

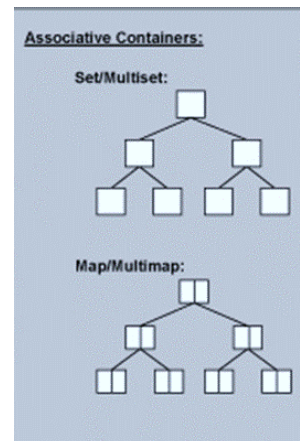
CPSC 131, Data Structures – Fall 2023

Homework 4: Ordered Associative Containers



Learning Goals:

- Familiarization and practice with key/value association data structure usage and binary search tree concepts
- Familiarization and practice using the same key across associative containers to join the contents into a single view
- Familiarization and practice using the STL's map container interface
- Reinforce the similarities and differences between sequence and associative containers
- Reinforce reading persistent data from disk files and storing in memory resident data structures
- Reinforce modern C++ object-oriented programming techniques



Description:



This GroceryStore Inventory project builds on the GroceryItem, Shopping Cart, and GroceryItemDatabase from previous assignments by adding the maintenance of a grocery store's inventory at the checkout counter. Here you are given a collection of customers (e.g., Woodstock, Lucy, Carlie) pushing their shopping cart, each already filled with the groceries from their shopping list. Each customer goes through the checkout line where the groceries are scanned and a receipt is given. As groceries are scanned, the store's inventory is updated by reducing the number of groceries on hand for that grocery item. At the end of the day after all customers have been processed, you take inventory and reorder groceries that are getting low.

You are provided with starter code that together with your work from previous assignments form your point of departure to complete this assignment.

1. **main.cpp** – Function main() orchestrates the flow of execution. A partial implementation has been provided. You are to complete the implementation.
2. **GroceryItem.hpp/GroceryItem.cpp** – Reuse the GroceryItem class from your previous assignments. Unless you find an error or omission in your code, these files should require no modification.
3. **GroceryItemDatabase.hpp/GroceryItemDatabase.cpp** – A partial implementation of the GroceryItemDatabase class has been provided. The previous assignment used the sequential container `std::vector` for the memory resident database. This assignment uses the associative container `std::map`, a binary search tree, associating a UPC with a grocery item. An updated GroceryItemDatabase.hpp incorporating this change is provided and requires no modifications. You are to modify GroceryItemDatabase.cpp by completing the implementation. Leverage the code you wrote in the last assignment adjusting for `std::map` vice `std::vector`. Function `find()`, for example, will no longer be implemented as a recursive linear search function. Instead, it should delegate to the map's binary search function `find()`. If duplicate keys are discovered during construction, use the first occurrence.

INVENTORY TRACKER										breakfast	lunch	dinner	snacks
Ingredient	UPC	Price (per unit)	Quantity	Unit Price	Total Price	Category	Brand	Weight	Volume	Price (per unit)	Quantity	Unit Price	Total Price
1. Cereal Box	0.5	0.5	0	1.5	0	1	0.5	0.50					
2. Wheat Flour	0.5	0.5	0	1.5	0	1	0.5	0.50					
3. Macaroni Box	2	0.5	0	1	0	1	0.5	1.00	1.00				
4. Can of Tuna	1	0.5	0	1	0	1	0.5	1.00					
5. Spaghetti Box	1	0.5	0	1	0	1	0.5	1.00					
6. Can of Beans	1	0.5	0	1	0	1	0.5	1.00					
7. Can of tomatoes	1	0.5	0	1	0	1	0.5	1.00					
8. 1 lb ground beef	2	0.5	0	1	0	1	0.5	1.00	1.00				
9. Pasta Sauce	1	0.5	0	1	0	1	0.5	1.00					
10. Chicken leg of 6	1	0.5	0	1	0	1	0.5	1.00					
11. Cream of Chicken	2	0.5	0	1	0	1	0.5	1.00					
12. Cream Soup Mix	1	0.5	0	1	0	1	0.5	1.00					
13. Popcorn bag	1	0.5	0	1	0	1	0.5	1.00					
14. Granola Box	0.5	0.5	0	1.5	0	1	0.5	0.50					
15. Assorted nuts	1	0.5	0	1	0	1	0.5	1.00					

4. **GroceryStore.hpp/GroceryStore.cpp** – A partial implementation of the GroceryStore class has been provided. You are to complete the implementation. GroceryStore.hpp requires no modifications. You are to modify GroceryStore.cpp by completing the implementation.

- a. GroceryStore constructor – This function takes as a parameter the name of the store's inventory database file containing pairs of UPC and quantity, each pair on a separate line. Populate the store's inventory database with the contents of the file. For example, the contents of "GroceryStoreInventory.dat" may look like:

```
0051600080015    36
0019600923015    34
0688267141676    36
0657622604842    25
```

In this assignment, there are two text files used to persist data. Don't confuse this file (GroceryStoreInventory.dat) that persists a store's inventory with the file (Grocery_UPC_Database-Large.dat) used by class GroceryItemDatabase above that persists the full description and price of a grocery item. Both use the UPC as the key but contain quite different information.

- b. inventory – This function takes no arguments and returns a reference to the store's inventory database. The inventory, designed as a binary search tree and implemented using the STL's map class, is an association between UPC (the key) and the quantity on hand (the value). As a convenience, an alias called Inventory_DB has been created as part of the GroceryStore interface in GroceryStore.hpp. Use this alias in your work everywhere as appropriate.
- c. ringUpCustomers – This function takes a collection of shopping carts (i.e., a collection of customers with their shopping cart) and an optional receipt output stream as parameters and returns a collection of unique UPCs for groceries that have been sold. Pretend you have a lot of customers waiting in line to purchase the groceries in their cart. Ring up each customer individually while accumulating the groceries sold. As a convenience, aliases called GroceryItemsSold, ShoppingCart and ShoppingCarts have been created as part of the GroceryStore interface in GroceryStore.hpp. Use these aliases in your work everywhere as appropriate.
- d. ringUpCustomer – This function takes a shopping cart and a receipt output stream as parameters and returns a collection of UPCs purchased by this customer. Scan all the groceries in the shopping cart, print a receipt with an amount due, and deduct the groceries purchased from the store's inventory. The logic to scan groceries and print a receipt should be carried forward from your last assignment. Look at last assignment's posted solution if you need help. Add the grocery item to the collection of groceries purchased.
- e. reorderGroceryItems – This function takes a collection of UPCs for groceries sold and an optional reorder report output stream as a parameter and returns nothing. For each grocery item sold, check the store's inventory for the quantity on hand. If the number of grocery items on hand has fallen below the REORDER_THRESHOLD, print a message indicating the current number on hand then order LOT_COUNT more by adding LOT_COUNT to the current number on hand. A sample report might look like:

Re-ordering grocery items the store is running low on.

```
1: {"00025317533003", "Applegate Farms", "Hotdog Uncrd Big Apple", 12.57}
   only 8 in stock after selling 1 unit(s) below threshold (15), re-ordering 20 more

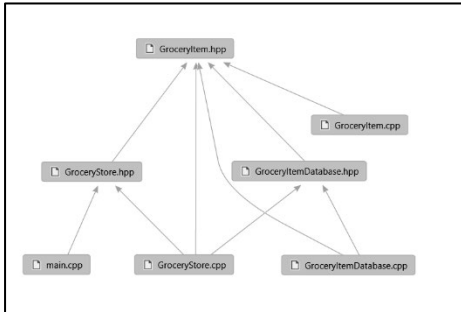
2: {"00041331092609", "Goya", "Frz Spnsh Omelet", 16.84}
   *** no longer sold in this store and will not be re-ordered

3: {"00070596000647", "Ultra Glow", "Ultra Glow Black Soap", 21.19}
   only 12 in stock after selling 1 unit(s) below threshold (15), re-ordering 20 more
```

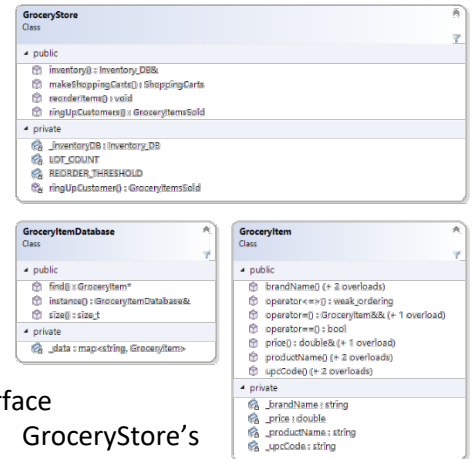
Program Overview:

Sometimes it helps to see pictures of what you're building. The following is just for information.

Class Structure: To help you visualize the interface your three classes provide, and to summarize instance attributes for the developer, the class diagram on the right is provided.

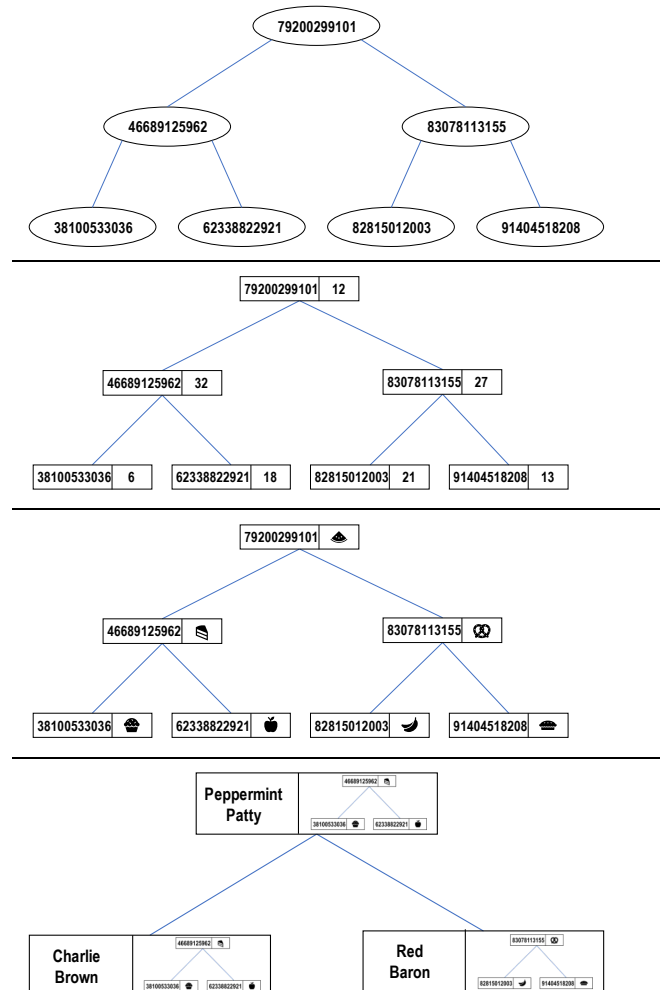


File Dependencies: To help you visualize the dependencies between included files, the diagram on the left is provided. Notice that each class has a header and source file pair, and that the class's source file always includes the class's header file. In this assignment, main.cpp requires only GroceryStore's interface (header file). GroceryStore's interface requires only GroceryItem's interface, but GroceryStore's implementation (source file) requires GroceryItem's, GroceryItemDatabase's, and of course GroceryStore's interfaces.



Data Types: To help you visualize the data structures, types and aliases, the following is provided:

- **GroceryItemsSold:** a set of unique UPCs organized as a binary search tree.
- **Inventory_DB:** an association from UPC to quantity on hand organized as a binary search tree.
- **ShoppingCart:** a collection of groceries indexed by UPC organized as a binary search tree.
- **ShoppingCarts:** a collection of shopping carts indexed by customer's name organized as a binary search tree. Analogous to a matrix being an array of arrays, ShoppingCarts is a tree of trees.



Rules and Constraints:

1. You are to modify only designated TO-DO sections. **The grading process will detect and discard any changes made outside the designated TO-DO sections, including spacing and formatting.** Designated TO-DO sections are identified with the following comments:

```

//////////////////// TO-DO (X) //////////////////////
...
//////////////////// END-TO-DO (X) //////////////////////

```

Keep and do not alter these comments. Insert your code between them. In this assignment, there are 13 such sections of code you are being asked to complete. 5 of them are in `main.cpp`, 3 in `GroceryItemDatabase.cpp`, and 5 in `GroceryStore.cpp`.

Reminders:

- The C++ using directive `using namespace std;` is **never allowed** in any header or source file in any deliverable product. Being new to C++, you may have used this in the past. If you haven't done so already, it's now time to shed this crutch and fully decorate your identifiers.
- A clean compile is an entrance criterion. Deliveries that do meet the entrance criteria cannot be graded.
- Always initialize your class's attributes, either with member initialization, within the constructor's initialization list, or both. Avoid assigning initial values within the body of constructors.
- Use Build.sh on Tuffix to compile and link your program. There is nothing magic about Build.sh, all it does is save you (and me) from repeatedly typing the very long compile command and all the source files to compile.
- Filenames are case sensitive on Linux operating systems, like Tuffix.
- You may redirect standard input from a text file, and you must redirect standard output to a text file named `output.txt`. Failure to include `output.txt` in your delivery indicates you were not able to execute your program and will be scored accordingly. A screenshot of your terminal window is not acceptable. See [How to build and run your programs](#). Also see [How to use command redirection under Linux](#) if you are unfamiliar with command line redirection.

Deliverable Artifacts:

Provided files	Files to deliver	Comments
GroceryItem.hpp GroceryItemDatabase.hpp GroceryStore.hpp	1. GroceryItem.hpp 2. GroceryItemDatabase.hpp 3. GroceryStore.hpp	You shall not modify these files. The grading process will overwrite whatever you deliver with the ones provided with this assignment. It is important your delivery is complete, so don't omit these files.
GroceryItem.cpp	4. GroceryItem.cpp	Replace with your (potentially updated) file from the previous assignment.
GroceryItemDatabase.cpp main.cpp GroceryStore.cpp	5. GroceryItemDatabase.cpp 6. main.cpp 7. GroceryStore.cpp	Start with the files provided. Make your changes in the designated TO-DO sections (only). The grading process will detect and discard all other changes. GroceryItemDatabase.cpp is similar to the previous assignment but needs updating to move from <code>std::vector</code> to <code>std::map</code> .
sample_output.txt	8. output.txt	Capture your program's output to this text file and include it in your delivery. Failure to deliver this file indicates you could not get your program to execute.
	readme.*	Optional. Use it to communicate your thoughts to the grader
Grocery_UPC_Database-Large.dat GroceryStoreInventory.dat		Text files with a grocery item's full description and a grocery store's inventory databases. Do not modify these files. They're big and unchanged, so don't include it in your delivery.
RegressionTests/ GroceryItemDatabaseTests.cpp GroceryStoreTests.cpp GroceryItemTests.cpp CheckResults.hpp		When you're far enough along and ready to have your class tested, then place this folder in your working directory. These tests will be added to your delivery and executed during the grading process. The grading process expects all tests to pass.