Lab #2 Assignment

CSE 1320-002

Fall 2012

Design Document Due Date: **Sept. 25, 2012 at noon (12pm)**

Lab Due Date: **Oct. 2, 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 covered in Chapters 1–4 and parts of later chapters in Foster and Foster as covered in class. 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

 Strings

 Control structures

 Functions

 Arithmetic and relational operators

Input and output,

Passing parameters

 Data types

 Global constants

 Pre-processor directives

 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 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, structs, passing by reference (except arrays), or linked lists. You should do input and output using printf and scanf (you are allowed to use getchar and putchar if desired.)

Assumptions for the lab #2:

1) Strings will be single words with no blanks and the maximum length will be 40 chars. 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. 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.

3) You will have a maximum number of 25 rooms (MAXROOMS) and 50 employees (also called staff) (MAXSTAFF) for Lab #2. These values will change in later labs.

4) For date checking in Lab #2, 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

 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)

2) End the program

Data Description: For lab #2, you will be storing the input data in some single arrays and some multi-dimensional arrays.

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 and amount 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 then followed by that many numbers indicating amounts of each type of stock. 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 6500 500 15000 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, 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) 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.

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

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

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

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

n) Sales room name – a single word with no blanks and less than 40 characters long that gives a name for each of the sales rooms. In lab #2 we are using strings for the name but we are making the string input simpler by insuring that all strings are one word, i.e. Children’s\_Reading\_Room is one word because there are underscores instead of blanks.

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 words (strings), at least two one dimensional arrays, 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 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 #2.

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 others may hold character data. For arrays related to the same data, 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.

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: NEW = 0, USED = 1, AUDIOBOOK = 2, MUSIC = 3, VIDEO = 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 new books, then *stockinfo*[NEW][AMOUNT] will be store the amount 15000 which represents the amount of new books (from the earlier example). If music is not sold at this store, then *stockinfo*[MUSIC][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.}

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 use 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 final array is related to room data. The floating point array will be a three dimensional array that contains the information about the length and width of each room, *roomsize*. This array will have five layers (room type) with 23 rows (each room of a particular type) and two columns, one for LENGTH and one for WIDTH. The values stored in this array should be measurements in fractional feet of the length and width of the current room. 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. 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 . {Think about why Dr. T says that you only need 23 rows in this array when the max number of rooms is 25…}

The multi-dimensional array for strings will store one string per row. The string array will have 23 rows and 40 columns, *salesroom[23][40]*. 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 *roomstock* when the first index is [SALES]. Make sure to declare all your constants (such as max string length) before you try to use.

In addition to the arrays described above you may use more of them if desired for the remaining data.

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 to two data input methods described later in the document for "individual data method" and "line of data 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.

*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.

*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.

*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.

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

*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 *stocktype*[*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 *stocktype*[NEW][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 *stocktype* 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 *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 may have the user enter fractional feet directly for each measurement OR you may 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.]

*Sales room name* – a single word with no blanks and less than 40 characters long that gives a name for each sales room that has one.. The string will be read in with the %s specifier in a scanf statement into a temporary string. The length of the string should be checked using the *strlen* command. If the string is less that 40 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.]

# Data entry phase

Input Implementation: For Lab #2 the user must enter at least 3 rooms (see earlier requirements) and no more than 10 rooms. The user must enter at least 3 staff members and no more than 20. The user must have at least two 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.

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.

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

For Lab #2, you must implement the following two methods of input. The first method will prompt the user for each piece of data individually and the second method will allow the user to input all needed pieces of data about one event on one line. The two 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 both choices of methods: Individual data method or line of data input 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. For this lab you may NOT use file input or any other method of entering data from other than the keyboard.

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. {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.

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.

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.

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 either in three int variables or in an integer array at *openingday[DAY]* and then in MO and YR. 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*][AMOUNT] 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. After this step, use a loop for step g.

 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*][AMOUNT] array in the appropriate type for this valid character. After the user enters a stock type, do the g. step for that type then repeat step f.2. to get the next stock type.

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*][AMOUNT] equals 1 (the store has this kind of stock)

Then

1. Ask the user for the amount available of items of this stock type. Check that they entered 0 or a positive number and then store it in the array *stocktype*[*loop counter*][AMOUNT] *.*

2. 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] *.*

3. 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]

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 greater than or equal to 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 fractional feet or in feet and inches. Convert the value if necessary 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 fractional feet or in feet and inches. Convert the value if necessary 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] *.*

 If current room type is SALES,

 Then Ask the user for the single word name of the venue.

Make sure to indicate that blanks are not allowed.

Read it into a temporary string and check the length.

If it is OK, 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 #2, 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 6500 4500 500 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 Main\_room

20.0 20.0 Media\_room

20.0 20.0

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 with the amounts – 6500 used books, 4500 music items and 500 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, then room name (if appropriate). In the example above, the first sales room is 40' by 60', and it is named "Main\_room". Likewise for the second sales room called "Media\_room". The third room is the 20' x 20' stockroom. 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 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 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.

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

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

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

K) Determine profit scenario 1 (see description below)

L) Determine profit scenario 2 (see description below)

M) Determine profit scenario 3 (see description below)

N) 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

The list-all-stock-less-than-price function I 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 J 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 K 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 L 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 M 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

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, files, the exit command, goto, break (except in a switch), continue, structs, 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 #2 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 #2 Design Document must be turned in in order for your Lab #2 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:

 structs

 global variables

files

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 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.**

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:

Code: (66%)

 Program header and function headers for all functions (6 points)

 Comments (line comments and block comments) (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!) (6 points)

 Style (indentation, consistency, meaningful identifiers, lateral separation of code from line comments, etc.) (5 points)

 Correct manipulation of the 2-dimensional integer and float arrays (6 points)

 Correct manipulation of the 3-dimensional arrays (6 points)

 Correct use of required control structures (6 points)

 Correct manipulation of the 2-dimensional char array for names (3 points)

 Correct function structure as required (8 points)

 Correct implementation of algorithms as given (6 points)

 Proper implementation of data input and input error checking (8 points)

Output: (34%)

 User clearly understands what is being requested for input (5 points)

 Find-all tasks perform correctly (6 points)

 Calculate and scenario tasks perform correctly (9 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)

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, structs or linked lists will result in an overall grade of 0 (zero) [-100 deduction]

 Use of file input or any input other than keyboard input for the two input methods will result in a grade of 0 (zero)

 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 6500 4500 500 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 Main\_room

20.0 20.0 Media\_room

20.0 20.0

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.