CS262, Lab Assignment 9:

Dynamic Memory, Structures and Arrays

Due: Sunday, April 17 at 11:59 pm ET

Description:

In this exercise, you will work with structures, arrays, and dynamic memory allocation to develop a simple menu-driven inventory management system. This program uses **an array of structures** as a database. Specifically, you will develop a simple database of information about items stored in a warehouse. The program is built around array of structures, with each structure containing information about the items.

Define a **structure** appropriate for holding the following members of an item:

```
itemId        (int)
itemName      (string)
quantity      (int)
pricePerItem  (double)
```

Use `typedef` to define a new type `Item` for this structure.

- Define an initial array `itemInventory` of size `N`, where `N=5`. Declare `N` as global variable.
- You need to use `malloc()` or `calloc()` to create the array.

Your main function should display the following menu:

```
*********************************************
Enter your Choice:
'i' - insert an item
'u' - update the database
's' - search the database
'd' - display the database
'q' - quit the program
*********************************************
```

Note: The user is allowed to enter the menu choice both in lower and upper case, i.e., `q` and `Q` are both valid choices. If the input choice is not valid print the message "**%c is not a valid choice**" (where **%c** is replaced with the user input) and ask the user to re-enter the choice.

Your program will support the following menu choices:

- **Insert an item:** Insert a new item in the database after accepting all the necessary information about the item from the user. Your program should print an error message "**Error Inserting an Item**", if the `ItemId` is already in the database. The database is not going to be full. This is contrary to the static array characteristics. If the database is full, you should expand the array by doubling the current size (do not forget to update `N` with the new size). Note that you can use the `realloc()` function here. This menu choice prompts the user for all item information and calls the `insterItem()` function. Use the following function prototype:

  ```
  void insertItem(Item *itemInventory, Item item)
  ```

  - where `itemInventory` is a pointer to the database (i.e., a pointer to the struct `Item`)

- **Update the database:** Given the `itemId` and a new `quantity` of an item, update the quantity of the specific `itemId` in the database. If the `itemId` does not exist in the database, your program should display **"Item Not Found".** This menu choice prompts the user for the `itemId` and the new `quantity` and calls the `updateItem()` function. Use the following function prototype:

  ```
  void updateItem(Item *itemInventory, int itemID, int quantity)
  ```

- **Search the database:** Given an `ItemId`, search the database for the item information and display the id, name, the quantity, and price per item. Display in the following format:

  ```
  ********************
  Item ID: 005
  Item Name: TV
  Item Quantity: 2500
  Price per Item: 1500.00
  ************************
  ```

  If the `ItemId` does not exist, your program should display **"Item Not Found"**. This choice prompts the user for the `ItemId` and calls `searchItem()` function. Use this function prototype:

  ```
  void searchItem(Item *itemInventory, int itemID)
  ```

- **Display the database:** Print a table showing all the items' information in the database. The table displays as:

  ```
  ********************************************************
   Item ID        Item Name     Item Quantity   Item Price
  ********************************************************
   001            Pen           200             0.50
   002            Desk          350             100.00
  ```

  Use the following function prototype:

  ```
  void printData(Item *itemInventory)
  ```

- **Quit the program:** This option calls the `quit()` function to deallocate all the dynamic memory and exits the program. Use the following function prototype:

  ```
  void quit(Item *itemInventory)
  ```

Note: In order to decide if the database is full, you have to keep track of the number of elements in the database and the size of the database. You are allowed to define your own helper functions.

Makefile:

You will compile your program with a Makefile using the gcc compiler. Edit the Makefile from Lab 8, but make sure that your Makefile does not contain any references to any programs and files other than those necessary for this lab.

Remember that your code should compile without any warnings.

Submission:

You will submit a typescript file similar to the one of previous Labs:

1. Create a typescript file named `lab9_typescript_<username>_<labsection>`
2. Show that you are logged onto Zeus
3. Show the content of your source code.
4. Compile the code using your `Makefile`
5. Show a detailed listing of the directory
6. Run the code using the examples described previously
7. Remove the executable using your `Makefile`
8. End the typescript
9. Be sure your directory ONLY contains the *source file*, *script* and `Makefile`
10. Create a `tarfile` of your `lab9_<username>_<labsection>` directory
11. Submit the `tarfile` to Blackboard
12. Verify that your submitted `tarfile` can be extracted and it's the right `tarfile`.

Congratulations! You have completed your assignment