IPC144SCP 2191.

Grocery Store Inventory System

Subject Workshop / Project
Milestone 1-2 V0.9 (submitter is not set up yet)

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

> **Commented [CG1]:** Removed optional delete, search, and sort modules which are part of milestone #5

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: THE USER INTERFACE TOOLS

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

USER INTERFACE TOOLS

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

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

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

> Prints the following two lines and goes to newline

Commented [CG2]: Replaced abbreviated function name

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

`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**

`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

`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.

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

> **Commented [CG5]:** Replaced abbreviated function name

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);
```

> **Commented [CG6]:** Replaced abbreviated function name

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>
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
Is this milestone 1? (Y/N): Yes
Only (Y)es or (N)o are acceptable: y
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 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 144SCP_w4ms1 <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: 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 create the following three 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.

Commented [CG7]: Replaced abbreviated function name

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

Commented [CG8]: Leave this? Doesn't hurt I suppose – but the students will never get to do this part

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:

```
---=== 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...
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
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 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 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.