Lab #4 Assignment

CSE 1320-002

Fall 2012

Design Document Due Date: **Thursday, November 22, 2012 at noon (12pm)**

Lab Due Date: **Thursday, November 15, 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:

 Pointers

Structs

 Strings

 C date and time

 Control structures

 Functions

 Algorithms

 One-dimensional Array

 Multi-dimensional Arrays

 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. In Lab #4 your design document must include some additional information to indicate the order in which you plan to work on and develop your various code modules.

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, or linked lists. You should do input and output using printf, scanf, fprintf, fscanf, and getline (you are allowed to use getchar and putchar if desired.)

Assumptions for the lab #4:

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 #4. These values will change in later labs.

4) For date checking in Lab #4, you must use real calendar calculations.

5) There will be a maximum of 100 unique stock items in this lab. Each stock item will be of one of the existing stock types.

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. Type of stock with greatest total amount in the store

 v, Stock item which will give the greatest profit per item

 C) List

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

 ii. Items of stock with actual prices less than a given value

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

 iv. All staff members with starting dates earlier than some given value

 D) Determine revised scenarios:

 i. Calculate total yearly expected profit on 100% sales (Scenario 1)including rent, staff, and startup costs

 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

 iv. Employee with highest wage rate

 v. Most expensive stock item

 F) Sort

 i. Room names

 ii. Total gross and net profit for each room

 iii. Room sizes

 iv. Room number

 v. Employee total pay for current year

 vi. Stock items by name

 vii. Stock items by price

 viii. Stock items by type and name

 G) Sell items

 H) Restock items

2) End the program

Data Description: For lab #4, you will be storing almost all the input data in arrays of structs.

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

a) Start up costs – a money amount composed of the cost of a business license, the total cost of the utility connection fees, 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

cd) Stock item – for Lab #4, each unique item will have its own representation. Each item will fall into one of the stock types above and will then also have a stock item name, a stock item number, an actual cost for this item, a markup amount for this item, a total current inventory amount of this item, and the monthly recurring restock amount for this item. So if the store has Harry Potter books, then there could be a stock item that has the data: "Harry Potter and the Sorcerer's Stone", stock number 1234, stock type N, actual cost 3.00, markup amount 2.5, a total inventory of 20, and each month another 5 copies are bought for the store. Another item might be "Hagrid's motorcycle stuffed toy", 2466, O, 4.50, 1.25, 8.

e) Staff types – there will be four types of staff members - manager, owner, floor staff, stockroom staff. For each type of staff, there will be an integer representing the number of staff members there are on the payroll of this particular type.

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 stock item 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.

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. 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.

s) Staff data – This will include employee first name, last name, date hired, type of employee, employee number, do they have an hourly wage versus a monthly salary for wage type, hourly pay rate OR monthly salary amount, employee birthday, employee address, and hours worked in current month.

The format for each piece of data is listed below.

The format of the data is the major change in this lab assignment. To hold the data that was previously in a variety of arrays, you will now be creating struct types and having arrays of those struct types. You should use the same constants you have used previously for stock, staff, and room types where appropriate. (Look back at previous labs if you need to refresh your memory on these types.)

The first new struct type will be for stock. The *struct stockitem* will contain all the data needed about any one kind of stock item as described above at cd). The required members of the struct are listed above but you may add additional information for the *struct stockitem* type if desired.

The *struct stockitem* variables should be stored in an array. There is a maximum of 100 unique stock items for this lab.

The second new struct type will be for rooms. The *struct roomdata* will contain all the data needed about any one room. This includes room name (if it has one), room type, room number, length, width, area, rental rate, amount and stock item number of each kind of item in the room, profits per room, any other data required, and any data you would like to keep about each room. The stock information for a room is different than before because now there are possibly 100 different stock items which could be in the room. You will need a two dimensional array (*roomstock)* inside the *struct roomdata* in order to hold the stock item number for any item in the room and the amount of that item in the room.

The *struct roomdata* variables should be stored in an array. There is still a maximum of 25 rooms in the whole store. You should use the *roomtype* array from your earlier labs to hold the numbers of each type of room in the store just as was done before. However, all the rooms are now stored in the one array of *struct roomdata* variables instead of in the other multi-dimensional arrays.

The third new struct is *struct staffdata*. This struct will hold the information described in s) above. You will still use the *stafftype* array from your earlier labs to hold the numbers of each type of staff. However, now all the staff information is stored in one array of *struct staffdata*. There is still a maximum of 50 staff members.

In addition to the structs and arrays described above you may use more of them if desired for any data that does not fit with one of the given struct types and you may add additional members to any of the struct types to store more data in them.

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

Detailed descriptions of each data value. The data values below must be stored as described. You may also store additional values into the strucs if desired.

It is NOT necessary to have the user enter the data in exactly the order listed if you are doing individual data entry. When asking for data directly from the user, you may get data from the user in whatever order makes sense. However, you must implement the required 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. The date struct should be a component of the startup cost 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 item count –* an integer count listing the number of different items that are in the store of the current stock type. This is different than the previous use of amount which lumped all items of the same type together and gave a total. The count will indicate how many unique items there are of the current type. Ex: If the store only carries Harry Potter books as the only New stock items then the item count for NEW would be 10 (The seven HP books plus the 3 extra little books like "The Tales of Beedle the Bard"). There is a maximum of 100 unique items total for the combined stock types for Lab #4. This item count is stored in the *stocktype* array and replaces the value of AVAILABLE if a one was stored there initially.

*Each unique stock item data:*

*[These values are stored in the stockdata struct variables]*

*Stock item type* – value indicating which category of stock this item belongs to. (NEW, USED, etc.)

*Stock item name –* a string giving the title of any book, music or video item or a text description of other items.

*Stock item number* – a unique number that represents one particular item. There should be no duplicate stock numbers. ((You do not have to error check uniqueness in Lab #4 but do tell your user.))

*Stock item cost per item* - a floating point money amount that is the actual cost for one item of the current item number.

*Desired profit amount (markup) per stock item* - a floating point number between 0 and 3 inclusive. This limits the profit to 300% of the cost, i.e. if the actual 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.

*Stock inventory* – the current number of this item in the store

*Monthly stock purchase amount* – the number of copies of this item bought for the store monthly

*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).

*Each unique staff member data:*

*[These values are stored in the staffdata struct variables]*

*Staff first name* – a string with the employee's first name. The string must not have blanks.

*Staff last name* – a string with the employee's last name. The string must not have blanks.

*Hire date* – a date struct inside the *staffdata* struct that holds the date the employee was hired. This date must be in the past and this must be error checked against the current date which is gotten from the computer.

*Staff member type* – value indicating which category of staff this employee belongs to. (MANAGER, etc.)

*Staff member employee number* – a unique number that represents one particular staff member. There should be no duplicate employee numbers. ((You do not have to error check uniqueness in Lab #4 but do tell your user.))

*Pay type* – a value indicating if this employee is paid an hourly wage or a monthly salary. Each employee can only be paid one way. Managers and owners are paid a monthly salary regardless of the specific number of hours worked. Floor and office staff are paid an hourly wage per hour worked each month.

*Staff hourly wage* - a floating point money amount that is the hourly wage paid to the current staff member if this person is paid hourly.

*Staff monthly salary* - a floating point money amount that is the monthly salary paid to the current staff member if this person is paid monthly.

*Staff birthdate* – a date struct inside the *staffdata* struct that holds the date the employee was born. This date must be in the past and this must be error checked against the current date which is gotten from the computer.

*Staff address* – an address including at least a street name, street number, city, state, country, and postal code. The address should also include at least one phone number for the employee. You may create a struct type for this if desired and use the type inside the staff struct 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, there should be 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).

*Each unique room data:*

*[These values are stored in the roomdata struct variables]*

*Room type* – value indicating which category of room this belongs to. (SALESROOM, etc.)

*Room number* – this is an integer counter value that is stored in the *roomdata* struct. When you are getting the data, you will be looking for 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 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 the name string in the *roomdata* struct. Ex: The name of sales room 5 which is the audiobook room might be the string "Sounds like…" and it would be stored in the *roomdata* struct corresponding to Salesroom 5.

*Square foot rental rate per year –* a floating point value that will be used with the length and width 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 *roomdata* struct. Once you have gotten the length and width for a room, you should calculate the area and also store that in the *roomdata* struct.

*Room stock content* – This will give the amount and stock item number of each kind of item in the room. The stock information for a room is different than before because now there are possibly 100 different stock items which could be in the room. You will need a two dimensional integer array (*roomstock)* inside the *struct roomdata* in order to hold the stock item number for any item in the room and the amount of that item in the room. Since the stock item numbers are unique, this will be enough information for your program to get the remaining stock info from the stock array to do profit calculations and determine how many items are in the room of any one stock type. 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.]

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.

*Room profit* – The values for gross profit and net profit should be calculated and stored in the room array.

# Data entry phase

Input Implementation: For Lab #4 the user must enter at least 6 rooms (see earlier requirements) and no more than 25 rooms. The user must enter at least 15 staff members and no more than 50. The user must have at least three types of products in their store and within these three types there must be at least 3 stock items of each type. As this data is entered, 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 indicate how this issue should be handled.

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 #4, you MUST implement file input and either individual input or line of data input . The individual input method will prompt the user for each piece of data individually, the line of data method will allow the user to input all needed pieces of data about one event on one line plus the room info, and the file input 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: File method (required) and either 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.

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 ask the user for the number of unique items of this stock type are in the store. Check that they entered 0 or a positive number and then store it in the array *stocktype*[*loop counter*][AVAILABLE] to replace the 1 *.*

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

 Ignore this stock type and increment to the next

g.1. Now create a double loop to get the data for each stock item. One loop will increment through the different stock types.

If there are no items of this type in the store (from the data in *stocktype*[*type*][AVAILABLE]) then skip this type and go to next

Else, there are items of this type so have the second loop increment up to *stocktype*[*type*][AVAILABLE]

 Inside the loop, get the stock data of name, item number, item type, actual cost, markup amount, current inventory, and monthly purchase amount. Store this data into the current *stockdata* struct in the stock array.

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*].

h.1. Now create a double loop to get the data for each staff member. One loop will increment through the different staff types.

If there are no employees of this type in the store (from the data in *stafftype*[*type*][AVAILABLE]) then skip this type and go to next

Else, there are employees of this type so have the second loop increment up to *stafftype*[*type*][AVAILABLE]

 Inside the loop, get the staff member first name, last name, staff type, employee number, hire data, type of pay, pay amount, hours worked per month, birthdate, staff member address and phone. Store this data into the current *staffdata* struct in the staff array.

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*].

i.1. Now create a double loop to get the data for each room. One loop will increment through the different room types.

If there are no rooms of this type in the store (from the data in *roomtype*[*type*][AVAILABLE]) then skip this type and go to next

Else, there are rooms of this type so have the second loop increment up to *roomtype*[*type*][AVAILABLE]

 Inside the loop, get 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 current *roomdata* struct in the rooms array. Do the same for width, rental rate, room stock content (using the array that can hold 100 stock items), room name if any, room type, and room number (or assign this). Calculate and store area, gross profit, and net profit.

File input method and Line of data input method:

Your file or line of data input method is to have all the necessary data entered by the user on a few lines. The input would be all the pieces of data listed above. You must tell the user exactly how to enter the line of data. For Lab #4, use the file input and/or 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 N M A 10 4 3 2 M F 2 10 4 S T B O 2 1 2 1

124456 N 4.50 2.3 8 5 Harry Potter and the Philosopher's Stone

124457 N 4.50 2.3 4 3 Harry Potter and the Chamber of Secrets

124458 N 5.00 2.0 3 3 Harry Potter and the Prisoner of Azkaban

124459 N 5.00 2.2 3 3 Harry Potter and the Goblet of Fire

124460 N 5.50 2.2 1 3 Harry Potter and the Order of the Phoenix

124461 N 6.00 2.0 2 3 Harry Potter and the Half-Blood Prince

124462 N 6.50 2.0 2 3 Harry Potter and the Deathly Hallows

124531 N 2.25 2.5 2 2 Fantastic Beasts and Where to Find Them

124532 N 2.25 2.5 2 2 Quidditch through the Ages

124533 N 3.00 2.0 3 2 The Tales of Beadle the Bard

392554 M 5.55 2.75 4 2 Sargent Pepper's Lonely Hearts Club Band

392563 M 5.50 2.5 2 2 The White Album

392571 M 5.60 2.55 2 3 Rubber Soul

392588 M 5.40 2.6 2 2 Yellow Submarine

814456 A 6.0 1.5 2 1 Harry Potter and the Philosopher's Stone

814461 A 6.10 1.4 2 2 Harry Potter and the Half-Blood Prince

814462 A 6.0 1.7 2 3 Harry Potter and the Deathly Hallows

Kate Daniels F 6540 10 2 2010 H 10.00 160 7 5 1985 45 Trail Ave, Savannah, GA 31244 912-465-2020

Andrea Kane F 4734 9 9 2010 H 12.00 160 9 23 1984 342 White St, Atlanta, GA 30345 404-833-1889

Raphael Ware F 7344 4 5 2011 H 8.50 80 3 3 1980 224 Peachtree St, Atlanta, GA 30365 404-929-3444

Beatrice Ware M 2453 4 1 2008 S 3750.00 160 5 6 1960 65 Green St, Atlanta, GA 30365 470-908-3332

Jim Bent F 5545 7 2 2009 S 1654.75 160 8 3 1979 445 Suter Dr, Atlanta, GA 31136 470-598-2571

Dali Niesan F 7393 3 15 2012 H 7.35 80 8 23 1992 2022 6th Ave, Decater, GA 31876 678-223-3565

Brenna Loop F 6399 9 4 2010 H 9.95 120 12 15 1990 213 Forest St, Atlanta, GA 31154 404-929-7285

Julie White F 4392 11 1 2009 S 2200.00 160 2 9 1979 1600 8th St. Smyrna, GA 32566 470-713-6545

Derek Sims M 3327 3 8 2007 S 3405.50 160 4 14 1972 42 Ninth Ave, Roswell, GA 32245 678-432-4378

Rowena Gale F 4530 1 15 2010 H 11.10 160 5 13 1988 1600 8th St. Smyrna, GA 32566 470-843-3343

Peter Weiss F 6772 5 22 2011 H 8.00 80 11 29 1987 3498 Macon Dr, Atlanta, GA 31167 404-452-9345

Red Moore F 8128 9 1 2012 H 7.25 40 5 6 1994 892 Penn Ave, Marietta, GA 34276 770-334-8765

S 1 40 0 60 0 22.00 Main room

124456 4 124457 2 124458 2 124459 2 124460 1 124461 1 124462 2 124533 2 124532 2 124531 1

S 2 20 6 20 8 22.00 Media room

392554 3 392563 2 392571 2 392588 2 814456 1 814461 1 814462 2

T 1 20 0 20 0 18.00

124456 4 124457 2 124458 1 124459 1 124461 1 124533 1 124531 1 392554 1 814456 1 814461 1

B 1 10 6 10 0 25.00

B 2 9 6 10 0 25.00

O 1 15 3 10 4 18.00

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 – new books, music, and audiobooks. The stock includes 10 different new books, 4 different music items, and 3 different audiobooks. There are two kinds of staff members – managers and floor staff. There are two managers and 10 floor staff. There are 4 types of rooms in the store and the following amounts – sales (2), stockroom (1), bathrooms (2), office (1).

Each stock item listed stock number (124456), stock type (N), actual item cost (4.50), markup (2.3), copies in inventory (8), copies to order each month (5), and title ("Harry Potter and the Philosopher's Stone"). There are 10 N books, 4 M items and 3 audiobooks listed.

Following the stock there is staff data which is listed as first name with no spaces ("Kate"), last name with no spaces ("Daniels"), staff type (F), employee number (6540), hire date (10 2 2010 = Oct. 2, 2010), pay type either hourly or salary (H = hourly), hourly pay rate (10.00), number of hours worked per month (160), birthdate (7 5 1985), street number with no spaces (45), street name which is everything up to the comma ("Trail Ave"), city name up to next comma (Savannah), two letter state abbreviation (GA), 5-digit zip code (31244), 10-digit phone number (912-465-2020). There are 10 floor staff and two managers as listed earlier. Note that they are not in any particular order.

Following the staff data is the room data. Each room has two lines of info. The first line is room type (S), room number (1), room length as feet and inches (40 0), room width as feet and inches (60 0), rental rate per square foot (22.00), room name if one is given ("Main room"). The second line is a list of stock items and amounts in the current room listed as item number (124456) then amount of that item (4) for every item in the room. This line should be read until you encounter the end of the line which will indicate that the stock entry is complete for that room.

All the preceding data is typed with single spaces between the items on a 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.)

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

File input method additional information:

This input method is to have all the necessary data entered by the user from a file. The input would be all the pieces of data listed above and would use the format specified above. For Lab #4, your data file may have any name but MUST be in the same directory as your C program. Be sure to tell the user the name for the data file so that they can run the program with their own data file.

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

Input verification:

When the user has entered all the store 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 type of stock (N, U, Etc) with greatest total amount in the store

J) Calculate which individual stock item will give the greatest gross profit on each item

K) List all individual stock items with actual prices less than a given value

L) List all stock items with inventory amounts larger than a given amount

L2) List all staff members with starting dates earlier than some given value

M) Determine profit scenario 1 (see description below). Revised to use each stock item.

N) Determine profit scenario 2 (see description below). Revised to use each stock item.

O) Determine profit scenario 3 (see description below). Revised to use each stock item.

P) Deleted

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

S2) Search for employee with the highest wage rate

S3) Search for most expensive stock item

T) Sort the rooms array by name by bubble sort

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

V) Sort each room in the *rooms* array by total area from largest to smallest

W) Sort each room in the *rooms* array by room number

X) Sort the rooms by gross profit by merge sort

Y) Sort the rooms by net profit

Z) Sort the staff by employee total pay for the year

Z2) Sort stock items by name

Z3) Sort stock items by actual cost

Z4) Sort stock items by price charged, i.e. actual cost plus cost markup amount is price charged.

Z5) Sort stock items by type and then by name within their type

Z6) Sell items

Z7) Restock items

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 room,

get length and width

multiply L x W to get area of that room

sum the areas of all the rooms

 Total sum is total store square footage

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

 For each staff member in the store,

 If staff member paid hourly,

 Multiply staff hourly wage

times hours per month to get monthly staff cost

Else if staff member paid salary, use monthly salary value

 Sum the monthly staff cost for each staff member

 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 item of stock in the store

 Multiply amount of items of stock

 Times actual 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 *stock* array,

 For each item of stock

 Add its amount to the appropriate stock amount for its 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 data in the *stock* array,

 Use actual cost and markup to calculate the gross profit for each stock item

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

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

The list-all-stock-amounts-greater function should ask the user to enter the an inventory amount of a stock item then go through every stock item and print the complete information for all items that have an inventory amount equal to or greater than the user’s entered value.

The list-all-staff-starting-before function should ask the user for a date then go through each staff member and print the complete information for each staff member who has a hiring date earlier than the given date.

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

*Assumptions:*

Given the monthly amounts of stock (this is the amount of stock that is purchased of that item each month), assume that 100% of stock is sold every month.

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

 For each stock item

 Get monthly amount and determine amount sold (based on this scenario)

 Use actual price of each stock item

 Times the amount of items sold

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

 Calculate total monthly gross profit as the sum of each of the item profits

 Multiply this value times months of the year to get yearly gross 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 item type sells 100%

 Highest price stock item 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 actual prices for the items of stock

Follow the same steps as for scenario 1

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

 Using the data in the *rooms* array,

 For each room

 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 *rooms* array,

 For each type of stock

 For each room

 Sum the amount of stock of each type in the rooms (all the N items for example)

 If the amount of the current stock type 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 *rooms* 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

Search-for-highest-wage in the store by linear search:

 Using the data in the *staff* array,

 For each staff member

 If the staff member is paid by hourly wage,

 Then if the wage of the current staff is larger than the current largest wage,

 Then save the employee number and the employee wage

Search-for-most expensive stock item in the store:

 Using the data in the *stock* array,

 Calculate price for each stock item

 Find the largest price using any algorithm

Sort the rooms array by name by bubble sort

Using the data in the *rooms* 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 rooms array by room number appended at end of string

Using the data in the *rooms* 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 room in the *rooms* array by total area from largest to smallest

Using the data in the *rooms* array,

 Sort the rooms by the area of the room, *rooms[i].area* from largest to smallest

 You may use any sorting algorithm you wish.

Sort each room in the *rooms* array by room number

Using the data in the *rooms* array,

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

 [Note that you will also have to sort the room types to totally reorganize it unless

 you numbered the rooms in a different way than the sample data]

 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 *rooms* array,

 Sort the array by the gross profit, ex. *rooms[i].grossprofit* 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 *rooms* array,

 Sort the array by the net profit, ex. *rooms[i].netprofit* from largest profit to smallest profit

 You may use any sorting algorithm

Sort the staff by employee total pay for the year

Using the data in the *staff* array,

 Determine the total yearly pay for each employee, then

 Sort the array by total yearly pay

 You may use any sorting algorithm

Sort the stock array by name by bubble sort

Using the data in the *stock* 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 stock array by actual cost of an item

Using the data in the *stock* array,

 Sort the array by the item cost, ex. *stock[i].actualcost* from largest to smallest cost

 You may use any sorting algorithm

Sort the stock array by price of an item

Using the data in the *stock* array,

 Determine the price charge for the item; i.e. actual cost plus cost markup amount

 Ex. *stock[i].price =* *stock[i]. actualcost + stock[i].markup\* stock[i]. actualcost*

 Sort the array by the item price, ex. *stock[i].price* from largest to smallest price

 You may use any sorting algorithm

Sort stock items by type and then by name within their type

Using the data in the *stock* array,

 Sort the items by stock type,

 Within each stock type,

 Sort the items by name

Sell items:

Using the data in the *stock* and *rooms* arrays,

 Let the user select an item to purchase and a room to purchase from

 "Sell" the item by reducing the *stock* item inventory amount and the *rooms* amount of that stock item

Restock items:

Using the data in the *stock* and *rooms* arrays,

 Let the user select an item to restock, an amount to restock, and a room to put stock in

 "Restock" the item by increasing the *stock* item inventory amount and the *rooms* amount of that stock item

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. This lab has additional requirements for the design document so make sure you read them.

 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, or linked lists.

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

Implementation Requirements:

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

 a) all the functions you expect to write,

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

 c) some indication of which function calls what other functions, and

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

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

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

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

The program should use the following data structures:

 Single-dimensional struct arrays and other structures as needed.

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

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

 #define EVENTMAX 10

 const int EVENTMAX = 10;

The program should use the following control structures:

 Function calls to perform tasks

 A selection structure to choose input style

 A while or for loop to read the input data

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

The program should NOT use:

 global variables

goto

 exit

 break (except in a switch)

 continue

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.

One good guideline for function size is to keep functions between 50 and 200 lines long unless the longer function content is truly obvious. If you find yourself writing virtually the same code in multiple places, then that code would probably make a good function which can be called in multiple places.

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 10 items of stock, 8 members 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 have given you a small amount of sample data 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:

Coding: (55%)

Program header and function headers for all functions, and

Comments (line comments and block comments), and

Style (indentation, consistency, meaningful identifiers,

lateral separation of code from line comments, etc.) (7 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!) (5 points)

Correct definition of all struct types including all new data elemens (5 pts)

Correct declarations of arrays of structs (2 pts)

Correct manipulation of struct array member data of all types (5 pts)

Correct passing of struct arrays to functions (4 pts)

Correct implementation of file input and error checking (6 pts)

Correct implementation of second method input and error checking (3 pts)

Correct implementation of required algorithms –

linear search, binary search, bubble sort, merge sort, and calculations (8 pts)

Correct modification/rewriting of menu functions to handle arrays of structs (6 pts)

Correct implementation of functions for sell and restock (4 pts)

Output: (45%)

User clearly understands what is being requested for input (3 points)

Find-all (List) tasks perform correctly (3 points)

Calculate tasks perform correctly (3 points)

Linear searches perform correctly (3 pts)

Binary searches perform correctly (5 pts)

Bubble sorts perform correctly (4 pts)

Merge sorts perform correctly (6 pts)

File input performs correctly (4 pts)

Input verification shows valid values and list of inputs correctly saved and printed (3 points)

Output gives clear information to explain the values to the user (6 points)

Output contains all the given test data and one additional data set (3 points)

Test data is meaningful and of correct size (2 pts)

Extra Credit:

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

Grading Deductions:

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

 Use of the exit, break (outside a switch), continue command, or 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 or late submission of design document

will result in an overall grade of 0 (zero) UNLESS

student has obtained instructor approval by e-mail [-100 deduction]

 Missing or late submission of softcopy of code and/or script file

will result in an overall grade of 0 (zero) UNLESS

 student has obtained instructor approval by e-mail [-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:

[This is a sample data set that can be put in a file as test data. Make sure the file is plain text only.]

500.00 250.00 50.00 15000.00 1 12 2012 3 N M A 10 4 3 2 M F 2 10 4 S T B O 2 1 2 1

124456 N 4.50 2.3 8 5 Harry Potter and the Philosopher's Stone

124457 N 4.50 2.3 4 3 Harry Potter and the Chamber of Secrets

124458 N 5.00 2.0 3 3 Harry Potter and the Prisoner of Azkaban

124459 N 5.00 2.2 3 3 Harry Potter and the Goblet of Fire

124460 N 5.50 2.2 1 3 Harry Potter and the Order of the Phoenix

124461 N 6.00 2.0 2 3 Harry Potter and the Half-Blood Prince

124462 N 6.50 2.0 2 3 Harry Potter and the Deathly Hallows

124531 N 2.25 2.5 2 2 Fantastic Beasts and Where to Find Them

124532 N 2.25 2.5 2 2 Quidditch through the Ages

124533 N 3.00 2.0 3 2 The Tales of Beadle the Bard

392554 M 5.55 2.75 4 2 Sargent Pepper's Lonely Hearts Club Band

392563 M 5.50 2.5 2 2 The White Album

392571 M 5.60 2.55 2 3 Rubber Soul

392588 M 5.40 2.6 2 2 Yellow Submarine

814456 A 6.0 1.5 2 1 Harry Potter and the Philosopher's Stone

814461 A 6.10 1.4 2 2 Harry Potter and the Half-Blood Prince

814462 A 6.0 1.7 2 3 Harry Potter and the Deathly Hallows

Kate Daniels F 6540 10 2 2010 H 10.00 160 7 5 1985 45 Trail Ave, Savannah, GA 31244 912-465-2020

Andrea Kane F 4734 9 9 2010 H 12.00 160 9 23 1984 342 White St, Atlanta, GA 30345 404-833-1889

Raphael Ware F 7344 4 5 2011 H 8.50 80 3 3 1980 224 Peachtree St, Atlanta, GA 30365 404-929-3444

Beatrice Ware M 2453 4 1 2008 S 3750.00 160 5 6 1960 65 Green St, Atlanta, GA 30365 470-908-3332

Jim Bent F 5545 7 2 2009 S 1654.75 160 8 3 1979 445 Suter Dr, Atlanta, GA 31136 470-598-2571

Dali Niesan F 7393 3 15 2012 H 7.35 80 8 23 1992 2022 6th Ave, Decater, GA 31876 678-223-3565

Brenna Loop F 6399 9 4 2010 H 9.95 120 12 15 1990 213 Forest St, Atlanta, GA 31154 404-929-7285

Julie White F 4392 11 1 2009 S 2200.00 160 2 9 1979 1600 8th St. Smyrna, GA 32566 470-713-6545

Derek Sims M 3327 3 8 2007 S 3405.50 160 4 14 1972 42 Ninth Ave, Roswell, GA 32245 678-432-4378

Rowena Gale F 4530 1 15 2010 H 11.10 160 5 13 1988 1600 8th St. Smyrna, GA 32566 470-843-3343

Peter Weiss F 6772 5 22 2011 H 8.00 80 11 29 1987 3498 Macon Dr, Atlanta, GA 31167 404-452-9345

Red Moore F 8128 9 1 2012 H 7.25 40 5 6 1994 892 Penn Ave, Marietta, GA 34276 770-334-8765

S 1 40 0 60 0 22.00 Main room

124456 4 124457 2 124458 2 124459 2 124460 1 124461 1 124462 2 124533 2 124532 2 124531 1

S 2 20 6 20 8 22.00 Media room

392554 3 392563 2 392571 2 392588 2 814456 1 814461 1 814462 2

T 1 20 0 20 0 18.00

124456 4 124457 2 124458 1 124459 1 124461 1 124533 1 124531 1 392554 1 814456 1 814461 1

B 1 10 6 10 0 25.00

B 2 9 6 10 0 25.00

O 1 15 3 10 4 18.00