)

ii. Calculate total yearly expected profit on 80% sales (Scenario 2)

iii. Calculate total yearly expected profit using a formula (Scenario 3)

iv. Calculate total yearly expected profit using a different formula (Scenario 4)

E) Search for

i. Largest room in the store

ii Largest concentration of stock for sale in any one room

iii. Given net profit by binary search

F) Sort

i. Room names

ii. Total gross and net profit for each room

iii. Room sizes

iv. Room number

2) End the program

Data Description: For lab #3, you will be storing the input data in some single arrays, some multi-dimensional arrays, and 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, the cost of a business permit, and an amount to have as a start up fund.

b) Type of stock (New books, Used books, Music, Video, Audiobooks, Other misc.) - an integer of either 1, 2, 3, 4, 5, or 6 followed by that many letters of either N,U, M, V, A, or O. Must offer either new books or used books 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 U A N O

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, 3.50 12.00 22.00 10.50

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 costs – there will be four types of staff members - manager, owner, floor staff, stockroom staff. For each of these four staff types you must get the hours per month as fractional values, number of each type of staff (no partial people), and average hourly wage for each staff type as a money amount. A total yearly staff cost will be calculated from this info.

f) Type of rooms in the store (Sales room, sTockroom, Bathroom, bReak room, Office) - an integer of either 1, 2, 3, 4, or 5 followed by that many letters of either S, T, B, R, and/or O. Must have sales (S), stockroom (T), 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 T B O

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 one sales room, one stockroom, 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) Amount of stock items of each type in each room – for each room in the store, you will be asking the user for how many of each type of book are in that room. You can make assumptions about rooms with no books if you want to but make sure to write comments about this assumption in your code and to fill in the array (described later) appropriately for those rooms. This is different from Lab #2.

j) Square foot rental rate per year – a floating point money amount that is the amount that one square foot of space rents for. Ex: 500.00 to represent $500.00 per square foot per year.

k) Opening date – Date as day, month, and year - three separate numbers

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

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

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

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

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

q) Room number – This will be assigned by the program but stored as a value in the room array to allow for sorting and re-sorting

r) Profits per room – For each room in the store that has stock, this array will hold the gross profit and net profit for the room. These values are calculated by the program based on the data input for each room, the stock types and amounts in each 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 format for each piece of data is listed below.

Overall you are required to use at least three multi-dimensional arrays (data content and sizes will be specified for each one), one array holding strings, at least two one dimensional arrays, one struct, and then enough one dimensional and multi-dimensional arrays to hold the remaining data. [OK, this isn't as confusing as it seems. ☺]

As an example, lets look at some of the room data for a bookstore. You will have store information about what the room is used for (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 two dimensional array to store data about the rooms. Each row will be one room. Each column will represent different data about that room in that array. Since we will also be naming the rooms, then in the string array, the name of each room will be on the same row as the data about that room in the 2D array. ((Chapter 9 in the book talks about multi-dimensional arrays.))

You will have a maximum number of 25 rooms (MAXROOMS) and 50 employees (also called staff) (MAXSTAFF) for Lab #3.

This information then dictates the sizes of the various types of arrays you will need. You will create 4 multi-dimensional arrays, including an array of strings, and 6 one-dimensional arrays. Some of the arrays hold integer data, some hold float data, and some hold character data. For the 3 one-dimensional character arrays, there will be corresponding one-dimensional integer arrays and the same row always refers to the data in the corresponding array. For the multi-dimensional float and integer arrays, the number of columns is determined by how many pieces of data have to be stored. The number of rows is the same as the maximum number of stock types or rooms or employees that can be stored. 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* but this array is used differently than in Lab #2. 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: NEW = 0, USED = 1, AUDIOBOOK = 2, MUSIC = 3, VIDEO = 4, OTHER = 5. The three columns in the array will represent AVAILABLE = 0, AVGPRICE = 1, and MARKUP = 2. The AVAILABLE column will be used to represent whether a particular kind of stock is available at that store or not. So if the store sells new books, then *stockinfo*[NEW][AVAILABLE] will be set to 1.0 to represent that this is true. If music is not sold at this store, then *stockinfo*[MUSIC][AVAILABLE] will be set to 0.0 to represent false.

If the AVAILABLE column is true, then values should be stored in the AVGPRICE and MARKUP columns. If the AVAILABLE column is 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.}

The second and third arrays will hold amount information about the numbers of types of staff members and the numbers of each room type. These arrays will be one-dimensional integer arrays. Declare constants to use for the indices, ex. MANAGER = 0, OWNER = 1, etc. for the four staff types and the five room types. Inside the array, store the amount (number of) items associated with that type. These amounts are requested from the user. As an example, if the user says there are three stock rooms then the value for *roomtype*[STOCKROOM] should be set to 3. {You may make two separate arrays *roomtype* and *stafftype* or you can just use one array *srtype*. Be sure to declare your constants correctly for whichever array(s) you are using.} Error check the values that are stored in these arrays so that the user does not exceed the maximum number of rooms or staff members.

The next array will be a floating point array of staff data. This array has four rows and four columns. The rows will be indexed by the constants for the staff types, MANAGER, etc., while the columns will be indexed by AVAILABLE (as used before), HRWAGES = 1, MOHOURS=2. This array will hold the average hourly wages for each of the four job types and the average number of hours worked a month for each job type. This will be used with the data in the amount array to determine the total yearly staff cost.

The next room array is a three dimensional floating point array. You used the name *roomsize* for this array in Lab #2 but now we are going to add some more data items in columns to this array. This three dimensional array will continue to contain the length and width but will now also include information about the room area in square footage, how much stock of each type is in each room and a room number as a value also. This array will have five layers (room type) with 23 rows (each room of a particular type) with 10 columns (holding the 6 stock types, length, width, area and room number). The room type will use the constants already declared, like STOCKROOM. The rows will be counted from 0 to *roomtype*[index]-1 where *roomtype*[index] is the value for the current room type, for ex. if *roomtype*[STOCKROOM] is 3, then the meaningful values for the second index of *roomsize* are 0, 1, and 2 and all the rows after 2 would not be used because there are only three stockrooms.

The third index for *roomsize* will use the constants for the stock types as its values plus two new constant values for LENGTH=6 and WIDTH=7, AREA=8, along with ROOMNUM=9. The data stored in each element of the array then is the amount of a particular kind of stock stored in one specific room of a particular room type, then the length and width, and then a copy of the original row counter which will be the room number as counted from 0 to *roomtype*[index]-1. *roomsize*[STOCKROOM][1][NEW] would contain the number of new books stored in stockroom 1 and *roomsize*[SALES][5][OTHER] would contain the amount of other items in sales room 5. This replaces the amount information in the stocktype array in Lab #2. Note that even though the array can hold floating point numbers, the stock values should not have fractional parts, i.e. you won't have half (.5) of an audiobook for example. The values for length and width stored in this array 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 *roomsize*[OFFICE][2][WIDTH] should be 8.5 and *roomsize*[OFFICE][2][LENGTH] should be 9.33. The area of the room should be calculated automatically be the program and stored in the AREA column for that room. The final data in the array is the room number. This is a copy of the original row 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 multi-dimensional array for strings will store one string per row. The string array will have 23 rows and 100 columns, *salesroom[23][100]*. You will be using entire rows of the array rather than single characters, i.e. *salesroom[k]* will be the string that contains the name of the *k*th sales room. The value of *k* will match the values of the second index in the *roomsize* when the first index is [SALES]. Make sure to declare all your constants (such as max string length) before you try to use.

A two dimensional array is needed for storing the profit from each room, *roomprofit*. The array should store gross profit, i.e. all the money paid for stock by customers, and net profit, which is the gross profit minus the actual cost (the AVGPRICE) for each stock item. This array will have 25 rows and 4 columns minimum. The first column will store the room type (SALESROOM, BREAKROOM, etc.), the second column stores the room number, the third row stores the gross profit from all the stock in that room assuming 100% sales, and the fourth column contains the net profit. You may add columns for things like total cost of stock in the room or other data for each room if desired.

All of the data in this *roomprofit* array 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 this array with data. Note that this array only has 25 lines because the store has a max of 25 rooms. However, the rooms are all the different types which is why you must store both room type and room number as data in this array. 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

In addition to the arrays described above you may use more of them if desired for the remaining data and you may add additional columns to any of the defined arrays to store more data in them.

In Lab #3 you will also define 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. When asking for data directly from the user, you may get data from the user in whatever order makes sense. However, you must implement at least two of the data input methods described later in the document for "individual data method", "line of data method", and "file input method".

*Start up costs:*

*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:*

*Stock type* – a single character or a group of characters representing the kind of stock that will be sold. They must match one of the valid stock type codes. For each valid code, a one (1) should be stored in the array *stocktype*[*type*][AVAILABLE] 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 *stocktype*[NEW][AVAILABLE].

*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 *stocktype* 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 *stocktype* array at the [*type*][MARKUP]where *type* is the constant stock type value that matches the current type.

*Staff data:*

*Staff type and amount* - a single character or a group of characters representing the kind of staff members that work at the store. They must match one of the valid staff type codes. For each valid code, the user should be asked for an amount of staff members of that type. This value should be stored in the array for staff data at *stafftype*[*type*] where *type* is the constant staff type value that matches the entered code. Ex. If user enters M as one of the characters, then the user should be asked for how many managers there are and this value should be stored in the staff array at *stafftype*[MANAGER] (or *srtype*[MANAGER] if using a single array).

*Staff hourly wage* - a floating point money amount that is the average hourly wage paid to a staff member of the current type. This value is stored in the floating point multi-dimensional staff array at the [*type*][HRWAGES]where *type* is the constant staff type value that matches the current type.

*Staff hours per month* - a floating point amount that is the average number of hours that a staff member of the current type works in a month. This value is stored in the floating point multi-dimensional staff array at the [*type*][MOHOURS]where *type* is the constant staff type value that matches the current type.

*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 R as one of the characters, then the user should be asked for how many breakrooms there are and this value should be stored in the room array at *roomtype*[BREAKROOM] (or *srtype*[BREAKROOM] if using a single array).

*Square foot rental rate per year –* a floating point value that will be used with the roomsize to work out the yearly rental cost for the store.

*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. This data is stored in the *roomsize* array. Each room will be denoted by the first two indices in the array giving room type and room number. The room type will be one of the defined types and the room numbers will be assigned values from zero (0) to (*roomtype*[*type*] – 1). The third index in the array indicates if the value being stored is the length or width. Ex: If there are two bathrooms in the store, i.e. *roomtype*[BATHROOM] = 2, then the user would have to enter the following length and width info for the bathrooms – *roomsize*[BATHROOM][0][LENGTH], *roomsize*[BATHROOM][0][WIDTH], *roomsize*[BATHROOM][1][LENGTH], and *roomsize*[BATHROOM][1][WIDTH]. The bathrooms would correspondingly be numbered 0 and 1. 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 sales and 1 stock room, 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 *roomsize* array.

*Room stock content* – this is an integer value giving the amount of a given stock type in each room of the store. This data is stored in the *roomsize* array. For all the types of items sold in the store, some items of each of those types might be in any 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 in the *roomsize* array. Ex: If the store sells new books, music, and video, 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 sales room has music and video in it, then you will get and store 0 in *roomsize*[SALES][2][NEW] and get a value, ex. 500, for music to be stored in *roomsize*[SALES][2][MUSIC] and an amount to be stored in *roomsize*[SALES][2][VIDEO].

You are allowed to make assumptions about rooms with no stock (like BATHROOM or BREAKROOM [although when I worked in a bookstore, there WERE books in the break room]) but make sure to write comments about this assumption in your code and to fill in the array 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 sales room 3 and get length, width, amount of new books, amount of used books, amount of music, amount of video, amount of audiobooks, and amount of other (if the store sells all of these types) and then go on to ask about sales room 4.

*Room number* – this is an integer counter value that is stored in the *roomsize* array. When you are getting the data from the user, you will be asking them about the data for SALESROOM 2, for example. In addition to getting the length, width, area, and stock data for the room, make sure you also store the number of the room in the array, in this case 2. This will later allow you to sort the array by amount of NEW 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 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 100 chars in length then it should be copied to *salesroom[number]* in the multi-dimensional array where *number* is the number of the sales room with that name. Ex: The name of sales room 5 which is the audiobook room might be the string "Sounds like…" and it would be stored in *salesroom*[5] in 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.]

*Room profit* – none of the data for this array is directly read from the user. It is all combined or calculated from other data that has been input.

# Data entry phase

Input Implementation: For Lab #3 the user must enter at least 5 rooms (see earlier requirements) and no more than 15 rooms. The user must enter at least 8 staff members and no more than 20. The user must have at least three kinds of products in their store. 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.

Your program must loop for each room to read and store input. You also have the choice to make two separate loops; one to enter all of the size data for all rooms and then one to enter all stock data for all rooms.

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

For Lab #3, you must implement at least two of the following three methods of input. The first method will prompt the user for each piece of data individually, the second method will allow the user to input all needed pieces of data about one event on one line plus the room info, and the third method will use the same format as the line of data input method but the data will be read from a file instead of from the keyboard. The three forms of the input are described below. The program must ask the user how they want to input the data. Your program must give them at least two choices of methods: Individual data method, line of data input method, or file method. Once the user has chosen the method of input, all the input will be done that way for the current run of the program.

Individual data method:

Note: For the individual data method, you do not have to follow these steps exactly in this order although you can if you wish. The steps are to explain what is meant by "individual data method". You may also wish to tell the user what group of data you are going to ask about such as the "start up cost" data or the "room" data. The goal is for the user to clearly understand what to do.

a. Ask the user for the business license cost. Read in their number, check to make sure it is valid and then save this in a variable in the start up struct. {Think about what "valid" means in the context of a cost.}

b. Ask the user for the total cost of connecting up all the utilities. Read in their number, check to make sure it is valid and then save this in a variable in the start up struct.

c. Ask the user for the business permit cost. Read in their number, check to make sure it is valid and then save this in a variable in the start up struct.

d. Ask the user how much money they wish to have in their start up fund. Read in their number, check to make sure it is valid and then save this in a variable in the start up struct.

e. Ask the user for the month, day, and year for the opening day of the store. Make sure it is a valid date, i.e. it is a real date and it is in the future, and then store it in three int variables in a date struct. Remember to declare the constants if you use them.

f. Give the user the list of choices for the stock types. There are two ways you can get the stock type info:

f.1. Ask the user to enter the number of types of stock there are followed by the letter of each type. Use the array *stocktype*[*type*][AVAILABLE] to store a 1 in each matching type. Ex: If the user 3 U M A, then the *types* USED, MUSIC, and AUDIOBOOKS should have a value of 1.

f.2. OR Ask the user for the stock type letter and give them the choices. Check to make sure it is a valid type. Store a 1 in the *stocktype*[*type*][AVAILABLE] array in the appropriate type for this valid character.

g. For each kind of stock*,* loop and do the following (the *loop counter* should be less than the total number of *types* of stock):

If *stocktype*[*loop counter*][AVAILABLE] equals 1 (the store has this kind of stock)

Then

1. Ask the user for the average cost of one item of this stock type. Check that they entered 0 or a positive number and then store it in the array *stocktype*[*loop counter*][AVGPRICE] *.*

2. Ask the user what percentage of markup do they want on this type of item. Verify that the value entered is valid and store the valid value at *stocktype*[*loop counter*][MARKUP]

Else if *stocktype*[*loop counter*][AVAILABLE] equals 0

Ignore this stock type and increment to the next

h. Handle the staff data input in the same fashion as the stock input. First find out how many types and which types of staff there are and store an amount for each kind of staff. Store in *stafftype*[*type*] or *srtype*[*type*]. Then for each kind of staff, if that kind works at this store get the average hourly wage, stored in the staff array at [*type*][HRWAGES] and get the average hours per month worked for each staff type and store in the staff array at the [*type*][MOHOURS].

i. The room data input starts in the same way as the stock and staff input. Give the user a list of the types of rooms, find out which rooms the store has and how many of each. Store in *roomtype*[*type*].

j. Ask the user for the rental rate per square foot of the store per year. Save this value in a variable.

k. For each kind of room*,* loop and do the following (the *loop counter* should be less than the total number of *types* of room):

If *roomtype*[*loop counter*] is 1 (the store has this type of room)

Then For each room in the amount of rooms of this type, loop and do the following (the *loop 2 counter* should be less that the value of *roomtype*[*loop counter*]):

1. Ask the user for the length of the current room in feet and inches. Convert the value and then check that they entered 0 or a positive number and then store it in the array *roomsize*[*loop counter*][*loop 2 counter*][LENGTH] *.*

2. Ask the user for the width of the current room in feet and inches. Convert the value and then check that they entered 0 or a positive number and then store it in the array *roomsize*[*loop counter*][*loop 2 counter*][WIDTH] *.*

For each type of stock, loop and do the following (the *loop 3 counter* should be less than the total number of *types* of stock):

If *stocktype*[*loop 3 counter*][AVAILABLE] equals 1 (might have this stock)

Then Ask the user how many items of the current stock type are

in the current room and

store at *roomsize*[*loop counter*][*loop 2* *counter*][*loop 3 counter*].

[Don't forget to store the room number (from the *loop 2 counter*) and the area calculated from LENGTH and WIDTH]

If current room type is SALES,

Then Ask the user for the string name of the venue.

Read it into a temporary string and check the length.

If it is OK, append the room number onto the name string

Then store it into the name string in *salesroom[loop 2 counter]*

Line of data input method:

Your second input method is to have all the necessary data entered by the user on one 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 #3, use the line of data method with inputs in exactly the order specified below.

The data for the 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}. Here's an example:

prompt>500.00 250.00 50.00 15000.00 1 12 2012 3 U M A 4.50 11.99 12.25 .75 .90 1.00 2 M F 2 10 15.00 9.00 160.0 76.0 4 S T B O 2 1 2 1 27.60

40.0 60.0 6500 0 0 Main room

20.0 20.0 500 750 200 Media room

20.0 20.0 1200 150 50

10.5 10.0

9.5 10.0

15.0 10.0

which represents a business license cost of $500.00, a utility connection cost of $250, a business permit cost of $50, and a start up fund of $15,000. The opening day is Dec. 1, 2012. The store will have three types of stock – used books, music, and audiobooks. The average costs of these stock types is $4.50 for used books, $11.99 for music, and $12.25 for audiobooks. The markup is 75% for used books, 90% for music, and 100% for audiobooks. There are two kinds of staff members – managers and floor staff. There are two managers and 10 floor staff. Managers make $15.00 per hour and average 160 hours per month. Floor staff make $9.00 per hour and average 76 hours per month. There are 4 types of rooms in the store and the following amounts – sales (2), stockroom (1), bathrooms (2), office (1). The rental rate per square foot per year is $27.60. [For this example, I assume that bathrooms and offices 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 remaining data is one line for each room. The order of the lines matches the order of the rooms given so 2 sales rooms, 1 stockroom, 2 bathrooms, and 1 office is 6 lines. For sales rooms there is length, width, amount of first type of stock, amount of second type of stock,…,amount of 6th type of stock (if used), then room name. In the example above, the first sales room is 40' by 60', it has 6500 used books, no music, and no audiobooks, and it is named "Main\_room". Likewise for the second sales room called "Media\_room". The third room is the 20' x 20' stockroom with 1200 used books, 150 music items, and 50 audiobooks. The 4th room is the first bathroom of size 10.5' by 10', the second bath is 9.5' x 10', and the office is 15' x 10'.

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 read all the numbers, chars, and the string one 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 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. When you have multiple values of the same sort in a row, you should loop to read and store them.

File input method:

Your third 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 for the line of data method. For Lab #3, your data file must be named "Lab3input.txt" and must be in the same directory as your C program. 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.

Input verification:

When the user has entered all the store and room 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 number 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: (use any number scheme you wish)

A) Print all the data about the store

B) Print all the data about business costs

C) Print all data for stock

D) Print all data for staff

E) Print all data for rooms (entered by user and calculated, if any)

F) Calculate estimated square footage of bookstore

G) Calculate staff costs only for one year using the given input values

H) Calculate total estimated first year costs

I) Calculate the stock with greatest total amount in the store

J) Calculate which stock will give the greatest gross profit if all (100%) of the stock is sold

K) List all types of stock with average prices less than a given value

L) List all rooms with square footage larger than a given amount

M) Determine profit scenario 1 (see description below)

N) Determine profit scenario 2 (see description below)

O) Determine profit scenario 3 (see description below)

P) Determine profit scenario 4 (see description below)

Q) Search for largest room in the store by linear search

R) Search for largest amount of one type of stock for sale in any room

S) Search for a given net profit by binary search

T) Sort the sales room names array by name by bubble sort

U) Sort the sales room names array by room number appended at end of string

V) Sort each type of room (layer) in the *roomize* array by total area from largest to smallest

W) Sort each type of room (layer) in the *roomize* array by room number

X) Sort the room profits by gross profit by merge sort

Y) Sort the room profits by net profit

Z) Sort the room profits by room type and number

AA) End the program

The print functions A – E let 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-square-footage function F has the following steps:

For every type of room,

loop through each individual room's data to

get length and width

multiply L x W to get area of that room

sum the areas of all the rooms of this type

sum the areas for each type of room

Total sum is total store square footage

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

For each type of staff member in the store

Multiply number of staff

times hours per month

times hourly wage to get monthly staff cost

Sum the monthly staff cost for each type of staff

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

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

Sum all the startup costs

Find the yearly staff cost

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 square foot rental rate per year

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

Function I, calculate-greatest-amount, has the following steps:

Using the data in the *roomsize* array,

For each type of stock

For each layer

For each room type

For each room in that type

Sum all the amounts of the current stock type

Save the total for this stock type

Use the stock type totals to find the greatest amount of any one type of stock

Function J, calculate-greatest-gross-profit, has the following steps:

Using the stock totals from the calculate-greatest-amount function,

Use AVGPRICE and MARKUP to calculate the gross profit for each stock type

Use the gross profit values to find the greatest profit of any one type of stock

The list-all-stock-less-than-price function should ask the user to enter an average price for any item of stock then go through every type of stock and print the complete information for all types that have an average price equal to or less than the user’s entered price.

The list-all-rooms-greater function 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 or greater than the user’s entered value.

The profit-scenario-1 function has the following assumptions and steps:

*Assumptions:*

Given the monthly amounts of stock, assume that 100% of stock is sold every month.

*Steps – Calculate the profit for one year with the above assumption (scenario 1)*

Calculate total stock sold of each type

Use average price of each stock type

Times markup percentage to determine the amount of profit for the item type

Calculate total monthly profit for each stock type using the above

assumption for how much of stock is sold

Multiply this value times months of the year to get yearly profit for scenario 1

The profit-scenario-2 function has the following assumptions and steps:

*Assumptions:*

Given the monthly amounts of stock, assume that 80% of stock is sold every month.

*Steps – Calculate the profit for one year with the above assumption (scenario 2)*

Follow the same steps as for scenario 1

The profit-scenario-3 function has the following assumptions and steps:

*Assumptions:*

Given the monthly amounts of stock, assume the following is sold every month:

Lowest price stock type sells 100%

Highest price stock sells 70%

All other types of stock sell 85%

*Steps – Calculate the profit for one year with the above assumption (scenario 3)*

Find the highest and lowest average prices for types of stock

Follow the same steps as for scenario 1

The profit-scenario-4 function has the following assumptions and steps:

*Assumptions:*

Assume that the sum of all the average prices should be 100%

Determine what percent each average price is of the total amount

Using the percentages for the average prices, assign that percentage of sales in the OPPOSITE order to the average prices

Ex: The average price of an item of type NEW is 11.99, USED 4.75, and MUSIC 13.95.

11.99 + 4.75 + 13.95 = 30.69 and the percentages associated with each average price are

39% + 15% + 46% = 100%. Then to assign them in the opposite order we get

USED items are 46% of sales, NEW items are 39% of sales, and MUSIC is 15% of sales.

*Steps – Calculate the profit for one year with the above assumption (scenario 4)*

Determine how much of each kind of stock is sold

Follow the same steps as for scenario 1

Search-for-largest-room in the store by linear search:

Using the data in the *roomsize* array,

For each layer

For each room type

For each room number in that type

If the area of the current room is larger than the current largest room,

Then save the room type, room number, and area of the new largest room

Search-for-largest-amount of one type of stock for sale in any room

Using the data in the *roomsize* array,

For each type of stock

For each layer

For each room type

For each room number in that type

If the amount of the current stock in the current room is larger than the current largest amount,

Then save the room type, room number, stock type and amount of the new largest stock amount

Search-for–nearest-profit by recursive binary search

Using the data in the *roomprofit* array,

Sort the array from smallest to largest by net profit value

Get a net profit value from the user

A. Check middle array value and make sure array has one or more values

B. If too large, calculate positive difference and save difference and index value.

C. Using smaller half of array, go back to step A

D. If too small, calculate negative difference and save difference and index value.

E. Using larger half of array, go back to step A

F. If exact value found, return current index

G. Else if array has one or fewer values,

H. If absolute value of positive difference is smaller than absolute value of negative difference

I. Return index of positive difference

J. Else Return index of negative difference

Sort the sales room names array by name by bubble sort

Using the data in the *roomname* array and the bubble sort algorithm discussed in class, this function should

Have a loop to make all the needed passes through the array to insure that it is sorted,

Have an inner loop that does all the comparisons to go once through the array,

Compare each adjacent pair of names to alphabetize the list,

Swap the names if they are out of order

NOTE: to swap the names, you'll need a temp string and strcpy

Sort the sales room names array by room number appended at end of string

Using the data in the *roomname* array, this function should

Sort the strings in the list based on the room number that you appended to the end of each name.

To get the room numbers, you will need to look at the last "word" in the string and then

convert it to a number. See the sample code posted by Dr. T for this.

You may use any sorting algorithm you wish.

Sort each type of room (layer) in the *roomsize* array by total area from largest to smallest

Using the data in the *roomsize* array,

Within each layer, i.e. within each type of room, ex. *roomsize[BREAKROOM]*

Sort the rooms by the area of the room, *roomsize[roomtype][roomnumber][AREA]* from largest to smallest

You may use any sorting algorithm you wish.

Sort each type of room (layer) in the *roomsize* array by room number

Using the data in the *roomsize* array,

Within each layer, i.e. within each type of room, ex. *roomsize[BREAKROOM]*

Sort the rooms by the room number, ex. *roomsize[roomtype][roomnumber]* from smallest to largest

This puts the array back in the original order

You may use any sorting algorithm you wish.

Sort the room profits by gross profit by merge sort

Using the data in the *roomprofit* array,

Sort the array by the gross profit column, ex. *roomprofit[GROSS]*from largest profit to smallest profit

You must sort using the recursive merge sort as discussed in class and in the book.

Sort the room profits by net profit

Using the data in the *roomprofit* array,

Sort the array by the net profit column, ex. *roomprofit[NET]*from largest profit to smallest profit

You may use any sorting algorithm

Sort the room profits by room type and number

Using the data in the *roomprofit* array,

Sort the array by the room type and room number columns, ex. *roomprofit[roomtype]*and *roomprofit[roomnumber]*

You will need to sort the rooms by type first and then

Within each room type,

Sort by room number from smallest to largest

You may use any sorting algorithm

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), continue, variable strings (except as described), passing by reference (except arrays), pointers, or linked lists.

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

Implementation Requirements:

For Lab #3 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.

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), and c) 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 #3 Design Document must be turned in in order for your Lab #3 assignment to be graded.

The program should use the following data structures:

Multi-dimensional integer, floating point and string arrays, single-dimensional character arrays

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

variable strings (except as described)

exit

break (except in a switch)

continue

pointers

passing by reference (except arrays)

linked lists.

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 similar to those used in the examples in the textbook. This header should include at least the function name, the purpose of the function, 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, two for getting input, one for printing, and all of the functions listed for the user choices. 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!

The program should 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 from the keyboard

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 4 kinds of stock, 3 kinds of staff, and 5 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.**

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 run correctly for all implemented menu choices 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:

TBD

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, pointers, or linked lists 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]

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 50.00 15000.00 1 12 2012 3 U M A 4.50 11.99 12.25 .75 .90 1.00 2 M F 2 10 15.00 9.00 160.0 76.0 4 S T B O 2 1 2 1 27.60

40.0 60.0 6500 0 0 Main room

20.0 20.0 500 750 200 Media room

20.0 20.0 1200 150 50

10.5 10.0

9.5 10.0

15.0 10.0

Your test data may be drawn from real data, i.e. a real bookstore, if you want to research it.