Computing Problem 3a: Manipulating an Array of Structures

Objectives:

1. To apply modular/structured programming
2. To define the structure of an array of structures
3. To insert records into the array of structures
4. To update an existing record in the array of structures
5. To delete an existing record from the array of structures
6. To search and display records existing in the array of structures

Instructions:

Write a modular program for the specifications below. **All functions must have parameters** (except for the Menu from which the user chooses a task). All identifier names must be descriptive.

1. Simulate a grocery store's maintenance (adding, editing, deleting) of item records. The user should be able choose from any of the following modules:

   a) Add an Item Record

   Your structure will have the following fields: (decide on your identifier names)

   Item number (integer) – a primary key; cannot be duplicated; a 5-digit number
   Item name (string)
   Item company (string)
   Item type (string) (e.g., canned goods, dairy products, vegetables, baked goods, pasta, sauces, etc.)
   Item Selling Price (float)
   Quantity left or stock-on-hand (integer) – the number of items in stock
   Reorder Point (integer) – the number of items that determines if the item needs to be reordered from the supplier

   The items must be stored in a way that the item numbers are in ascending order.

10123	12346	25978		
Corned Beef 380g	Fortified Milk 1L	Spaghetti Sauce 500g		
Purefoods	Selecta	Del Monte		
Canned Goods	Dairy Product	Sauces		
166.00	75.00	58.85		
500	100	190		
200	100	200		
[0]	[1]	[2]	…	[SIZE-1]

   This means that if a record that will be added has an item number of 10001, it will be stored in the first location. The existing records must be moved forward.

10001	10123	12346	25978		
Soy Sauce 1L	Corned Beef 380g	Fortified Milk 1L	Spaghetti Sauce 500g		
Silver Swan	Purefoods	Selecta	Del Monte		
Condiment	Canned Goods	Dairy Product	Sauces		
55.00	166.00	75.00	58.85		
100	500	100	190		
50	200	100	200		
[0]	[1]	[2]	[3]	…	[SIZE-1]

b) *Search and Edit a Record*

Let the user enter the item name or item type. The user should be able to search for the item (name or type) in any case (lower- or uppercase) -- you can use the string function *strcmpi()* for this. If there are two or more records with the same type, show all of them then let the user choose the item name. Only allow editing on the selling price and quantity fields.

c) *Search and Delete a Record*

Search for the record as with the editing process. When deleting records in the first or middle locations, the components in the array should be moved <u>backwards</u>.

For example:

10123 Corned Beef 380g Purefoods Canned Goods 166.00 500 200	12346 Fortified Milk 1L Selecta Dairy Product 75.00 100 100	25978 Spaghetti Sauce 500g Del Monte Sauces 58.85 190 200		
[0]	[1]	[2]	…	[SIZE-1]

If the user decides to delete **<u>Fortified Milk 1L</u>**, the record of *Spaghetti Sauce 500g* will be moved backward:

10123 Corned Beef 380g Purefoods Canned Goods 166.00 500 200	12346 Fortified Milk 1L Selecta Dairy Product 75.00 100 100	25978 Spaghetti Sauce 500g Del Monte Sauces 58.85 190 200		
[0]	[1]	[2]	…	[SIZE-1]

Result:

10123 Corned Beef 380g Purefoods Canned Goods 166.00 500 200	25978 Spaghetti Sauce 500g Del Monte Sauces 58.85 190 200			
[0]	[1]	[2]	…	[SIZE-1]

d) *Filter out (search for and display) all the records that are equal to or below the reorder point*.

If this is the original content of your array of structures, then your program must output items 12346 and 25978 (including the data the goes with these item numbers)

10123 Corned Beef 380g Purefoods Canned Goods 166.00 500 200	12346 Fortified Milk 1L Selecta Dairy Product 75.00 **100** 100	25978 Spaghetti Sauce 500g Del Monte Sauces 58.85 **190** 200		
[0]	[1]	[2]	…	[SIZE-1]

   e) ***Filter out all the records of a particular item type -*** *tabular form*
   f) ***Filter out all the records of a particular item company -*** *tabular form*
   g) ***Display all records -*** *tabular form (continue to the next empty row if the records cannot fit in the original display area)*
   h) ***Exit*** *(from the program)*

2. Allow the user to repeat the whole process (through the Menu) as often as he or she wants.
3. Trap all possible user errors. Output error messages where applicable and guide the user to correct the mistake.
4. Include *informational messages* (e.g., "The record was successfully saved."), *error messages* (e.g., "That item number is out of range."), and *warning or confirmation messages* (e.g., "You are about to delete this record. Do you want to continue (Y/N)? ") in your program where they are applicable.
5. Save your C program as ***<FamilyName>CP3a.c*** (e.g., *RiveraCP3a.c*) – DO NOT save it as .cpp
6. Upload your C file to assignment link in the Virtual Classroom

REMINDERS*:*

- **Do not forget to place the required comments prior to the preprocessing directives. Otherwise, deductions will apply.**
- **Follow the naming convention as specified above (green highlight). Otherwise, your submission will not be checked. Therefore, you will get a zero (0) for this activity.**
- **Copying another student's work is not allowed. Similar assignments, especially similar identifiers, sequence of a function's process, and mistakes, will get a zero (0) for this activity.**