IPC144SCP 2191.

Grocery Store Inventory System

Subject Workshop / Project
Workshop 7 – Milestone 7-8 V0.8 (draft)

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

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 a 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 several milestones and therefore the same number of deliverables. All but deliverables are your workshops for the second half of the semester and the last deliverable is your final project for the semester. 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 do, you can submit that milestone and start the next.

FILE STRUCTURE OF THE PROJECT

For each milestone, you will create or modify a header file (.h) and a source file (.c). The file names must exactly match to what is asked in the milestone, otherwise it can not be submitted. Each milestone includes a tester program file 144_wXmY.c that contains the main() tester program provided by your professor to test your implementation of the functions of the milestone.

> X is replaced with the workshop number and Y with the milestone number, therefore the name of each tester will be like:
> 144_w4m1.c
> 144_w4m2.c
> 144_w5m3.c
> 144_w6m4.c and so on….

These 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 the header and source files requested and 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. Make sure you do not make any permanent 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.

MILESTONE 1 (WS4): THE USER INTERFACE TOOLS

In interface.c code the following functions and add their prototype in interface.h :

USER INTERFACE TOOLS

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

> Prints the following line and goes to newline twice
> ```
> >---=== Grocery Inventory System ===---<
> ```

```c
void printHeader(void);
```

> Prints the following two lines and goes to newline

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

```c
void printFooter(double gTotal);
```

Prints the following line and goes to newline

```
      >-------------------------------------------------+---------------<
```

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

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

Use this format specifier for printing gTotal : **%12.2lf**

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

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

The function is already coded in class, see: https://github.com/Seneca-144100/IPC144SCP-Notes/blob/master/ClassNotes/07-Feb20/myFuncs.c

```c
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.

```c
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.

The function is already coded in class, see: https://github.com/Seneca-144100/IPC144SCP-Notes/blob/master/ClassNotes/07-Feb20/myFuncs.c

```c
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.

`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.

The function is already coded in class, see: https://github.com/Seneca-144100/IPC144SCP-Notes/blob/master/ClassNotes/07-Feb20/myFuncs.c

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>
Testing Data Entry functions:
Testing getInt
Enter an integer: one
Invalid integer, please try again: 1
You entered: 1
Testing GetIntLimited
Enter an integer between 10 and 20: 1
Invalid value, 10 < value < 20: 30
Invalid value, 10 < value < 20: 11
Your entered 11
Testing Get Double
Enter a floating point number: one
Invalid number, please try again: 1.1
You entered: 1.10
Testing Get Double Limited
Enter a floating point number between 10.00 and 20.00: 1.1
Invalid value, 10.000000< value < 20.000000: 30.1
Invalid value, 10.000000< value < 20.000000: 11.1
You entered: 11.10
Test entries again? (Y/N): no
Only (Y) or (N) are acceptable: n
End of tester program for IO tools!
```

MILESTONE 1 SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h** and professor's **144_ms1_tester.c** to your matrix account. Compile your code as follows:
```
> gcc -Wall -o ms1 interface.c ms1tester.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  1 <ENTER>
To check due dates:
~profname.proflastname/submit 144SCP_w4ms1 -due<ENTER>
To See submission options:
~profname.proflastname/submit<ENTER>
```

and follow the instructions.

Note
Use the same inputs (shown in red) as shown in the sample output's described previously in this document.

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 (WS4): APP INTERFACE

THE APPLICATION USER INTERFACE SKELETON

In grapp.c code the following functions and add their prototype in grapp.h :

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,
First include "interface.h" and "grapp.h" in grapp.c.
Then create the following two functions.

```
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 and prints `Goodbye!`, 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 < value < 7: 1
List Items under construction!
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
Search Items under construction!
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
Checkout Item under construction!
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
Stock Item under construction!
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
Add/Update Item under construction!
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
> 6
Delete Item under construction!
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
> 7
Search by name under construction!
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 : 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
Goodbye!
```

MILESTONE 2 SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h , grapp.c** , **grapp.h** and professor's **144_ms2_tester.c** to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms2 interface.c grapp.c ms2tester.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 144SCP_w4ms2 <ENTER>
```

and follow the instructions.

Note

Use the same inputs (shown in red) as shown in the sample output's described previously in this document.

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 (WS5): THE ITEM INPUT/OUTPUT

Before starting milestone 4 add compilation safeguards to all your header files in the project, and from now on any header file you create, make sure the compilation safeguards are added:

HOW TO ADD COMPILATION SAFEGUARDS:
A compilation safeguard is added to a header file as follows:
If the name of the header file is "abcd.h" then at the first line of header file (before all other code) add these two lines:

```
#ifndef ABCD_H__
#define ABCD_H__
```

Then after the last line of the header file add the following line

```
#endif
```

For example safeguard lines for your interface.h will be:

```
#ifndef INTERFACE_H__
#define INTERFACE_H__

Your function prototypes go here...

#endif
```

In item.h code the following define statements and the Item structure:

Define the following values (using #define)

```
LINEAR       to be  1
FORM         to be  0
STOCK        to be  1
CHECKOUT     to be  0
MAX_ITEM_NO  to be  21
MAX_QTY      to be  999
SKU_MAX      to be  999
SKU_MIN      to be  100
```

These values will be used throughout the project development.

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

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

```c
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.

In `item.c` implement the following `Item` related functions and add their prototype in `item.h` :

```c
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).

```c
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.

```c
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 1-2).

Data entry ranges: (enforce with your get…Limited() functions)

Price: Limited between 0.01 and 1000.00 inclusive

Quantity: Limited between 1 and `MAX_QTY` inclusive

Minumum Quantity: Limited between 0 and 100 inclusive

Here is the format of the data Entry: (<u>Underlined</u> *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);
```

The values are printed as follows
(Example for a non taxed item):

```
>          SKU: 999<
>         Name: Red Apples<
>        Price: 4.54<  Two digits after the decimal point
>     Quantity: 50<
>Minimum Qty: 5<
>      Is Taxed: No<
```

(Example for Taxed item):

```
>          SKU: 999<
>         Name: Red Apples<
>        Price: 4.54<
>     Quantity: 2<
>Minimum Qty: 5<
>      Is Taxed: Yes<
```

If the Quantity is low (use `isLowQuantity() function) add the following line at the end`:

```
>WARNING: Quantity low, please order ASAP!!!<
```

MILESTONE 3 SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h , grapp.c** , **grapp.h, item.c, item.h** and professor's **ms3tester.c** to your matrix account. Compile your code as follows:
```
> gcc -Wall -o ms3 interface.c grapp.c item.c ms3tester.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 144SCP_w5ms1 <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 (WS5): THE ITEM LISTING AND SEARCH

In item.c implement the following `Item` related functions and add their prototype in item.h :

```
void displayItemInLine(struct Item item);
```

This function receives and `Item` structure as an argument.
The Item values are printed 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 (if isTaxed is 1 print Yes, otherwise print No)
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 (totalAfterTax())
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|***<
```

```
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);
```

> This function receives a constant array of Items and their number. Also it receives an sku to look for in the Item array.
> The function loops through all the elements of the item array and if the SKU of an element is a match to the sku passed through the argument list, it will return that index and ends the function.
> If the function reaches the end of the array and no matching sku is found, it will return (-1);

MILESTONE 4 SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h , grapp.c** , **grapp.h, item.c, item.h** and professor's **ms4tester.c** to your matrix account. Compile your code as follows:
```
> gcc -Wall -o ms4 interface.c grapp.c item.c ms4tester.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 144SCP_w5ms4 <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 5 (WS6): ITEM STORAGE AND RETRIEVAL IN ARRAY

In this milestone, you are implementing a function that works with an array of **Items.**
Implement workshop 6 (milestones 5 and 6) functions in the item module (itemSRA.h, itemSRA.c)

Include item.h in itemSRA.h and then add the following 3 prototypes:

```
int findItemIndex(const struct Item item[], int numOfRecs, int* indexPtr);
void updateItem(struct Item* itemptr);
void addItem(struct Item item[], int *NoOfRecs, int sku);
```

Implement the following function itemSRA.c:
```
int findItemIndex(const struct Item item[], int numOfRecs, int* indexPtr)
```

findItemIndex() prompts the user to enter the SKU, then it calls **locateItem()** passing the item array, numOfRecs and SKU to the function. If an index was found it sets the target of indexPtr to the index of the matched item in the array and returns true (1). If no match is found in the array, it will return false (0).

DETAILS
Prompt:
>Please enter the SKU: <
then receive an integer between SKU_MIN and SKU_MAX. (use the functions in the interface module)
Call the **locateItem()** function and see if the item is found.
If found set the target of indexPtr to the found index and return true (1)
If not found only return false (0).

MILESTONE 5 SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h , grapp.c , grapp.h, item.c, item.h, itemSRA.c, itemSRA.h** and professor's **ms5tester.c** to your matrix account. Compile your code as follows:
> gcc -Wall -o ms5 interface.c grapp.c item.c itemSRA.c ms5tester.c <ENTER>

This command will compile your code and name your executable "**ms5**"

Execute ms5 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 144SCP_w6ms5 <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 6 (WS6): ITEM STORAGE AND RETRIEVAL IN ARRAY

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

updateitem, modifies the fields of an Item. The function receives the address of an already existing 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:
>Item updated!< and go to new line
Otherwise print:
>Overwrite 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:
>Item 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.

MILESTONE 6 SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h** , **grapp.c** , **grapp.h, item.c, item.h, itemSRA.c, itemSRA.h** and professor's **ms5tester.c** to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms6 interface.c grapp.c item.c itemSRA.c ms5tester.c
<ENTER>
```

This command will compile your code and name your executable "**ms6**"

Execute ms6 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 144SCP_w6ms6 <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.

WORKSHOP 7 MILESTONE 7: FILE IO

In this milestone, you are implementing two functions that save and load an **Item** into and from a file. Implement workshop 7 (milestones 7 and 8) functions in the file module (grfile.h, grfile.c)

ADJUSTMENTS TO ITEMENTRY FUNCTION IN MILESTONE 3:

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<
```

> 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 two functions:

```c
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.

```c
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.

MILESTONE 7 (WS7) SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h** , **grapp.c** , **grapp.h, item.c, item.h, itemSRA.c, itemSRA.h , grfile.c, grfile.h** and professor's **ms7Tester.c** to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms7 interface.c grapp.c item.c itemSRA.c grfile.c
ms7Tester.c <ENTER>
```

This command will compile your code and name your executable "**ms7**"

Execute ms7 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 144SCP_w7ms7 <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.

WORKSHOP 7 MILESTONE 8: FILE IO

In this milestone, you are implementing two functions that save and load an array of **Items** into and from a file. Implement workshop 7 (milestones 7 and 8) functions in the file module (grfile.h, grfile.c)

```
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**) if at least it read minimum of one record from the file.

MILESTONE 8 (WS7) SUBMISSION

If not on matrix already, upload your **interface.c** , **interface.h , grapp.c** , **grapp.h, item.c, item.h, itemSRA.c, itemSRA.h , grfile.c, grfile.h** and professor's **ms8Tester.c** to your matrix account. Compile your code as follows:

```
> gcc -Wall -o ms8 interface.c grapp.c item.c itemSRA.c grfile.c
ms8Tester.c <ENTER>
```

This command will compile your code and name your executable "**ms8**"

Execute ms8 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 144SCP_w7ms8 <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.