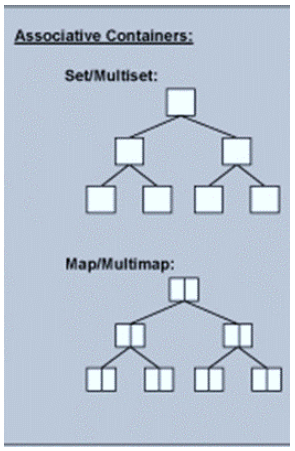
CPSC 131, Data Structures – Fall 2020
Inventory: Associative Containers Homework

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

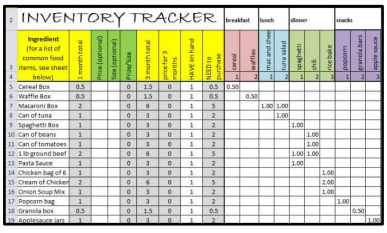


Description:

This Bookstore Inventory project builds on the Book, Book Cart, and Book Database from previous assignments by adding the maintenance of a bookstore'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 books from their shopping list. Each customer goes through the checkout line where the books are scanned and a receipt is given. As books are scanned, the store's inventory is updated by reducing the number of books on hand for that book. At the end of the day after all customers have been processed, you take inventory and reorder books that you're getting low on.

You are provided with starter code that together with 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 modify by implementing the five TO-DO sections.



2. **Book.hpp/Book.cpp** – Reuse the Book class from your previous assignments. Unless you find an error or omission in your code, these files should require no modification.

3. **BookDatabase.hpp/BookDatabase.cpp** – A partial implementation of the BookDatabase 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 an ISBN with a book. An updated (from the last assignment) BookDatabase.hpp incorporating this change is provided, requires no modifications, and will be overwritten during the grading process. You are to modify BookDatabase.cpp by implementing the three TO-DO sections. Leverage the code you wrote in the last assignment, making adjustments 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 find() function.

4. **Bookstore.hpp/Bookstore.cpp** – A partial implementation of the Bookstore class has been provided. You are to complete the implementation. Bookstore.hpp requires no modifications and will be overwritten during the grading process. You are to modify Bookstore.cpp by implementing the four TO-DO sections.

   a. Bookstore constructor – This function takes as a parameter the name of store's inventory database file containing pairs of ISBN 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 "BookstoreInventory.dat" may look like:

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

   b. inventory – This function takes no arguments and returns a reference to the store's inventory database. The inventory, defined as a binary search tree and implemented using the STL's map class, is an association between ISBN (the key) and the quantity on hand (the value). As a convenience, an alias called Inventory_DB has been created as part of the Bookstore interface in Bookstore.hpp. Use this alias in your work as appropriate.

   c. processCustomerShoppingCarts – This function takes a collection of shopping carts (i.e., a collection of customers with their shopping cart) as a parameter and returns a collection of ISBNs for books sold. For each customer, scan all the books in their shopping cart, print a receipt with an amount due, and deduct the books bought from the store's inventory. The logic to scan books 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 book to the collection of books sold. As a convenience, aliases called BooksSold, ShoppingCart and ShoppingCarts have been created as part of the Bookstore interface in Bookstore.hpp. Use these aliases in your work as appropriate.

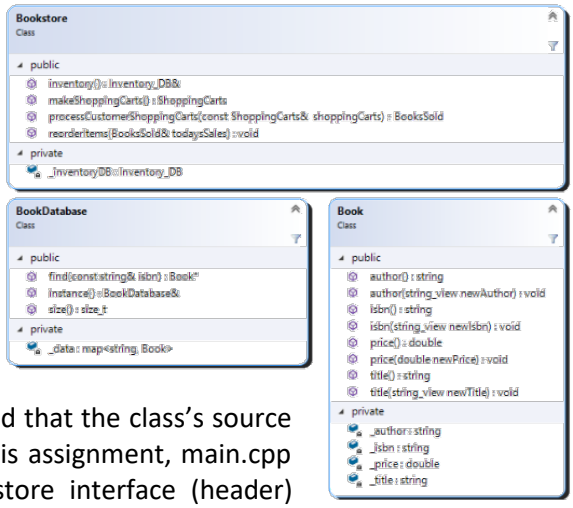
   d. reorderBooks – This function takes a collection of ISBNs for books sold as a parameter and returns nothing. For each books sold, check the store's inventory for the quantity on hand. If there are less than 15 books, print a message indicating the current number on hand then order 20 more by adding 20 to the current number on hand. A sample report might look like:

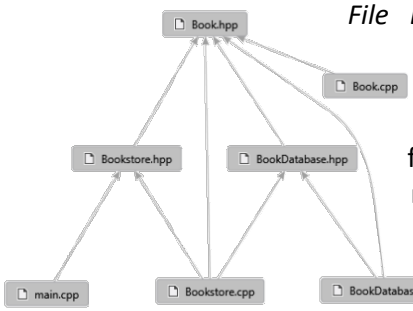
```
Re-ordering books the store is running low on.

1: {"9789999706124", "Professor Bear's Mathematical World (1st edition)", "James N. Boyd", 72.91}
     only 14 remain in stock which is 1 unit(s) below reorder threshold (15), re-ordering 20 more

2: {"9802161748", "Wild mammals of Venezuela (1st edition)", "Rexford D. Lord", 47.74}
     *** no longer sold in this store and will not be re-ordered

3: {"9789999746892", "Children and Television, Vol557 (1st edition)", "Amy B. Jordan", 59.60}
     only 13 remain in stock which is 2 unit(s) below reorder 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 to the class consumer, and to summarize instance attributes for the developer, the class diagram is on the right provided.

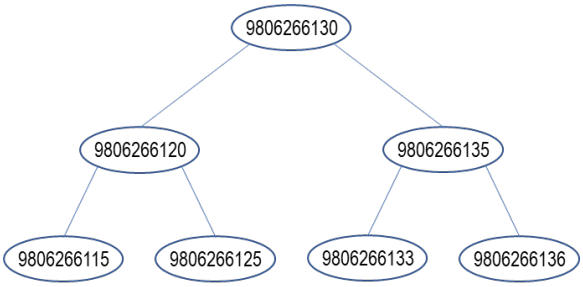




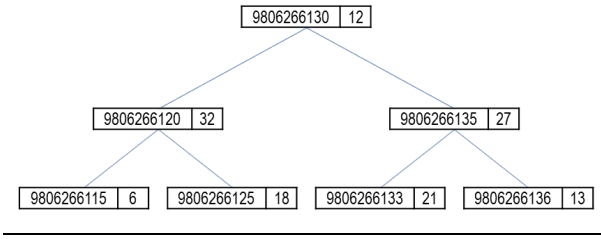
File Dependencies: To help you visualize the dependencies between included files, the diagram on the left is provided. Notice that each class has a header and source file pair, and that the class's source file always includes the class's header file. In this assignment, main.cpp requires only the Bookstore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

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

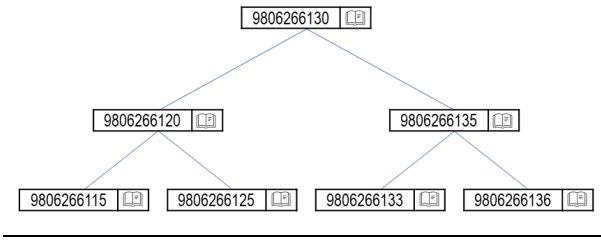
- BooksSold: a set of unique ISBNs organized as a binary search tree.



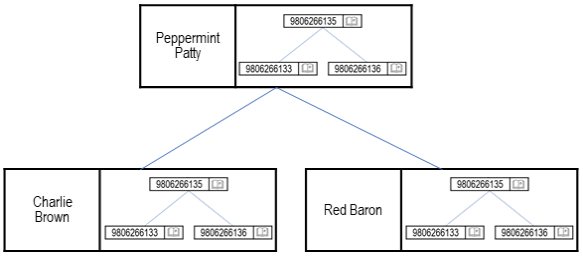
- Inventory_DB: an association from ISBN to quantity on hand organized as a binary search tree.



- ShoppingCart: a collection of books indexed by ISBN organized as a binary search tree.



- ShoppingCarts: a collection of shopping carts indexed by customer's name organized as a binary search tree.



Rules and Constraints:

1. You are to modify only designated TO-DO sections. Do not modify anything outside such designated areas. Designated TO-DO sections are identified with the following comments:

```
//////////////////////// TO-DO ////////////////////////////
...
//////////////////////// END-TO-DO ////////////////////////
```

   Keep these comments and insert your code between them. In this assignment, there are 6 such sections of code you are being asked to complete. 3 of them are in BookDatabase.cpp and 3 are in Bookstore.cpp.

Reminders:

- The C++ using directive `using namespace std;` is never allowed in any header or source file in any deliverable products. Being new to C++, you may have used this is the past. If you haven't done so already, it's now time to shed this crutch and fully decorate your identifiers.
- Use Build.sh to compile and link your program on Tuffix – it employs the correct compile options.
- Filenames are case sensitive, both in source code and in your OS file system. Windows doesn't care about filename case, but Linux does.
- 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 execute 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
Book.hpp BookDatabase.hpp Bookstore.hpp	1. Book.hpp 2. BookDatabase.hpp 3. Bookstore.hpp	You should not modify these files. The grading process will overwrite whatever you deliver with the ones provided with this assignment. It is important that you deliver complete solutions, so don't omit these files in your delivery.
Book.cpp	4. Book.cpp	You should replace the provided file stub with your (potentially) updated files from the previous assignment.
BookDatabase.cpp	5. BookDatabase.cpp	Modify the file as described above leveraging your solution from the previous assignment.
main.cpp Bookstore.cpp	6. main.cpp 7. Bookstore.cpp	Modify the file as described above.
	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.
Open Library Database-Large.dat BookstoreInventory.dat		Text files with a bookstore's item and inventory databases. Do not modify these files. They're big and unchanged, so don't include it in your delivery.
Regression/ BookDatabaseTests.cpp BookstoreTests.cpp BookTests.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.
sample_output.txt sample_test_results.txt		Sample output of a working program. Use for reference, your output format may be different. But the contents should match. Do not include this file with your delivery.