Shopping App

Online shopping applications abound nowadays. These provide a convenient way for buyers to browse at different options and to purchase items without going to the usual brick-and-mortar stores. These applications also provide a venue for budding entrepreneurs to sell their products without need to pay rent and with reach to many more potential buyers.

For your project, you are to create a shopping program that facilitate the buying and selling of items. This also entails that the program should maintain (keep track of) the set of at most 100 users, the set of at most 20 items per seller, and the set of transactions. Below are the descriptions of what each of these should contain.

Users – Users can be buyers and/or sellers. Each user information has a numeric userID, a string of at most 10 characters for the password, a string of at most 30 characters for the address, a numeric contact number, and a string of at most 20 characters for the name.

Items – Each item has a numeric productID, a string (of at most 20 characters) item name, a string (of at most 15 characters) category, a string (of at most 30 characters) item description, numeric quantity available, numeric unit price, and numeric sellerID (which corresponds to the seller's userID)

Transactions – Each transaction contains a date (which further contains the numeric month, day, and year), the set of at most 5 items, the buyer's user ID, the seller's user ID and total amount of the transaction. It should be noted that though there may be at most 5 items, these may be of different quantity per item and all of these items in the transaction should be from the same seller. The buyer should also not be the seller.

Your shopping program should have the following features and sub-features: [Note that at the beginning of the program, the file Users.txt and Items.txt, if existing, should be loaded to the main memory for processing. Note that format of the text files are listed under Exit.]

1. Register as a User

   Once this option is chosen, the user is tasked to input all the necessary information. For simplicity, the userID is also given by the user as input. However, this userID should be unique (that is, no other such userID exists in the list of users). This information should be added to the array of Users. After registering, the user is redirected back to the main menu.
2. User Menu

   Once this option is chosen, the user is asked to input his userID and password. If userID and password does not match any registered user, there should be an error message and the user is redirected back to the main menu. Upon successful log-in, the user is presented with the following options:

   2.1 Sell Menu

   2.1.1 Add New Item

   In this option, the user can add a new item that he wants to sell. Information for the item should be encoded by the user (including product ID), except the seller ID – this is supposed to be automatically stored since the user is logged in. The product ID should be unique regardless of seller. To ensure that there is no monopoly of the market, each seller can only sell at most 20 different items. Thus, the program should check that the seller has not reached the limit yet before allowing the user to add an item to sell. Only 1 item can be added every time this option is chosen. The data is stored in the Items array. Afterwhich, the user is shown the Sell Menu again.

   2.1.2 Edit Stock

In this option, the set of all of this seller's products is first shown in table format (see Show My Products for how it is supposed to be displayed). Then, asks the productID whose information is to be edited.

If an invalid productID is given, a message should be shown, then the user is redirected back to the Sell Menu.

If a valid productID is given, the following submenu is repeatedly offered to the user until the user chooses to finish editing.

2.1.2.1 Replenish

The user is tasked to input the quantity that will be ADDED to the existing quantity for this item.

2.1.2.2 Change Price

The user is asked to input the new unit price.

2.1.2.3 Change Item Name

The user is asked to input the new item name.

2.1.2.4 Change Category

The user is asked to input a new category.

2.1.2.5 Change Description

The user is asked to type a new description.

2.1.2.6 Finish Editing

The user is redirected back to the Sell Menu

2.1.3 Show My Products

The information should be shown in this sequence: productID, item name, category, unit price, quantity. The data should be shown in table format, sorted in increasing order based on the productID. Numbers should be right justified, strings should be left justified in their "columns". Hint: use the capacity of the containers to determine the size of the "columns" and refer to CCPROG1 notes on printf().

2.1.4 Show My Low Stock Products

All the information (including product description, but excluding seller's userID) about each product whose quantity is BELOW 5 should be shown. Display should be done one at a time per product. Allow the user to press N to see the next and press X to exit the view.

2.1.5 Exit Sell Menu

The user is directed back to the User Menu.

2.2 Buy Menu

Once this option is chosen, the program checks the current directory if there is a previously saved cart for this user (see Exit User Menu below) and loads the contents of the binary[1] file to main memory for possible processing.

2.2.1 View all Products

The program displays all items from 1 seller at a time, sorted in increasing order based on seller ID. So, show seller ID first, then followed by a table of the products of that seller. The table is similar as the one discussed in Show My Products.

When the user press N, the next seller ID is shown and under it, the table of products of that seller, and so on. The user can press X to exit this view and go back to the Buy Menu.

2.2.2 Show all Products by a Specific Seller

The program asks the user to input the seller's ID that he wishes to view. Then, the program displays the seller's ID and below it the table of that seller's products (the table is in same format as that discussed in Show My Products).

2.2.3 Search Products by Category

The program asks the user to input the category he wants to see. Product information should be displayed similar to how it is presented in Show My Low Count Products, including allowing the user to type N and X to navigate through the display. Only those products that fit the given category should be displayed.

2.2.4 Search Products by Name

The program asks the user to input keywords of the product he wants to see. Note that substring search is applied here, meaning the keyword the user entered may appear as part of a longer string in the product name of some of the products. For example, if user wants to search for shirt, then ITEM NAMES that contain shirt, like T-shirt, long-sleeved polo shirt, couple shirts should all be displayed. Those that match, the displayed information should be similar to Show My Low Count Products, including allowing the user to type N and X to navigate through the display.

2.2.5 Add to Cart

Each user can have up to at most 10 different items (of varying quantities) in his cart at any given time. If the cart is full already, no additional items can be added to the cart and a suggestion that the user proceed to Edit Cart or Check Out first before adding more items. The user is asked to input the productID and the quantity that the user wants to buy. Note that the productID should be existing. The quantity should also be available. Otherwise, error messages will be displayed and item is not added to the cart. A buyer cannot buy his own product.

2.2.6 Edit Cart

All items in the Cart will be displayed first following the display format described in Show All Products by a Specific Seller (of course, no input seller ID is necessary).

2.2.6.1 Remove all items from Seller

The user is asked to input the seller ID. All items in the cart that is sold by this seller is deleted from the cart.

2.2.6.2 Remove Specific Item

The user is asked to input the product ID. That item in the cart will be removed.

2.2.6.3 Edit Quantity

The user is asked to input the product ID and the new quantity. The item's quantity is updated.

2.2.6.4 Finish Edit Cart

The program goes back to Buy Menu.

2.2.7 Check Out Menu

Get the date as input from the user. [This is for testing and our demo purposes. In real life, the machine's date is used.] Note that one final check is done with the availability (because part of the items to be checked out might have been from previously saved items in the cart and the seller might have updated the information already (like quantity and price). If there is any change, the buyer should be notified by displaying the old and the new quantity and price; the changes should also be updated in the cart. The buyer is given notice (message) that he can still go to Edit Cart.

2.2.7.1 All

All items in the Cart will be bought by the user. Each transaction will contain all items from the same seller only. A display of summary of each transaction should be displayed on the screen. The display should look similar to a receipt, that is in a table format, it should list the following: quantity, product ID, item name, unit price, total price for item. Then below the table, a total amount due for the transaction and payable to sellerID and seller name. Transaction information

should be added to the binary[1] file Transactions.dat. Items already bought should be removed from the cart and product's quantity should also be updated in the Items array. [For simplicity, we omitted the payment facility and delivery facility. We assume that is already done upon check out.]

2.2.7.2 By a Specific Seller

The user is asked to input the seller ID whose items in cart he wishes to buy already. A summary of the transaction is shown, transaction data is added to the binary[1] file, items bought should be removed from the cart, quantity in Items array should be updated. [Refer to All for details.]

2.2.7.3 Specific Item

The user is asked to input the product ID of the item he had included in the cart to check out. A summary/receipt is displayed, transaction is included in the binary[1] file, item should be removed from the cart, and quantity should be updated in Items array.

2.2.7.4 Exit Check Out

The program reverts back to the Buy Menu.

2.2.8 Exit Buy Menu

The user is directed back to the User Menu

2.3 Exit User Menu

In this option, if there are any items left in the user's cart, these will be saved to a binary[1] file of Items with the filename <user's ID>.bag (example: 123.bag). The program then exits the user menu and goes back to the main menu.

3. Admin Menu

The user is assumed to be the administrator of the program. For verification, he is asked to input the password. For simplicity, let us assume that the correct password is "H3LLo?". If invalid log-in, a message "Unauthorized access not allowed" is shown, then program goes back to Main Menu. Upon successful log-in, the following features are available:

3.1 Show All Users

This feature shows all the users arranged by user ID. The display should be in table format in the following sequence: userID, password, name, address, phone number

3.2 Show All Sellers

The program shows all the users who have items to be sold (i.e., if there exists in the Items array a seller ID matching the user's ID). The display should be in table format in the following sequence: userID, password, name, address, phone number, number of items for sale [not quantity]

3.3 Show Total Sales in Given Duration

The program asks the user to input 2 dates to serve as the start and end dates. The program then checks the contents of Transactions.dat whose dates fall within the duration from start to the end dates, the total amount of all the transactions.

3.4 Show Sellers Sales

The program asks the user to input 2 dates to serve as the start and end dates. The program then checks the contents of Transactions.dat whose dates fall within the duration from start to the end dates, display the total sales for each seller in table format in the following sequence: seller ID, seller name, total sales in the duration

3.5 Show Shopaholics

The program asks the user to input 2 dates to serve as the start and end dates. The program then checks the contents of Transactions.dat whose dates fall within the duration from start to the end dates, display the total

amount for each buyer in table format in the following sequence: buyer ID, buyer name, total amount bought in the duration

3.6 Back to Main Menu
The program reverts back to the Main Menu

4. Exit – The program saves into text file all current data on set of users and set of items. Then the program terminates properly. The users should be saved in Users.txt with the following format:

```
<user1 id><space><password><new line>
<name><new line>
<address><new line>
<contact number><new line>
<new line>
<user2 id>><space><password><new line>
<name><new line>
<address><new line>
<contact number><new line>
<new line>
...
<userN id><space><password><new line>
<name><new line>
<address><new line>
<contact number><new line>
<new line>
<eof>
```

The items should be saved in Items.txt with the following format:

```
<product1 id><space><seller id><new line>
<item name><new line>
<category><new line>
<description><new line>
<quantity><space><unit price><new line>
<new line>
<product2 id><space><seller id><new line>
<item name><new line>
<category><new line>
<description><new line>
<quantity><space><unit price><new line>
<new line>
...
<productN id><space><seller id><new line>
<item name><new line>
<category><new line>
<description><new line>
<quantity><space><unit price><new line>
<new line>
<eof>
```

[1] - *All binary file requirements can also be replaced with text file, but the format has to be decided by the student. Note that text file requirements cannot be replaced with binary files.*

Bonus

A **maximum** of **10 points** may be given for features **over & above** the requirements (other features not conflicting with the given requirements or changing the requirements). For example: (1) showing sales report on statistics on top selling items, total sales, and total profit; (2) using top selling items and items with discounts to generate recommended items [top picks] to shoppers; (3) saving transactions from multiple days and comparing the sales of each day] subject to **evaluation** of the teacher. **Required features** must be **completed first** before bonus features are credited. Note that use of conio.h, or other advanced C commands/statements may **not** necessarily merit bonuses.

MP CheckPoint : March 11, 2023 (Saturday). This is a self-check and self-declaration on what has been accomplished so far. It is expected that by this date, at least the following features have been completed and tested already.

- All features of Manage Accounts Menu
- All features of Main Menu
- Add New Stock feature is limited to one Product per Category (at this checkpoint)
- View All Stocks and Browse All Products
- Modify Stock Info
- Restock
- Shutdown Kiosk
- Log out
- Modify User Info

- It is imperative that your implementation has considerable and proper use of arrays, structures, and user-defined functions, as appropriate, even if it is not strictly indicated.

Submission & Demo

- **Final MP Deadline: April 11, 2023, 0800AM via AnimoSpace**. No project will be accepted anymore after the submission link is locked and the grade is automatically 0.

  Requirements: Complete Program

  - Make sure that your implementation has considerable and proper use of arrays, structures, files, and user-defined functions, as appropriate, even if it is not strictly indicated.
  - It is expected that each feature is supported by at least one function that you implemented. Some of the functions may be reused (meaning you can just call functions you already implemented [in support] for other features. There can be more than one function to perform tasks in a required feature.
  - Debugging and testing was performed exhaustively. The program submitted has

    a. NO syntax errors

    b. NO warnings - make sure to activate -Wall (show all warnings compiler option) and that C99 standard is used in the codes

    c. NO logical errors -- based on the test cases that the program was subjected to

Important Notes:

1. Use **gcc -Wall** to compile your C program. Make sure you **test** your program completely (compiling & running).
2. Do not use brute force. Use **appropriate conditional** statements **properly**. Use, **wherever appropriate**, **appropriate loops** & **functions properly**.
3. You **may** use topics outside the scope of CCPROG2 but this will be **self-study**. Goto *label*, exit(), break (except in switch), continue, global variables, calling main() are **not allowed**.

4. Include **internal documentation** (comments) in your program.
5. The following is a checklist of the deliverables:

Checklist:
- ❒ Upload via AnimoSpace submission:
  - ❒ source code*
  - ❒ test script**
  - ❒ sample text file exported from your program
- ❒ email the softcopies of all requirements as attachments to **YOUR** own email address on or before the deadline

Legend:

***** Source code exhibit readability with supporting inline documentation (not just comments before the start of every function) and follows a coding style that is similar to those seen in the course notes and in the class discussions. The first page of the source code should have the following declaration (in comment) [replace the pronouns as necessary if you are working with a partner]:

/**
This is to certify that this project is my own work, based on my personal efforts in studying and applying the concepts learned. I have constructed the functions and their respective algorithms and corresponding code by myself. The program was run, tested, and debugged by my own efforts. I further certify that I have not copied in part or whole or otherwise plagiarized the work of other students and/or persons.

<your full name>, DLSU ID# <number>
**/

Example coding convention and comments before the function would look like this:

```
/* funcA returns the number of capital letters that are changed to small letters
   @param strWord - string containing only 1 word
   @param pCount - the address where the number of modifications from capital to small are
                   placed
   @return 1 if there is at least 1 modification and returns 0 if no modifications
   Pre-condition: strWord only contains letters in the alphabet
*/
int     //function return type is in a separate line from the
funcA(char strWord[20] ,    //preferred to have 1 param per line
      int * pCount)           //use of prefix for variable identifiers
{  //open brace is at the beginning of the new line, aligned with the matching close brace
   int    ctr;        /* declaration of all variables before the start of any statements –
                         not inserted in the middle or in loop- to promote readability */

   *pCount = 0;
   for (ctr = 0; ctr < strlen(strWord); ctr++)  /*use of post increment, instead of pre-
                                                  increment */
   {  //open brace is at the new line, not at the end
      if (strWord[ctr] >= 'A' && strWord[ctr] <= 'Z')
      {    strWord[ctr] = strWord[ctr] + 32;
           (*pCount)++;
      }
      printf("%c", strWord[ctr]);
   }

   if (*pCount > 0)
      return 1;
   return 0;
}
```

******Test Script should be in a table format. There should be at least 3 categories (as indicated in the description) of test cases **per function**. There is no need to test functions which are only for screen design (i.e., no computations/processing; just printf).

Sample is shown below.

Function	#	Description	Sample Input Data	Expected Output	Actual Output	P/F
sortIncreasing	1	Integers in array are in increasing order already	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P

	2	Integers in array are in decreasing order	aData contains: 53 37 33 32 15 10 8 7 3 1	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P
	3	Integers in array are combination of positive and negative numbers and in no particular sequence	aData contains: 57 30 -4 6 -5 -33 -96 0 82 -1	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	P

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the sample code in page 7, the following are four distinct classes of tests:

i.) testing with strWord containing all capital letters (or rephrased as "testing for at least 1 modification")
ii.) testing with strWord containing all small letters (or rephrased as "testing for no modification")
iii.) testing with strWord containing a mix of capital and small letters
iv.) testing when strWord is empty (contains empty string only)

The following test descriptions are **incorrectly formed**:

- Too specific: testing with strWord containing "HeLlo"
- Too general: testing if function can generate correct count OR testing if function correctly updates the strWord
- Not necessary -- since already defined in pre-condition: testing with strWord containing special symbols and numeric characters

6. Upload the softcopies via Submit Assignment in AnimoSpace. You can submit multiple times prior to the deadline. However, only the last submission will be checked. Send also to your **mylasalle account** a **copy** of all deliverables.
7. You are allowed to create your own modules (.h) if you wish, in which case just make sure that filenames are descriptive.
8. During the MP **demo**, the student is expected to appear on time, to answer questions, in relation with the output and to the implementation (source code), and/or to revise the program based on a given demo problem. Failure to meet these requirements could result to a grade of 0 for the project.
9. It should be noted that during the MP demo, it is expected that the program can be compiled successfully and will run. If the program does not run, the grade for the project is automatically 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) and other submissions (e.g., non-violation of restrictions evident in code, test script, and internal documentation) will still be checked.
10. The MP should be an HONEST intellectual product of the student/s. For this project, you are allowed to do this individually or to be in a group of 2 members only. Should you decide to work in a group, the following mechanics apply:

Individual Solution: Even if it is a group project, each student is still required to create his/her INITIAL solution to the MP individually without discussing it with any other students. This will help ensure that each student went through the process of reading, understanding, solving the problem and testing the solution.

Group Solution: Once both students are done with their solution: they discuss and compare their respective solutions (ONLY within the group) -- note that learning with a peer is the objective here -- to see a possibly different or better way of solving a problem. They then come up with their group's final solution -- which may be the solution of one of the students, or an improvement over both solutions. Only the group's final solution, with internal documentation (part of comment) indicating whose code was used or was it an improved version of both solutions) will be submitted as part of the final deliverables. It is the group solution that will be checked/assessed/graded. Thus, only 1 final set of deliverables should be uploaded by one of the members in the AnimoSpace submission page. [Prior to submission, make sure to indicate the members in the group by JOINing the same group number.]

Individual Demo Problem: As each is expected to have solved the MP requirements individually prior to coming up with the final submission, both members should know all parts of the final code to allow each to INDIVIDUALLY complete the demo problem within a limited amount of time (to be announced nearer the demo schedule). This demo problem is given only on the day/time of the demo, and may be different per member of the group. Both students should be present during the demo, not just to present their individual demo problem solution, but also to answer questions pertaining to their group submission.

Grading: the MP grade will be the same for both students -- UNLESS there is a compelling and glaring reason as to why one student should get a different grade from the other -- for example, one student cannot answer questions properly OR do not know where or how to modify the code to solve the demo problem (in which case deductions may be applied or a 0 grade may be given -- to be determined on a case-to-case basis).

11. Any form of **cheating** (**asking other people not in the same group for help**, **submitting as your [own group's] work part of other's work**, **sharing your [individual or group's] algorithm and/or code to other students not in the same group, etc.**) can be punishable by a grade of **0.0** for the **course** & a **discipline case**.

Any requirement not fully implemented or instruction not followed will merit deductions.