Grocery Register Project

This project sets up C code to handle basic tasks at a simple grocery store. This project uses a simplistic hash table to manage inventory in the store

Project Specification

This project implements a system that

1. Manages the inventory system for a grocery store
2. Checks out queues of customers with grocery items
3. Provides a menu (with *specific syntax*) to perform either of the above

Inventory System

Description

Inventory *items* should be represented by the following struct: ●

Item ***- Struct***

○ **Key** ***- Int***

■ The item’s identification number (This will be used as the key) ○ **Name** ***- String***

■ The name describing the item ○ **Threshold** ***- Int***

■ The point at which the store needs to order more of that item ○ **Stock** ***- Int***

■ The number of that item currently in stock ○ **Price -** ***float***

■ The price of that item, in $.

These items are to be read in from and written to a text file, this text file is known as the *inventory record*. For this program the *inventory record* will be `“inventory.txt”`.

You must use a ***hash table*** as defined in the book/slides to use for the *items*. The key for the hash table should be the item’s ***key***. When the program starts up, it should use the *item record* to populate your hash table. The inventory will be used during the checkout procedure. You must use a **struct** as defined in the book/slides for the *items* in the hash table.

Syntax and Examples

When customers check out at the register then the program should check each item the customer has in their *grocery list* and subtract the appropriate number of items from the inventory per item in the list.

When a queue of customers is checked out (this will be done via text file), the system should check if any items fall below their restock threshold. If this occurs, a message should be printed to the console after the queue of customers has emptied. If a customer were to try to buy a depleted item, assume it is an input error. More on this will be in the *check-out section*.

Additionally, the user should be able to manually add, delete, or replenish items in the inventory. They can either add specific numbers of items back to the inventory, or elect to restock each item that is below its threshold back up to that threshold. The syntax for getting to this functionality will be described in the *menu section*.

The syntax for the operations listed above are as follows:

1. `add` *key name threshold stock price*

    i. int - *key* is a valid, unique ***Key*** of an item

    ii. string - *name* is the ***Name*** of the item. iii. int- *threshold* is the restock ***Threshold*** for the item iv. int- *stock* is the current ***Stock*** of the item v. float - *price* is the ***Price*** of the item.

2. `delete` *key*

    i. int - *key* is the ***Key*** of an existing item.

3. `restock` *key num*

    i. int - *key* is a valid, unique ***Key*** of an item ii. int - *num* is the number to increase the stock of said item by.

4. `restock all`

This will be in a (test) file "inventory.txt" that will be in the same directory as your .c file.

```
{101, "Carrots", 5, 20, 1.99}
{102, "Apples", 10, 12, 0.99}
{216, "Swiss Cheese", 5, 6, 2.49}
{039, "Wine", 15, 20, 12.99}
{006, "Coffee", 5, 12, 5.00}
```

This would indicate:

Carrots has **key** 101, 5 is the restock ***threshold***, 20 in ***stock,*** and ***price*** $1.99.

Apples has ***key*** 102, 10 is the restock ***threshold***, 12 in ***stock,*** and ***price*** $0.99.

Swiss Cheese has ***key*** 216, 5 is the restock ***threshold***, 6 in ***stock,*** and ***price*** $2.49.

Wine has ***key*** 039, 15 is the restock ***threshold***, 20 in ***stock***, and ***price*** $12.99. *Coffee* has ***key*** 006, 5 is the restock ***threshold***, 12 in ***stock,*** and ***price*** $5.00.

<u>Sample Add:</u>

```
> add 002 "Limes" 5 10 3.00
```

Adds *item* with ***Key*** 002, ***name*** "Limes", ***threshold*** 5, ***stock*** 10, and ***price*** 3.00 to the inventory.

```
> add 101 "Limes" 5 10 3.00
```

This should print an error, as ***Key*** 101 is already taken. *<u>Sample</u> <u>Delete:</u>*

```
> delete 101
```

This should delete *Carrots* from the inventory hash table *<u>Sample</u> <u>Restock:</u>*

```
> restock 101 5
```

This should add 5 to the ***stock*** of *Carrots*.

<u>Sample Restock:</u>

```
> restock all
```

This should, for each *item* that has ***stock*** < ***threshold***, add to ***stock*** until ***stock*** = ***threshold***

Check-Out System

Description

A grocery item (*gItem*) is:

- ***Key***
  - *int* - the key that correctly corresponds to the item's ***key*** in the *inventory*.
- ***Amount***
  - *int* - the number of that item being purchased

Customers in the check-out queue will have the following:

- ***Name***
  - *string* - The customer's name (these do not have to be unique).
- ***Cash***
  - *float* - Amount of money they have on them.
- ***Grocery List***
  - A List of *gItems* the customer is wanting to buy.

A queue of *customers*, each with groceries to check out at the register (there is only one register), will be provided from a text file. The *customers* are to be enqueued, then one by one checked out at the register.

At the register each *customer* will be checked out, one *gItem* at a time. The price for the *gItem* should be added to the running total, the *inventory* system should have the corresponding *item* subtract the appropriate number from the *stock*, and if necessary flag the system to print a message to restock that *item*. If a *customer* does not have enough money for the entire purchase, they are rejected and all items are returned to the *inventory.* You do not have to worry about partial purchases.

Each *customer* will have a log associated with their transaction, this log will be written to a text file once the entire queue has been processed. This log will contain the *gItems* purchased (along with the number and price), and how much they paid in ***cash***. This log will be written to a file, if the file containing the customers is `"file1.txt"`, then the log will be written to a file `"file1_receipt.txt"`. Examples of this log will be in the next section.

Once the entire queue has been checked out and the receipt written, any message for restocking should be printed to the console to notify the user of which *items* need restocking. All *items* needing restock should be listed in the same message.

The system should be able to check out multiple queues of *customers* (given by text files), the syntax for getting to this functionality will be described in the *menu section*.

You must use a **struct** as defined in the book/slides for the *customers* in the queue. You must also use a **queue** to have the *customers* line up for the register. You do not have to make a **struct** for *gItems*, but you may want to for convenience.

Syntax and Examples

The check-out system should allow the following:

1. Take a text file containing customers, including their grocery items, which will line up in a queue to be checked out.

The syntax for the operations listed above are as follows:

1. `checkout` *filename*
    a. string - *filename* is the name of the file (same directory) containing the customers wanting to be checked out.

<u>This will be a (test) file “test.txt” provided where you got this pdf.</u>
I have added spaces here for readability, make sure you look at the actual syntax of the provided file!

```
{"Karen", 8.00, [{102, 3}, {216, 1}]}
{"Karen", 20.00, [{006, 3}, {039, 1}]}
{"Bob", 16.00, [{039, 1}]}
{"Elmo", 30.00, [{039, 2}, {102, 1}]}
{"Human McPerson", 150.00, {[039, 10]}
```

<u>This would indicate:</u>
Customer with ***name*** “Karen”, has $8.00 in ***cash***, their ***grocery list*** is:

3x *item* 102 (Apples at $0.99), and 1x *item* 216 (Swiss Cheese at $2.49) Customer with ***name*** “Karen”, has $20.00 in ***cash***, their ***grocery list*** is:

3x *item* 006 (Coffee at $5.00), and 1x *item* 039 (Wine at $12.99).

Customer with ***name*** “Bob”, has $15.00 in ***cash***, their ***grocery list*** is: 1x *item* 039 (Wine at $12.99).

Customer with ***name*** “Elmo”, has $30.00 in ***cash***, their ***grocery list*** is:

2x *item* 039 (Wine at $12.99), and 1x *item* 102 (Apples at $0.99).

Customer with ***name*** “Human McPerson”, has $150.00 in ***cash***, their ***grocery list*** is: 10x *item* 039 (Wine at $12.99).

Karen (first one) has a *total* of $6.48, they have enough ***cash*** and will be checked out. The *inventory* system should deduct 3x 102 (Apples), and 1x 216 (Swiss Cheese) from the *inventory*
.

At this point, item 102 (Apples) has fallen below the restock threshold and the system will print a message to the console after the receipt is printed.

Karen (second one) has a *total* of $27.99, Karen (second one) does not have enough money and will be rejected and no items will be deducted from the *inventory*.

Bob has a *total* of $12.99, they have enough ***cash*** and will be checked out. The *inventory* system should deduct 1x 039 (Wine) from the *inventory*.

Elmo has a *total* of $26.97, they have enough ***cash*** and will be checked out. The *inventory system* should deduct 2x 039 (Wine), and 1x 102 (Apples) from the *inventory*.

Human McPerson has a *total* of $129.90, they have enough ***cash*** and will be checked out. The *inventory system* should deduct 10x 039 (Wine) from the *inventory*.

At this point, item 039 (Wine) has fallen below the restock threshold and the system will print a message to the console after the receipt is printed.

Text_Receipt.txt:

Make the output log output log should have this format for each transaction:

```
Customer - Karen



Apples x3 @ $0.99
Swiss Cheese x1 @ $2.49

Total: $6.48 Thank you, come
back soon!
-------------------------------------------------------

Customer - Karen



Coffee x3 @ $5.00
Wine x1 @ $12.99


Total: $27.99
Customer did not have enough money and was REJECTED Thank you,
come back soon!
-------------------------------------------------------
```

Inventory Message:

If, after a queue of customers, item(s) need to be restocked due to going below their threshold then the message should look like this:

```
Warning! The following Item(s) may need to be restocked:
102 (Apples): 8 remain in stock, replenishment threshold is 10
039 (Wine):  9 remain in stock, replenishment threshold is 15
```

Menu System

Description

When the program starts up, the inventory should be populated as described in its section using `“inventory.txt”`.

After creating the hash table for the inventory, the program should display a menu allowing the following *unitil the program closes* :

1. Check-out Queue of Customers
2. Manage Inventory
3. Close Program

The system should allow the user to manage any number of 1-2 selections (or their submenus) until deciding to close the program.

The check-out option should take the name of a file containing customers with grocery lists that are ready to be checked out. The system will check them out, write a receipt, and print any restock messages that may be necessary.

The Inventory option should open the Inventory submenu, which will allow the user to:

1. Add items
2. Delete items
3. Restock an item
4. Restock all items under threshold

The syntax and examples for this submenu are located in the *Inventory* section.

The Close Program option should write the current inventory to the files from which it was written. This means that you should be able to print to the same file from which you read and then be able to read it again.

Examples and Syntax

Note: the syntax for these commands must be exactly the same (or at least support the exact syntax listed here in addition to whatever else you want to add)

Anything in `courier` *font is exact text, anything italicized is a variable*

<u>Running checkout on a file containing customers:</u>

`> checkout` *test.txt*

<u>Display Inventory submenu</u>

```
> inventory
```

<u>Actual Inventory submenu</u>

`>add item (syntax: add` *key name threshold stock price* `)`

`>delete item (syntax: delete` key `)`

`>restock item (syntax: restock` key amount `)`

```
>restock all (syntax: restock all)
>return to main menu (syntax: return)
```

<u>Close program</u>

```
> quit
```

Examples for the Inventory submenu can be found in the Inventory section.