

Winter of 2017.

# Grocery Store Inventory System

Final Project  
Milestone 5 (Optional)

In a grocery store, in order to be able to always have the proper number of items available on shelves, an inventory system is needed to keep track of items available in the inventory and make sure the quantity of the items does not fall below a specific count.

Your job for this project is to prepare an application that manages the inventory of items needed for a grocery store. The application should be able to keep track of the following information about an item:

- 1- The SKU number
- 2- The name (maximum of 20 chars)
- 3- Quantity (On hand quantity currently available in the inventory)
- 4- Minimum Quantity (if the quantity of the item falls less than or equal to this value, a warning should be generated)
- 5- Price of the item
- 6- Is the item Taxed

This application must be able to do the following tasks:

- 1- Print a detailed list of all the items in the inventory
- 2- Search and display an item by its SKU number
- 3- Checkout an item to be delivered to the shelf for sale
- 4- Add to stock items that are recently purchased for inventory (add to their quantity)
- 5- Add a new item to the inventory or update an already existing item
- 6- Delete an item from the inventory (optional for extra marks)
- 7- Search for an item by its name (optional for extra marks)
- 8- Sort Items by Name (optional for extra marks)

## PROJECT DEVELOPMENT PROCESS

To make the development of this application fun and easy, the tasks are broken down into several functions that are given to you from very easy ones to more complicated one by the end of the project

Since you act like a programmer in this project, you do not need to know the big picture. The professor is your system analyst and designs the system and all its functions to work together in harmony. Each milestone is divided into a few functions. For each function, firstly, understand the goal of the function. Secondly, write the code for it and test it with the tester. Once your

code for the function passes the test, set it aside and pick up the next function. Continue until the milestone is complete.

The Development process of the project is divided into five milestones and therefore five deliverables. The first four deliverables are mandatory and accounts as full submission of the project. The fifth milestone is optional; for those who want to do some extra work for the challenge and the bonus marks. For each deliverable, a tester program (also called a unit test) will be provided to you to test your functions. If the tester works the way it is supposed to, you can submit that milestone and continue to the next one. The approximate schedule for deliverables is as follows

- The UI Tools and app interface      10 days, Due Sat Mar 18<sup>th</sup>
- The Item IO      6 Days, Due Fri Mar 24<sup>th</sup>
- Item Storage and Retrieval in Array      9 Days, Due Sun Apr 2<sup>nd</sup>
- File IO and final assembly      5 Days, Due Fri Apr 9<sup>th</sup>
- Item Name Search, Delete and Sort      4 Days, Due Thu Apr 13<sup>th</sup> (optional)

## FILE STRUCTURE OF THE PROJECT

For each milestone, two source files are provided under the name `144_msX_tester.c` and `144_msX.c`.

`144_msX_tester.c` includes the `main()` tester program provided by your professor to test your implementation of the functions of the project. (Replace X with the milestone number from 1 to 5) This main program acts like a tester (a unit test) that simply makes different calls to the functions you have written to make sure they work properly.

Code your functions in `144_msX.c` test them one by one using the main function provided. You can comment out the parts of the main program for which functions are not developed yet. You are not allowed to change the code in tester. Make sure you do not make any modifications in the tester since at the time of submission the original copy of the tester will be used for compilation automatically by the submit command.

## MARKING:

Please follow this link for marking details:

[https://scs.senecac.on.ca/~ipc144/dynamic/assignments/Marking\\_Rubric.pdf](https://scs.senecac.on.ca/~ipc144/dynamic/assignments/Marking_Rubric.pdf)

## MILESTONE 1: THE USER INTERFACE TOOLS AND APP INTERFACE

Download or Clone milestone 1 from <https://github.com/Seneca-144100/IPC-Milestone1>

In `144_msX.c` code the following functions:

## USER INTERFACE TOOLS

```
void welcome(void);
```

Prints the following line and goes to newline

```
>----== Grocery Inventory System ==---<
```

```
void printTitle(void);
```

Prints the following two lines and goes to newline

```
>Row |SKU| Name | Price |Taxed| Qty | Min | Total |Atn<
>---+---+-----+---+---+---+---+-----+---<
```

```
void printFooter(double grandTotal);
```

Prints the following line and goes to newline

$$> \text{-----} + \text{-----} <$$

Then if grandTotal is greater than zero it will print this line: (assuming grandTotal is 1234.57) and go to new line.

```
> Grand Total: | 1234.57<
```

Use this format specifier for printing grandTotal : **%12.21f**

```
void flushKeyboard(void);
```

“flush Keyboard” Makes sure the keyboard is clear by reading from keyboard character by character until it reads a new line character.

*Hint: In a loop, keep reading single characters from keyboard until newline character is read ('**\n**'). Then, exit the loop.*

```
void pause(void);
```

Pauses the execution of the application by printing a message and waiting for user to hit <ENTER>.

Print the following line and DO NOT go to newline:

```
>Press <ENTER> to continue...<
```

Then, call `flushKeyboard` function.

Here the flushKeyboard function is used for a fool-proof <ENTER> key entry.

```
int getInt(void);
```

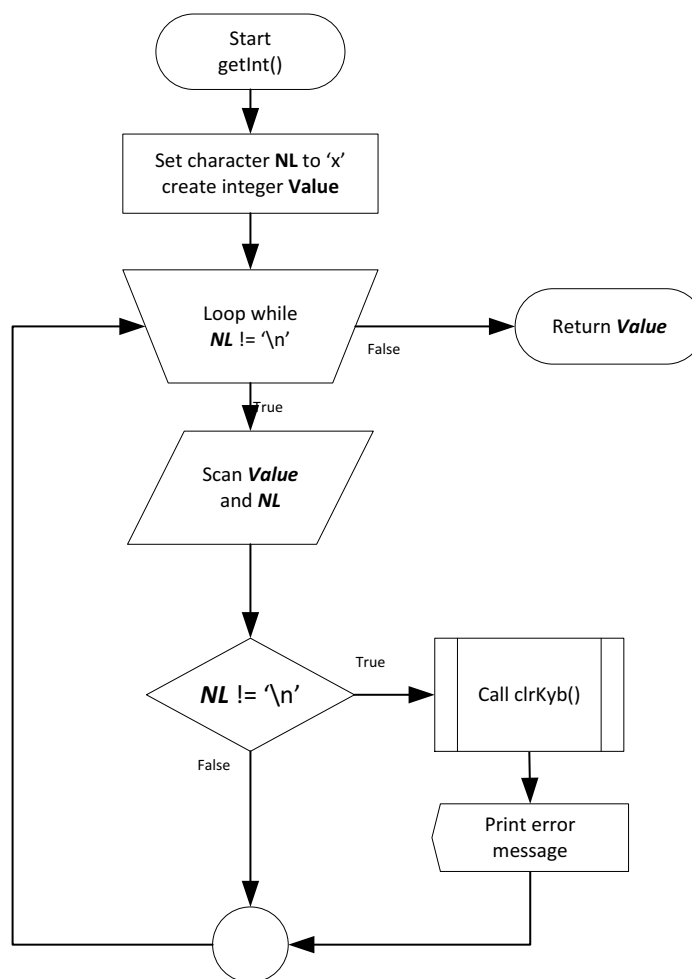
Gets a valid integer from the keyboard and returns it. If the integer is not valid it will print:

"Invalid integer, please try again: "

and try again.

This function must be fool-proof; it should not let the user pass, unless a valid integer is entered.

Hint: to do this, you can have two variables read back to back by scanf; an integer and then a character ("%d%c") and make sure the second (the character) is a new line. If the second character is a new line, then this guaranties that first integer is successfully read and also after the integer <ENTER> is hit. If the character is anything but new line, then either the user did not enter an integer properly, or has some additional characters after the integer, which is not good. In this case clear the keyboard, print an error message and scan the integer again. See the flowchart below.



```
int getIntLimited(int lowerLimit, int upperLimit);
```

This function uses getInt() to receive a valid integer and returns it. This function makes sure the integer entered is within the limits required (between *lowerLimit* and *upperLimit* inclusive). If the integer is not within the limits, it will print:

> "Invalid value, *TheLowerLimmit* < value < *TheUpperLimit*: " <  
and try again. (Change the lower and upper limit with their values.)

This function is fool-proof too; the function will not let the user pass until a valid integer is received based on the lower and upper limit values.

```
double getDouble(void);
```

Works exactly like *getInt()* but scans a double instead of an integer with the following error message:

"Invalid number, please try again: "

```
double getDoubleLimited(double lowerLimit, double upperLimit);
```

Works exactly like *getIntLimited()* but scans a double instead of an integer.

## OUTPUT SAMPLE:

(UNDERLINED, *ITALIC* **BOLD** RED VALUES ARE USER ENTRIES)

```
----- Grocery Inventory System -----
```

listing header and footer with grand total:

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
							Grand Total:	1234.57

listing header and footer without grand total:

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn

Press <ENTER> to continue... <ENTER>

Enter an integer: abc

Invalid integer, please try again: 10abc

Invalid integer, please try again: 10

You entered: 10

Enter an integer between 10 and 20: 9

Invalid value, 10 < value < 20: 21

Invalid value, 10 < value < 20: 15

Your entered 15

Enter a floating point number: abc

Invalid number, please try again: 2.3abc

Invalid number, please try again: 2.3

You entered: 2.30

Enter a floating point number between 10.00 and 20.00: 9.99

Invalid value, 10.000000 < value < 20.000000: 20.1

Invalid value, 10.000000 < value < 20.000000: 15.05

You entered: 15.05

End of tester program for IO tools!

## THE APPLICATION USER INTERFACE SKELETON

Now that the user interface tools are created and tested, we are going to build the main skeleton of our application. This application will be a menu driven program and will work as follows:

- 1- When the program starts the title of the application is displayed.
- 2- Then a menu is displayed.
- 3- The user selects one of the options on the Menu.
- 4- Depending on the selection, the corresponding action will take place.
- 5- The Application will pause to attract the user's attention
- 6- If the option selected is not Exit program, then the program will go back to option 2
- 7- If the option selected is Exit program, the program ends.

The above is essentially the pseudo code for any program that uses a menu driven user interface.

To accomplish the above create the following three functions:

**int yes(void)**

Receives a single character from the user and then clears the keyboard (`flushKeyboard()`). If the character read is anything other than "Y", "y", "N" or "n", it will print an error message as follows:

>Only (Y)es or (N)o are acceptable: <

and goes back to read a character until one of the above four characters is received.

Then, it will return 1 if the entered character is either "y" or "Y", otherwise it will return 0.

**int menu(void)**

Menu prints the following options:><

>1- List all items<

>2- Search by SKU<

>3- Checkout an item<

>4- Stock an item<

>5- Add new item or update item<

>6- delete item<

>7- Search by name<

>0- Exit program<

>> <

Then, it receives an integer between 0 and 7 inclusive and returns it. Menu will not accept any number less than 0 or greater than 7 (Use the proper UI function written in the UI tools).

### **void GroceryInventorySystem(void)**

This function is the heart of your application and will run the whole program.

GroceryInventorySystem, first, displays the welcome message and skips a line and then displays the menu and receives the user's selection.

If user selects 1, it displays:

>List Items under construction!< and goes to newline

If user selects 2, it displays:

>Search Items under construction!< and goes to newline

If user selects 3, it displays:

>Checkout Item under construction!< and goes to newline

If user selects 4, it displays:

>Stock Item under construction!< and goes to newline

If user selects 5, it displays:

>Add/Update Item under construction!< and goes to newline

If user selects 6, it displays:

>Delete Item under construction!< and goes to newline

If user selects 7, it displays:

>Search by name under construction!< and goes to newline

After receiving a number between 1 and 7, it will pause the application and goes back to display the menu.

If user selects 0, it displays:

>Exit the program? (Y)es/(N)o): <

and waits for the user to enter "Y", "y", "N" or "n" for Yes or No.

If the user replies Yes, it will end the program, otherwise it goes back to display the menu.

The following is a general pseudo code for a menu driven user interface. Using this pseudo code is optional. You can use any other logic if you like.

Application user interface pseudo code:

```
while it is not done
  display menu
  get selected option from user
  check selection:
    option one selected
      act accordingly
    end option one
    option two selected
      act accordingly
    end option two
    .
    .
    .
    Exit is selected
      program is done
    end exit
  end check
end while
```



## OUTPUT SAMPLE:

(UNDERLINED, *ITALIC* **BOLD** RED VALUES ARE USER ENTRIES)

----- Grocery Inventory System -----

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 8

Invalid value,  $0 < \text{value} < 7$ : 1

List Items under construction!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 2

Search Items under construction!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 3

Checkout Item under construction!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 4

Stock Item under construction!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 5

Add/Update Item under construction!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 6

Delete Item under construction!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 7

Search by name under construction!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 0

Exit the program? (Y)es/(N)o : x

Only (Y)es or (N)o are acceptable: n

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item

```

6- delete item
7- Search by name
0- Exit program
> 0
Exit the program? (Y)es/(N)o: y

```

## MILESTONE 1 SUBMISSION

If not on matrix already, upload your `144_ms1.c` and professor's `144_ms1_tester.c` to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms1 144_ms1.c 144_ms1_tester.c <ENTER>
```

This command will compile your code and name your executable "`ms1`"

Execute `ms1` and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_ms1 <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

## MILESTONE 2: THE ITEM INPUT/OUTPUT

Copy all the functions implemented in milestone 1 into `144_ms2.c` and add the following:

Define the following values (using `#define`)

```

LINEAR to be 1
FORM to be 0

```

Also create a global constant double variable called `TAX` that is initialized to 0.13.

Continue the development of your project by implementing the following Item related functions:

Item related information is kept in the following structure (do not modify this):

structure:

```
struct Item {
    double price;
    int sku;
    int isTaxed;
    int quantity;
    int minQuantity;
    char name[21];
};
```

price: price of a unit of the item

sku: Stock Keeping Unit, a 3 digit integer

isTaxed: an integer Flag, if true (non-zero), the tax is applied in price calculations. The value of Tax is kept in the global constant double TAX variable.

quantity: the quantity of the time in the inventory.

minQuantity: the minimum quantity number in inventory; any inventory quantity value less than this will cause a warning to order more of this item later in development.

name: a 20 character, C string (i.e a 21 character NULL terminated array of characters) to keep the name of the item.

Implement the following Item related functions:

```
double totalAfterTax(struct Item item);
```

This function receives an **Item** and calculates and returns the total inventory price of the item by multiplying the **price** by the **quantity** of the item and adding TAX if applicable (if **isTaxed** is true).

```
int isLowQuantity(struct Item item);
```

This function receives an **Item** and returns true (1) if the **Item quantity** is less than **Item minimum quantity** and false (0) otherwise.

```
struct Item itemEntry(int sku);
```

This function receives an integer argument for **sku** and creates an Item and sets its **sku** to the **sku** argument value.

Then it will prompt the user for all the values of the Item (except the **sku** that is already set) in the following order:

Name, Price, Quantity, Minimum Quantity and Is Taxed.

To get the **name** from the user, use the this format specifier in scanf: "%20[^\n]" and then clear the keyboard using **flushKeyboard()** function.

*This format specifier tells to scanf to read up to 20 characters from the keyboard and stop if "\n" (ENTER KEY) is entered. After this flushKeyboard() gets rid of the "\n" left in the keyboard.*

Use the data entry functions you created in milestone 1 to get the rest of the values. (for **isTaxed**, use the **yes()** function in milestone 1).

Here is the format of the data Entry: (Underlined *Italic* **Bold** **Red** values are user entries)

```
>      SKU: 999<
>      Name: Red Apples<
>      Price: 4.54<
>      Quantity: 50<
>Minimum Qty: 5<
>      Is Taxed: n<
```

```
void displayItem(struct Item item,int linear);
```

This function receives two arguments: an **Item** and an integer flag called **linear**.

This function prints an **Item** on screen in two different formats depending on the value of “**linear**” flag being true or false.

If linear is true it will print the Item values in a line as with following format:

- 1- bar char “|”
- 2- **sku**: integer, in 3 spaces
- 3- bar char “|”
- 4- **name**: left justified string in 20 characters space
- 5- bar char “|”
- 6- **price**: double with 2 digits after the decimal point in 8 spaces
- 7- bar char and two spaces “| ”
- 8- **IsTaxed**: Yes or No in 3 spaces
- 9- bar char and one space “| ”
- 10- **quantity**: integer in 3 spaces
- 11- space, bar char and space“ | ”
- 12- **minQuantity**: integer in 3 spaces
- 13- space and bar char “ | ”
- 14- **Total price**: double with 2 digits after the decimal point in 12 spaces
- 15- bar char “|”
- 16- if the quantity is low then three asterisks (“\*\*\*”) or nothing otherwise.

Example:

```
>|999|Red Apples      |    4.54|  No| 50 |   5 |      227.00|<
```

If low value and Taxed:

```
>|999|Red Apples      |    4.54| Yes|  2 |   5 |      10.26|***<
```

If linear is false (or in FORM format) then the values are printed as follows:

```
>      SKU: 999<
>      Name: Red Apples<
>      Price: 4.54< Two digits after the decimal point
>      Quantity: 50<
>Minimum Qty: 5<
```

```

> Is Taxed: No<
If low value and Taxed:
>     SKU: 999<
>     Name: Red Apples<
>     Price: 4.54<
>     Quantity: 2<
>Minimum Qty: 5<
> Is Taxed: Yes<
>WARNING: Quantity low, please order ASAP!!!<

```

```
void listItems(const struct Item item[], int noOfItems);
```

This function receives a constant array of **Items** and their number and prints the items in list with the grand total price at the end.

Create an integer for the loop counter and a double grand total variable that is initialized to zero.

First print the Titles of the list using printTitle() function.

Then it will loop through the **items** up to noOfItems.

In each loop:

Print the row number (loop counter plus one), left justified in four spaces and then display the item in LINEAR format. Then add the total price of the current Item element in the array to the grand total value.

After loop is done print the footer by passing the grand total to it. (use printFooter() function).

```
int locateItem(const struct Item item[], int NoOfRecs, int sku, int* index);
```

This function receives a constant array of Items and their number. Also an SKU to look for in the Item array. The last argument is a pointer to an index. The target of this index pointer will be set to the index of the Item-element in which the sku is found, otherwise no action will be taken on index pointer.

If an Item with the SKU number as the sku argument is found, after setting the target of the index pointer to the index of the found item, a true value (non-zero, preferably 1) will be returned, otherwise a false value (0) will be returned.

## MILESTONE 2 SUBMISSION

If not on matrix already, upload your [144\\_ms2.c](#) and professor's [144\\_ms2\\_tester.c](#) to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms2 144_ms2.c 144_ms2_tester.c <ENTER>
```

This command will compile your code and name your executable "[ms2](#)"

Execute [ms2](#) and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_ms2 <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

### MILESTONE 3: ITEM STORAGE AND RETRIEVAL IN ARRAY

Copy [144\\_ms2.c](#) to [144\\_ms3.c](#) and add the following definitions to your program in [144\\_ms3.c](#):

```
STOCK          1
CHECKOUT       0
MAX_ITEM_NO    21
MAX_QTY        999
SKU_MAX        999
SKU_MIN        100
```

In this milestone, you are implementing 5 functions that work with an array of **Items**.

#### Coding hint:

To be able to test your program in early stages of development as you program, first create the five functions with empty bodies. For example create the search function as follows:

```
void search(const struct Item item[], int NoOfRecs) {
    // nothing in here!!!
}
```

This will let your program to compile successfully with the tester program. Obviously your functions will not do anything, but this will allow you to test your code in early stages of development and makes debugging much easier. Happy coding!

```
void search(const struct Item item[], int NoOfRecs);
```

The **search** function receives an array of items and its size and searches through the array for an **Item** with a specific **sku** that is received from the user. If found, it will display the item in **FORM** format, otherwise it will print an error message.

#### DETAILS

Prompt:

>Please enter the SKU: <

then receive an integer between **SKU\_MIN** and **SKU\_MAX**.

Call the **locateItem()** function and see if the item is found.

If found, display the item , otherwise print:

>Item not found!< and go to new line.

```
void updateItem(struct Item* itemptr);
```

updateitem, modifies the fields of an Item. The function receives the address of the Item to update (itemptr).

#### DETAILS

Create an instance of **struct Item**.

Prompt:

>Enter new data:< and go to new line.

Use the **itemEntry()** function and the **SKU** of the Item pointed by **itemptr** to receive an Item and save it in the Item instance you just created.

Then ask the user to confirm the update by printing:

>Overwrite old data? (Y)es/(N)o: <

If user responds yes, overwrite the target of **itemptr** by the Item instance and print:

>---- Updated! ----< and go to new line

Otherwise print:

>---- Aborted! ----< and go to new line

```
void addItem(struct Item item[], int *NoOfRecs, int sku);
```

If the item array is not full, this function will ask the user to enter the data for an Item (with the SKU received through the argument list) and if the user confirms, it will add it to the array and add one to the target of **NoOfRecs** pointer.

#### DETAILS

Create an Item.

Check:

If the target of NoOfRecs is equal to **MAX\_ITEM\_NO**, print:

>Can not add new item; Storage Full!<

and exit the function.



Otherwise, using the **itemEntry()** function get the details of the new Item with the SKU from the argument list and Prompt:

>Add Item? (Y)es/(N)o: <

If the user replies yes, set the Item after the last one in the item array to the one you just got from the user and add one to the target of **NoOfRecs** pointer and print:

>---= Added! ===-< and go to new line and exit the function.

If the user replies no, print:

>---= Aborted! ===-< and go to new line and exit the function.

```
void addOrUpdateItem(struct Item item[], int* NoOfRecs);
```

addOrUpdateItem function, receives an SKU from the user and updates or adds an Item in an array of Items depending on the SKU being present in an item in the array or not.

#### DETAILS

Prompt:

>Please enter the SKU: <

Receive an integer within the limits of a valid SKU number; between **SKU\_MIN** and **SKU\_MAX**.

Try locating the item in the item array.

If found:

Display the item in **FORM** format and confirm that the user wants to update it by prompting:

>Item already exists, Update? (Y)es/(N)o: <

If the user replies yes, call the **updateItem()** function with the found Item in the array, otherwise print:

>---= Aborted! ===-< and go to new line and exit the function.

Otherwise (not found):

Call the **addItem()** function to add the item with the entered SKU at the end of the data in the array. Then exit the function.

```
void adjustQuantity(struct Item item[], int NoOfRecs, int stock);
```

Depending on the value of the stock argument being **STOCK** or **CHECKOUT**, this function will increase or reduce the quantity of the selected Item in the array by the value received from the user.

If stocking, (adding to storage) this value can vary between 0 to (MAX\_QTY – item\_quantity) and if checking out (removing from storage) this value can vary between 0 to item\_quantity.

#### DETAILS

Prompt:

>Please enter the SKU: <

Get a valid SKU value from the user and try locating the item with the same SKU in the item array.

If not found, print:

>SKU not found in storage!< and go to new line and exit the function.

If found, display the item in **FORM** format and print this message:

>Please enter the quantity %s; Maximum of %d or 0 to abort: <

If the value of **stock** argument is **STOCK** then:

%s should be replaced by >to stock< and %d should be replaced by the maximum number of items that can be stocked without exceeding the **MAX\_QTY** value, which is the **MAX\_QTY** value minus the quantity of the item in stock.

If the value of **stock** argument is **CHECKOUT** then:

%s should be replaced by >to checkout< and %d should be replaced by the quantity of the item in stock.

Then check the value entered by the user, which must be between 0 and the quantity displayed in the preceding message.

If the number input is zero (0), then print >---- Aborted! ----<, go to a new line and exit the function.

If the number input is not zero(0):

If the value of **stock** argument is **STOCK** then:

Increase the quantity of the item in the array by the amount received from the user and print: >---- Stocked! ----< and go to new line.

If the value of **stock** argument is **CHECKOUT** then:

Reduce the quantity of the item in the array by the amount received from the user and print: >---- Checked out! ---- < and go to new line.

When finished processing, if the quantity of the item is low (less than the re-order point), print the following warning:

>Quantity is low, please reorder ASAP!!!< and go to a new line.

## TESTER OUTPUT SAMPLES:

(UNDERLINED, *ITALIC* **BOLD** **RED** VALUES ARE USER ENTRIES)

## SEARCH TEST:

=====Search Test:

Enter 731:

Please enter the SKU: **731**

SKU: 731

Name: Allen's Apple Juice  
 Price: 1.79  
 Quantity: 100  
 Minimum Qty: 10  
 Is Taxed: Yes  
 Enter 222:  
 Please enter the SKU: 222  
 Item not found!

## UPDATE TEST:

=====Update Test:  
 Enter the follwoing:  
     SKU: 111  
     Name: Grape  
     Price : 22.22  
     Quantity : 22  
 Minimum Qty : 2  
     Is Taxed : y  
 Overwrite old data? (Y)es/(N)o: n  
 Enter new data:  
     SKU: 111  
     Name: Grape  
     Price: 22.22  
     Quantity: 22  
 Minimum Qty: 2  
     Is Taxed: y  
 Overwrite old data? (Y)es/(N)o: n  
 ---= Aborted! ===  
 Unchanged Item Data:  
     SKU: 111  
     Name: Ones!  
     Price: 11.11  
     Quantity: 11  
 Minimum Qty: 1  
     Is Taxed: Yes  
 Enter the follwoing:  
     SKU: 111  
     Name: Grape  
     Price : 22.22  
     Quantity : 22  
 Minimum Qty : 2  
     Is Taxed : y

Overwrite old data? (Y)es/(N)o: y

Enter new data:

SKU: 111

Name: Grape

Price: 22.22

Quantity: 22

Minimum Qty: 2

Is Taxed: y

Overwrite old data? (Y)es/(N)o: y

---== Updated! ==--

Updated Item:

SKU: 111

Name: Grape

Price: 22.22

Quantity: 22

Minimum Qty: 2

Is Taxed: Yes

## ADD TEST:

=====Add Test:

Total items in Storage: 20, Max no: 21

Enter the follwoing:

SKU: 222

Name: Grape

Price : 22.22

Quantity : 22

Minimum Qty : 2

Is Taxed : y

Add Item? (Y)es/(N)o: n

SKU: 222

Name: Grape

Price: 22.22

Quantity: 22

Minimum Qty: 2

Is Taxed: y

Add Item? (Y)es/(N)o: n

---== Aborted! ==--

Garbage here! Nothing is added, No of items in storage: 20

SKU: 0

Name:

Price: 0.00

Quantity: 0

Minimum Qty: 0

```

    Is Taxed: No
WARNING: Quantity low, please order ASAP!!!
Enter the follwoing:
    SKU: 222
    Name: Grape
    Price : 22.22
    Quantity : 22
Minimum Qty : 2
    Is Taxed : y
Add Item? (Y)es/(N)o: y
    SKU: 222
    Name: Grape
    Price: 22.22
    Quantity: 22
Minimum Qty: 2
    Is Taxed: y
Add Item? (Y)es/(N)o: y
---- Added! ----
New item is added, No of itemsinstorage: 21
    SKU: 222
    Name: Grape
    Price: 22.22
    Quantity: 22
Minimum Qty: 2
    Is Taxed: Yes
Adding 333:
Can not add new item; Storage Full!

```

## ADD OR UPDATE TEST:

```

=====AddOrUpdate Test:
Enter 731 and then 'n':
Please enter the SKU: 731
    SKU: 731
    Name: Allen's Apple Juice
    Price: 1.79
    Quantity: 100
Minimum Qty: 10
    Is Taxed: Yes
Item already exists, Update? (Y)es/(N)o: n
---- Aborted! ----
Enter 731, 'y' and then:
    Name: Apple
    Price: 1.80

```

Quantity: 101  
 Minimum Qty: 11  
 Is Taxed: n  
 Overwrite old data? (Y)es/(N)o: y  
 Please enter the SKU: 731  
     SKU: 731  
     Name: Allen's Apple Juice  
     Price: 1.79  
     Quantity: 100  
 Minimum Qty: 10  
 Is Taxed: Yes  
 Item already exists, Update? (Y)es/(N)o: y  
 Enter new data:  
     SKU: 731  
     Name: Apple  
     Price: 1.80  
     Quantity: 101  
 Minimum Qty: 11  
 Is Taxed: n  
 Overwrite old data? (Y)es/(N)o: y  
 ---- Updated! ----  
 Updated Item:  
     SKU: 731  
     Name: Apple  
     Price: 1.80  
     Quantity: 101  
 Minimum Qty: 11  
 Is Taxed: No  
 Enter 444:  
 Please enter the SKU: 444  
 Can not add new item; Storage Full!

## ADJUST QUANTITY TEST:

=====AdjustQuantity Test:  
 Invalid SKU Test; Enter 444:  
 Please enter the SKU: 444  
 SKU not found in storage!  
 Aborting Entry test; Enter 731 and then 0:  
 Please enter the SKU: 731  
     SKU: 731  
     Name: Allen's Apple Juice  
     Price: 1.79  
     Quantity: 100  
 Minimum Qty: 10

```

    Is Taxed: Yes
Please enter the quantity to checkout; Maximum of 100 or 0 to abort: 0
---- Aborted! ----
Checking out with low quantity warning; Enter 731 and then 90:
Please enter the SKU: 731
    SKU: 731
    Name: Allen's Apple Juice
    Price: 1.79
    Quantity: 100
Minimum Qty: 10
    Is Taxed: Yes
Please enter the quantity to checkout; Maximum of 100 or 0 to abort:
90
---- Checked out! ----
Quantity is low, please reorder ASAP!!!
Stocking; Enter 731 and then 50:
Please enter the SKU: 731
    SKU: 731
    Name: Allen's Apple Juice
    Price: 1.79
    Quantity: 10
Minimum Qty: 10
    Is Taxed: Yes
WARNING: Quantity low, please order ASAP!!!
Please enter the quantity to stock; Maximum of 989 or 0 to abort: 50
---- Stocked! ----
    SKU: 731
    Name: Allen's Apple Juice
    Price: 1.79
    Quantity: 60
Minimum Qty: 10
    Is Taxed: Yes

```

## MILESTONE 3 SUBMISSION

If not on matrix already, upload your [144\\_ms3.c](#) and professor's [144\\_ms3\\_tester.c](#) to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms3 144_ms3.c 144_ms3_tester.c <ENTER>
```

This command will compile your code and name your executable "[ms3](#)"

Execute [ms3](#) and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_ms3 <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

## MILESTONE 4.1: FILE IO

Copy `144_ms3.c` to `144_ms4.c` and add the following definitions to your program in `144_ms4.c`:

```
MAX_ITEM_NO    500

DATAFILE       "items.txt"
```

## ADJUSTMENTS TO ITEMENTRY FUNCTION IN MILESTONE 2:

If not done already, modify your `struct Item` `itemEntry(int sku)` and use `getIntLimitet()` and `getDoubleLimited()` to limit the following entries:

```
>      SKU: 999<
>      Name: Red Apples<
>      Price: 4.54< Limited between 0.01 and 1000.00 inclusive
>      Quantity: 50< Limited between 1 and MAX_QTY inclusive
>Minimum Qty: 5< Limited between 0 and 100 inclusive
>      Is Taxed: n<
```

Implement the following four functions:

```
void saveItem(struct Item item, FILE* dataFile);
```

This function writes the content of an `Item`, comma separated, in one line of a text file pointed by “`datafile`” argument in the following format:

```
sku,quantity,minQuantity,price,isTaxed,name<NEWLINE>
```

All the above variables are written with no special formatting except for the price that is written with 2 digits after the decimal point.

Assume that the `dataFile` pointer is already open and ready to be written into.



```
int loadItem(struct Item* item, FILE* dataFile);
```

This function reads all the fields of an **Item** from one line of a comma separated text file using **fscanf** and stores them in the **Item** structure that is pointed by the “**item**” pointer in the argument list. The format in which the values are read is the same as the **saveItem** function. Note that the name field may contain spaces.

Assume that the **dataFile** **FILE** pointer is already open and ready to be read from.

The function returns **true** if **fscanf** reads the six fields successfully or **false** otherwise.

```
int saveItems(const struct Item item[], char fileName[], int NoOfRecs);
```

**saveItems** uses the **saveItem** function to write an entire array of **Items** into a file.

**saveItems** receives a constant array of **Items** and the number of records in that array (**NoOfRecs**) and also the name of the file in which these items should be saved into.

**saveItems** opens a **FILE** using the **filename** received from the argument list for writing (overwrites the old file if it already exists).

If the file is not opened successfully, it ends the function and returns **zero**.

If the file is opened successfully, it goes through all the elements of the array, “**item**”, up to the “**NoOfRecs**” and saves them one by one using the **saveItem** function.

Then, it closes the **FILE** and exits the function returning **one** (**true**).

```
int loadItems(struct Item item[], char fileName[], int* NoOfRecsPtr);
```

**loadItems** uses the **loadItem** function to read all the records saved in a file into the “**item**” array and sets the target of the “**NoOfRecsPtr**” to the number of **Items** read from the file.

**loadItems** receives an array of **Items** and another pointer pointing to the number of records read from the file (**NoOfRecsPtr**) and also the name of the file in which these **items** are stored in.

**loadItems** opens a **FILE** using the **filename** received from the argument list for reading.

If the file is not opened successfully, it ends the function and returns **zero**.

If the file is opened successfully, using **loadItem** it reads the records from the file until **loadItem** fails, counting the number of **Items** read at the same time.

Then it sets the target of **NoOfRecsPtr** pointer to the number of **Items** read.

Finally, it closes the **FILE** and exits the function returning **one (true)**.

## MILESTONE 4.1, FILE IO TESTER:

To test the FILE IO functions, compile 144\_ms4.c with 144\_ms4\_tester.c and run it.

You should have the following output:

```
*****Testing saveItem:
Your saveItem saved the following in test.txt:
275,10,2,4.40,0,Royal Gala Apples
386,20,4,5.99,0,Honeydew Melon
240,30,5,3.99,0,Blueberries
*****
They have to match the following:
275,10,2,4.40,0,Royal Gala Apples
386,20,4,5.99,0,Honeydew Melon
240,30,5,3.99,0,Blueberries
*****END Testing saveItem!

Press <ENTER> to continue...
*****Testing loadItem:
Your loadItem loaded the following from test.txt:
|275|Royal Gala Apples   | 4.40| No| 10 | 2 | 44.00|
|386|Honeydew Melon     | 5.99| No| 20 | 4 | 119.80|
|240|Blueberries        | 3.99| No| 30 | 5 | 119.70|
*****
They have to match the following:
|275|Royal Gala Apples   | 4.40| No| 10 | 2 | 44.00|
|386|Honeydew Melon     | 5.99| No| 20 | 4 | 119.80|
|240|Blueberries        | 3.99| No| 30 | 5 | 119.70|
*****END Testing loadItem!

Press <ENTER> to continue...
*****Testing saveItems:
Your saveItems saved the following in test.txt:
275,10,2,4.40,0,Royal Gala Apples
386,20,4,5.99,0,Honeydew Melon
240,30,5,3.99,0,Blueberries
*****
They have to match the following:
275,10,2,4.40,0,Royal Gala Apples
386,20,4,5.99,0,Honeydew Melon
```

```
240,30,5,3.99,0,Blueberries
*****END Testing saveItems!
```

Press <ENTER> to continue...

```
*****Testing loadItems:
```

Your loadItems loaded the following from test.txt:

275	Royal Gala Apples	4.40	No	10	2	44.00
386	Honeydew Melon	5.99	No	20	4	119.80
240	Blueberries	3.99	No	30	5	119.70

```
*****
```

They have to match the following:

275	Royal Gala Apples	4.40	No	10	2	44.00
386	Honeydew Melon	5.99	No	20	4	119.80
240	Blueberries	3.99	No	30	5	119.70

```
*****END Testing loadItems!
```

Press <ENTER> to continue...

Done!

## MILESTONE 4 SUBMISSION

If not on matrix already, upload your [144\\_ms4.c](#) and professor's [144\\_ms4\\_tester.c](#) to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms4 144_ms4.c 144_ms4_tester.c <ENTER>
```

This command will compile your code and name your executable "[ms4](#)"

Execute [ms4](#) and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_ms4 <ENTER>
```

and follow the instructions.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

## FINAL ASSEMBLY:

After the implementation of the four FILE IO functions is complete, remove **144\_ms4\_tester.c** from your project and add the following main function to your **144\_ms4.c**:

```
int main(void) {
    GroceryInventorySystem();
    return 0;
}
```

Then modify **GroceryInventorySystem** function as follows:

In your **void GroceryInventorySystem (void)** function, done in Milestone 1, do the following:

- Create an array of Items. Use **MAX\_ITEM\_NO** for its size.
- Create an integer variable to Hold the number of records read.
- After the welcome() message use the loadItems() function to fill the Items array you created with the Item records kept in a the data file. The name of the data file is defined in **DATAFILE**.
- If the loadItems fails, print the following message and exit the program:  
**"Could not read from %s.\n"**, replace %s with defined value in **DATAFILE**.
- If loadItems is successful, run the menu section and action selection:
  - o If 1 is selected:  
Call the **listItems** function passing the item array and the number of records.
  - o If 2 is selected:  
Call the **search** function passing the item array and the number of records.
  - o If 3 is selected:  
Call the **adjustQuantity** function passing the item array, the number of records and **CHECKOUT**.  
Then, call the **saveItems** function passing the item array, **DATAFILE**, and the number of records to apply the changes to the **DATAFILE**.  
If **saveItems** fails, print the following message:  
**"could not update data file %s\n"** replacing %s with **DATAFILE**.
  - o If 4 is selected:  
Do exactly what you have done in option 3 but pass **STOCK** to **adjustQuantity** instead of **CHECKOUT**.
  - o If 5 is selected:  
Call the **addOrUpdateItem** function passing the item array and the address of the number of records.  
Then call the **saveItems** function passing the item array, **DATAFILE**, and the number of records to apply the changes to the **DATAFILE**.  
If **saveItems** fails, print the following message:  
**"could not update data file %s\n"** replacing %s with **DATAFILE**.

- Note: as you see the save procedures in options 3,4 and 5 are identical. You could create a function and call it to prevent redundancy.
- If 6 or 7 is selected:  
Print: **"Not implemented!\n"** and pause();
- Option 0 remains the same.

## PROJECT SUBMISSION

### comments.txt:

Create a file called comments.txt to be submitted along with your project.

Add the following information to comments.txt:

Citation:

If you have used any code or logic developed by others in your solution, you must identify the source of that code or logic in comments in your source code.

You must insert these comments at the place in your source code where the code/logic is being used.

You must also copy and list this information in comments.txt.

*Failure to identify any code or logic developed by others and included in your solution will result in an assessment of plagiarism by your professor.*

Enhancements:

If you have enhanced your solution with any features not in the specifications, you should identify them clearly through comments in your source code.

You should also copy and list this information in comments.txt.

## SUBMISSION METHODS

You have 3 choices for submitting your Final Assembly:

1- Short

You can select this one if you already successfully submitted milestones 1 to 4

2- Long

You can select this one if you have not submitted the past 4 milestones and want to only submit Final Assembly. Your program must work exactly as described in past 4 milestones.

3- Open

This submission does not test the output of your application but captures and submits it to your professor.

You can select this one if you believe your project works properly but you could not match the exact output requested in either Short or Long submissions.

YOU are responsible to do all the tests asked in either Short or Long submission. If you miss any steps of those tests your submission may be rejected and you have to re-submit your application and you will lose mark for resubmission and not being on time.

Also you must add a note in comments.txt file with explanation on why you have chosen the open submission rather than one of the other two.

## SHORT SUBMISSION:

If not on matrix already, upload your **144\_ms4.c** and **comments.txt** to your matrix account. Compile your code as follows:

```
> gcc -Wall -o fp 144_ms4.c <ENTER>
```

This command will compile your code and name your executable “**fp**”

Execute **fp** and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_fp_short <ENTER>
```

and run your application using the **RED underlined bold *italic*** values in the short output section below.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

## OUTPUT (SHORT SUBMISSION)

```
----- Grocery Inventory System -----
```

```
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 1
```

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	
4	916	Seedless Grapes	10.56	No	20	3	211.20	
5	385	Pomegranate	2.50	No	5	2	12.50	
6	495	Banana	0.44	No	100	30	44.00	
7	316	Kiwifruit	0.50	No	123	10	61.50	
8	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
9	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
10	359	Beefsteak Pie	5.29	Yes	40	5	239.11	
11	127	Curry Chicken	4.79	Yes	30	3	162.38	
12	238	Tide Detergent	16.99	Yes	10	2	191.99	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Jack's Apple Juice	1.50	Yes	80	10	135.60	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
19	222	Peaches	1.44	No	14	20	20.16	***
Grand Total:							2327.31	

Press <ENTER> to continue... <ENTER>

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 2

Please enter the SKU: 888

Item not found!

Press <ENTER> to continue... <ENTER>

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 3

Please enter the SKU: 275

SKU: 275

Name: Royal Gala Apples

Price: 4.40

Quantity: 10

Minimum Qty: 2

Is Taxed: No

Please enter the quantity to checkout; Maximum of 10 or 0 to abort: 0

=== Aborted! ===

Press <ENTER> to continue... <ENTER>

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item

5- Add new item or update item  
 6- delete item  
 7- Search by name  
 0- Exit program

> 4

Please enter the SKU: 275

SKU: 275

Name: Royal Gala Apples

Price: 4.40

Quantity: 10

Minimum Qty: 2

Is Taxed: No

Please enter the quantity to stock; Maximum of 989 or 0 to abort: 0

--== Aborted! ==--

Press <ENTER> to continue... <ENTER>

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item

7- Search by name

0- Exit program

> 5

Please enter the SKU: 240

SKU: 240

Name: Blueberries

Price: 3.99

Quantity: 30

Minimum Qty: 5

Is Taxed: No

Item already exists, Update? (Y)es/(N)o: n

--== Aborted! ==--

Press <ENTER> to continue... <ENTER>

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item

7- Search by name

0- Exit program

> 5

Please enter the SKU: 888

SKU: 888

Name: x

Price: 1

Quantity: 1

Minimum Qty: 1

Is Taxed: y

Add Item? (Y)es/(N)o: y

--== Added! ==--

Press <ENTER> to continue... <ENTER>

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item



7- Search by name

0- Exit program

> 1

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	
4	916	Seedless Grapes	10.56	No	20	3	211.20	
5	385	Pomegranate	2.50	No	5	2	12.50	
6	495	Banana	0.44	No	100	30	44.00	
7	316	Kiwifruit	0.50	No	123	10	61.50	
8	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
9	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
10	359	Beffsteak Pie	5.29	Yes	40	5	239.11	
11	127	Curry Checken	4.79	Yes	30	3	162.38	
12	238	Tide Detergent	16.99	Yes	10	2	191.99	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Jack's Apple Juice	1.50	Yes	80	10	135.60	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
19	222	Peaches	1.44	No	14	20	20.16	***
20	888	x	1.00	Yes	1	1	1.13	***
Grand Total:							2328.44	

Press <ENTER> to continue... <ENTER>

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item

7- Search by name

0- Exit program

> 0

Exit the program? (Y)es/(N)o: y

## LONG SUBMISSION:

If not on matrix already, upload your [144\\_ms4.c](#) and [comments.txt](#) to your matrix account. Compile your code as follows:

> gcc -Wall -o fp 144\_ms4.c<ENTER>

This command will compile your code and name your executable "fp"

Execute fp and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_fp_long <ENTER>
```

and run your application using the **RED** underlined **bold italic** values in the short output section below.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

```
----- Grocery Inventory System -----
```

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

```
> abc
```

```
Invalid integer, please try again: -1
```

```
Invalid value, 0 < value < 7: 8
```

```
Invalid value, 0 < value < 7: 1
```

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	
4	916	Seedless Grapes	10.56	No	20	3	211.20	
5	385	Pomegranate	2.50	No	5	2	12.50	
6	495	Banana	0.44	No	100	30	44.00	
7	316	Kiwifruit	0.50	No	123	10	61.50	
8	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
9	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
10	359	Beefsteak Pie	5.29	Yes	40	5	239.11	
11	127	Curry Chicken	4.79	Yes	30	3	162.38	
12	238	Tide Detergent	16.99	Yes	10	2	191.99	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Jack's Apple Juice	1.50	Yes	80	10	135.60	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
19	222	Peaches	1.44	No	14	20	20.16	***
Grand Total:							2327.31	

```
Press <ENTER> to continue... <ENTER>
```

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item

```
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 2
Please enter the SKU: 888
Item not found!
Press <ENTER> to continue... <ENTER>
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 2
Please enter the SKU: 222
    SKU: 222
    Name: Peaches
    Price: 1.44
    Quantity: 14
    Minimum Qty: 20
    Is Taxed: No
WARNING: Quantity low, please order ASAP!!!
Press <ENTER> to continue... <ENTER>
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 2
Please enter the SKU: 835
    SKU: 835
    Name: 7up 12 pack
    Price: 6.49
    Quantity: 20
    Minimum Qty: 2
    Is Taxed: Yes
Press <ENTER> to continue... <ENTER>
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 3
Please enter the SKU: 385
    SKU: 385
    Name: Pomegranate
    Price: 2.50
    Quantity: 5
    Minimum Qty: 2
```

Is Taxed: No  
Please enter the quantity to checkout; Maximum of 5 or 0 to abort: 10  
Invalid value, 0 < value < 5: abc  
Invalid integer, please try again: -1  
Invalid value, 0 < value < 5: 0  
==== Aborted! ====  
Press <ENTER> to continue... <ENTER>  
1- List all items  
2- Search by SKU  
3- Checkout an item  
4- Stock an item  
5- Add new item or update item  
6- delete item  
7- Search by name  
0- Exit program  
> 3  
Please enter the SKU: 385  
SKU: 385  
Name: Pomegranate  
Price: 2.50  
Quantity: 5  
Minimum Qty: 2  
Is Taxed: No  
Please enter the quantity to checkout; Maximum of 5 or 0 to abort: 2  
==== Checked out! ====  
Press <ENTER> to continue... <ENTER>  
1- List all items  
2- Search by SKU  
3- Checkout an item  
4- Stock an item  
5- Add new item or update item  
6- delete item  
7- Search by name  
0- Exit program  
> 1

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	
4	916	Seedless Grapes	10.56	No	20	3	211.20	
5	385	Pomegranate	2.50	No	3	2	7.50	
6	495	Banana	0.44	No	100	30	44.00	
7	316	Kiwifruit	0.50	No	123	10	61.50	
8	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
9	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
10	359	Beefsteak Pie	5.29	Yes	40	5	239.11	
11	127	Curry Chicken	4.79	Yes	30	3	162.38	
12	238	Tide Detergent	16.99	Yes	10	2	191.99	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Jack's Apple Juice	1.50	Yes	80	10	135.60	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
19	222	Peaches	1.44	No	14	20	20.16	***
Grand Total:							2322.31	

Press <ENTER> to continue... <ENTER>

```

1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 3
Please enter the SKU: 385
    SKU: 385
    Name: Pomegranate
    Price: 2.50
    Quantity: 3
Minimum Qty: 2
    Is Taxed: No
Please enter the quantity to checkout; Maximum of 3 or 0 to abort: 1
--== Checked out! ==--
Quantity is low, please reorder ASAP!!!
Press <ENTER> to continue... <ENTER>
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 4
Please enter the SKU: 888
SKU not found in storage!
Press <ENTER> to continue... <ENTER>
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 4
Please enter the SKU: 385
    SKU: 385
    Name: Pomegranate
    Price: 2.50
    Quantity: 2
Minimum Qty: 2
    Is Taxed: No
WARNING: Quantity low, please order ASAP!!!
Please enter the quantity to stock; Maximum of 997 or 0 to abort: 997
--== Stocked! ==--
Press <ENTER> to continue... <ENTER>
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name

```

0- Exit program

> 1

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	
4	916	Seedless Grapes	10.56	No	20	3	211.20	
5	385	Pomegranate	2.50	No	999	2	2497.50	
6	495	Banana	0.44	No	100	30	44.00	
7	316	Kiwifruit	0.50	No	123	10	61.50	
8	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
9	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
10	359	Beefsteak Pie	5.29	Yes	40	5	239.11	
11	127	Curry Chicken	4.79	Yes	30	3	162.38	
12	238	Tide Detergent	16.99	Yes	10	2	191.99	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Jack's Apple Juice	1.50	Yes	80	10	135.60	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
19	222	Peaches	1.44	No	14	20	20.16	***
Grand Total:							4812.31	

Press <ENTER> to continue... <ENTER>

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 5

Please enter the SKU: 888

SKU: 888

Name: Apples

Price: 3.99

Quantity: 10

Minimum Qty: 2

Is Taxed: n

Add Item? (Y)es/(N)o: y

--== Added! ==--

Press <ENTER> to continue... <ENTER>

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 1

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	

4	916	Seedless Grapes		10.56	No	20	3	211.20
5	385	Pomegranate		2.50	No	999	2	2497.50
6	495	Banana		0.44	No	100	30	44.00
7	316	Kiwifruit		0.50	No	123	10	61.50
8	355	Chicken Alfredo		4.49	Yes	20	5	101.47
9	846	Veal Parmigiana		5.49	Yes	3	5	18.61 ***
10	359	Beefsteak Pie		5.29	Yes	40	5	239.11
11	127	Curry Chicken		4.79	Yes	30	3	162.38
12	238	Tide Detergent		16.99	Yes	10	2	191.99
13	538	Lays Chips S&V		3.69	Yes	1	5	4.17 ***
14	649	Joe Org Chips		3.29	Yes	15	5	55.77
15	731	Jack's Apple Juice		1.50	Yes	80	10	135.60
16	984	Coke 12 Pack		6.69	Yes	30	3	226.79
17	350	Nestea 12 Pack		7.29	Yes	50	5	411.88
18	835	7up 12 pack		6.49	Yes	20	2	146.67
19	222	Peaches		1.44	No	14	20	20.16 ***
20	888	Apples		3.99	No	10	2	39.90
-----								
Grand Total:								4852.21

Press <ENTER> to continue... <ENTER>

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item

7- Search by name

0- Exit program

> 5

Please enter the SKU: 888

SKU: 888

Name: Apples

Price: 3.99

Quantity: 10

Minimum Qty: 2

Is Taxed: No

Item already exists, Update? (Y)es/(N)o: Y

Enter new data:

SKU: 888

Name: Red Delicious Apples

Price: 3.50\$

Invalid number, please try again: 3.50

Quantity: 10 boxes

Invalid integer, please try again: 10

Minimum Qty: only 2

Invalid integer, please try again: 2

Is Taxed: Y <- Capital Y!

Overwrite old data? (Y)es/(N)o: n

==== Aborted! ====

Press <ENTER> to continue... <ENTER>

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item

7- Search by name

0- Exit program

> 1

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	
4	916	Seedless Grapes	10.56	No	20	3	211.20	
5	385	Pomegranate	2.50	No	999	2	2497.50	
6	495	Banana	0.44	No	100	30	44.00	
7	316	Kiwifruit	0.50	No	123	10	61.50	
8	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
9	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
10	359	Beefsteak Pie	5.29	Yes	40	5	239.11	
11	127	Curry Chicken	4.79	Yes	30	3	162.38	
12	238	Tide Detergent	16.99	Yes	10	2	191.99	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Jack's Apple Juice	1.50	Yes	80	10	135.60	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
19	222	Peaches	1.44	No	14	20	20.16	***
20	888	Apples	3.99	No	10	2	39.90	
Grand Total:							4852.21	

Press <ENTER> to continue... <ENTER>

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 5

Please enter the SKU: 888

SKU: 888

Name: Apples

Price: 3.99

Quantity: 10

Minimum Qty: 2

Is Taxed: No

Item already exists, Update? (Y)es/(N)o: y

Enter new data:

SKU: 888

Name: Red Delicious Apples

Price: 3.50

Quantity: 10

Minimum Qty: 2

Is Taxed: n

Overwrite old data? (Y)es/(N)o: y

--== Updated! ==--

Press <ENTER> to continue... <ENTER>

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name



0- Exit program

> 1

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
-----+-----+-----+-----+-----+-----+-----+-----+-----								
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	240	Blueberries	3.99	No	30	5	119.70	
4	916	Seedless Grapes	10.56	No	20	3	211.20	
5	385	Pomegranate	2.50	No	999	2	2497.50	
6	495	Banana	0.44	No	100	30	44.00	
7	316	Kiwifruit	0.50	No	123	10	61.50	
8	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
9	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
10	359	Beefsteak Pie	5.29	Yes	40	5	239.11	
11	127	Curry Chicken	4.79	Yes	30	3	162.38	
12	238	Tide Detergent	16.99	Yes	10	2	191.99	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Jack's Apple Juice	1.50	Yes	80	10	135.60	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
19	222	Peaches	1.44	No	14	20	20.16	***
20	888	Red Delicious Apples	3.50	No	10	2	35.00	
-----+-----+-----+-----+-----+-----+-----+-----+-----								
Grand Total:							4847.31	

Press <ENTER> to continue... <ENTER>

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item

7- Search by name

0- Exit program

> 0

Exit the program? (Y)es/(N)o: Let me think!

Only (Y)es or (N)o are acceptable: n

1- List all items

2- Search by SKU

3- Checkout an item

4- Stock an item

5- Add new item or update item

6- delete item

7- Search by name

0- Exit program

> 0

Exit the program? (Y)es/(N)o: y

## OPEN SUBMISSION:

In “**comments.txt**”, explain why you have chosen the open submission rather than one of the other two.

If not on matrix already, upload your **144\_ms4.c** and **comments.txt** to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms4 144_ms4.c<ENTER>
```

This command will compile your code and name your executable "**fp**"

Execute **fp** and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_fp_open <ENTER>
```

and run your application using the **RED underlined bold *italic*** values in the long output section above.

Please note that a successful submission does not guarantee full credit for this workshop.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

## **MILESTONE 6, THE EXTRAS: DELETE AND SEARCH:**

Implement options 6 and 7 of the menu to work as follows:

### **DELETE:**

Ask for an SKU:

"Please enter the SKU: "

and search for the item using **locateItem** function. If not found, display:

"SKU not found!\n"

And display the menu after a pause.

If item was found, display it and then, display:

"This Item will be deleted, are you sure? (Y)es/(N)o: "

and ask for user's confirmation to delete the item from the list. If the user enters "N" for no, display:

"Aborted!\n"

and display the menu after a pause. If user enters "Y" for yes, then overwrite the data file by saving the items one by one, skipping the one you want to delete. Then load the items back and display:

"Deleted!\n"

And display the menu after a pause.

----- Grocery Inventory System -----

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 6

Please enter the SKU: 444

SKU not found!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 6

Please enter the SKU: 238

SKU: 238

Name: Tide Detergent

Price: 16.99

Quantity: 10

Minimum Qty: 2

Is Taxed: Yes

This Item will be deleted, are you sure? (Y)es/(N)o: n

Aborted!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item

- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 6

Please enter the SKU: 238

SKU: 238

Name: Tide Detergent

Price: 16.99

Quantity: 10

Minimum Qty: 2

Is Taxed: Yes

This Item will be deleted, are you sure? (Y)es/(N)o: y

Deleted!

Press <ENTER> to continue...

- 1- List all items
- 2- Search by SKU
- 3- Checkout an item
- 4- Stock an item
- 5- Add new item or update item
- 6- delete item
- 7- Search by name
- 0- Exit program

> 1

Row	SKU	Name	Price	Taxed	Qty	Min	Total	Atn
1	275	Royal Gala Apples	4.40	No	10	2	44.00	
2	386	Honeydew Melon	5.99	No	20	4	119.80	
3	916	Seedless Grapes	10.56	No	20	3	211.20	
4	385	Pomegranate	2.50	No	5	2	12.50	
5	495	Banana	0.44	No	100	30	44.00	
6	316	Kiwifruit	0.50	No	123	10	61.50	
7	355	Chicken Alfredo	4.49	Yes	20	5	101.47	
8	846	Veal Parmigiana	5.49	Yes	3	5	18.61	***
9	359	Beefsteak Pie	5.29	Yes	40	5	239.11	
10	127	Curry Chicken	4.79	Yes	30	3	162.38	
11	324	Tide Liq. Pods	10.49	Yes	40	5	474.15	
12	491	Tide Powder Det.	10.99	Yes	50	5	620.93	
13	538	Lays Chips S&V	3.69	Yes	1	5	4.17	***
14	649	Joe Org Chips	3.29	Yes	15	5	55.77	
15	731	Allen's Apple Juice	1.79	Yes	100	10	202.27	
16	984	Coke 12 Pack	6.69	Yes	30	3	226.79	
17	350	Nestea 12 Pack	7.29	Yes	50	5	411.88	
18	835	7up 12 pack	6.49	Yes	20	2	146.67	
Grand Total:							3157.21	

Press <ENTER> to continue...

## SEARCH:

Search for an item and display it using part of its name.

Display:

```
"Please enter part of the name: "
```

Then receive a string including spaces.

Loop through the items array and using strstr function in <string.h> check each of item's name for a substring matching the entered string. If there is a match display the item and continue until the end of the items array.

If there is no item name that includes the entered substring, display:

```
"No Item found in the system matching the value entered!\n"
```

Then exit search, pause and display the menu.

strstr(StringTobeChecked, SubString).

strstr searches in the first argument for the first appearance of the second argument, if a match is found, it returns a nonzero value. If no match is found it returns NULL.

Sample output:

```
----- Grocery Inventory System -----
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 2
Please enter part of the name: asdf
No Item found in the system matching the value entered!
Press <ENTER> to continue...
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
> 2
Please enter part of the name: Tide
    SKU: 324
    Name: Tide Liq. Pods
```

```

    Price: 10.49
    Quantity: 40
Minimum Qty: 5
    Is Taxed: Yes
Press <ENTER> to continue...
    SKU: 491
    Name: Tide Powder Det.
    Price: 10.99
    Quantity: 50
Minimum Qty: 5
    Is Taxed: Yes
Press <ENTER> to continue...
1- List all items
2- Search by SKU
3- Checkout an item
4- Stock an item
5- Add new item or update item
6- delete item
7- Search by name
0- Exit program
>

```

## EXTRAS SUBMISSION:

If not on matrix already, upload your [144\\_ms4.c](#) to your matrix account. Compile your code as follows:

```
> gcc -Wall -o extras 144_ms4.c<ENTER>
```

This command will compile your code and name your executable “[extras](#)”

Execute [extras](#) and make sure everything works properly.

Finally run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

```
~profname.proflastname/submit 144_fp_extras <ENTER>
```

and run your application demonstrating the delete and name search functionalities.

Note that this is an open submission and your output does not need to exactly match the output above.

If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

