CPSC 131, Data Structures – Spring 2020
Shopping Cart: Container Adapters Homework

Learning Goals:

- Familiarization with stack and queue concepts
- Reinforce the concept of adapting the stack and queue abstract data type to an underlying implementation data structure
- Familiarization and practice using the STL's adapter container interface
- Reinforce modern C++ object-oriented programming techniques

Description:

Continuing with our Grocery Store theme, you are now at the grocery store shopping for the items on your grocery list. As you walk up and down the aisles you place items into your shopping cart, one item on top of the other. The last item you place into your shopping cart will be on top and will be the first thing you take out. In fact, if you want to get to something at the bottom of your cart you'll have to take everything on top of it out first. You're a very smart shopper so you know to start with canned goods first so they won't break, and finish with the eggs last.

As luck would have it, you've almost completed your shopping and have a pretty full cart when the wheel breaks rendering your cart unmovable. Determined to complete your grocery shopping you grab another cart and begin moving items from the broken cart to the new cart when you realize that your breakable grocery items, like eggs, will now be on the bottom. But that's an easy problem to solve, all you have to do is get a third cart and move items between the two new carts so that the breakable items are always on top.

A recursive algorithm to carefully move grocery items from the broken cart to a working cart is:

```
START
Procedure carefully_move_grocery_items (number_of_items_to_be_moved, broken_cart, working_cart, spare_cart)

   IF number_of_items_to_be_moved == 1, THEN
      move top item from broken_cart to working_cart

   ELSE
      carefully_move_grocery_items (number_of_items_to_be_moved-1, broken_cart, spare_cart, working_cart)
      move top item from broken_cart to working_cart
      carefully_move_grocery_items (number_of_items_to_be_moved-1, spare_cart, working_cart, broken_cart)

   END IF

END Procedure
STOP
```

See [Data Structure & Algorithms - Tower of Hanoi](), [Tower of Hanoi video]() or [Tower of Hanoi recursion game algorithm explained]() for more about the recursive algorithm.

Once you fill your shopping cart you'll proceed to the checkout line. When it's your turn, you'll take everything out of the shopping cart and place each item on the counter where they will be scanned and a total calculated. As a grocery item is scanned, the UPC is used to query the store's persistent database for the product's full name, description, and price. You take your receipt and your bags of groceries and leave the store.

The output of your program is an itemized receipt with the total amount due, perhaps something like this:

```
UPC: "00075767200246", Brand: "Perry's Ice Cream", Product: "Perry's Ice Cream Panda Paws", Price: $19.00
UPC: "00888109110154", Brand: "Hostess", Product: "Hostess Mini Muffins Chocolate Chip - 20 Ct", Price: $10.01
UPC: "00723503568678", Brand: "Petmate", Product: "Petmate Booda Bones Steak, Bacon & Chicken Flavors - 9 Ct", Price: $6.96
-------------------------
Total  $35.97
```

How to Proceed:

The following sequence of steps are recommended to get started and eventually complete this assignment.

1. Review the solution to the last homework assignment. Use the posted solution to fix your solution and verify it now works. Your GroceryItem class needs to be working well before continuing with this assignment.
2. Implement the database functions first. Details are embedded in GroceryItemDatabase.cpp.
   a. The constructor should open a text file and populate a memory resident data store (an instance attribute called _*data*) with the contents of the Grocery_UPC_Database_Sample.dat file.
   b. The find() function takes a UPC, searches the memory resident data store, and returns a pointer to the item if found and a null pointer otherwise.
   c. The size() function takes no arguments and returns the number of entries in the database.
3. Implement the segments in main.cpp from top to bottom next. Details are embedded in main.cpp.
   a. Implement the carefully_move_grocery_items recursive algorithm first, then
   b. Snag an empty cart
   c. Shop for a while placing grocery items into my shopping cart
   d. A wheel on your cart breaks so move your grocery items to a new cart that works
   e. Checkout and pay for all this stuff by choosing a checkout line and placing items on the counter's conveyor belt
   f. Scan the items accumulating the amount due and creating a receipt with full product descriptions
      i. Don't assume the grocery item's UPC will be in the store's persistent database.

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 10 such sections of code you are being asked to complete. 7 of them are in main.cpp and 3 are in GroceryItemDatabase.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 – it employs the correct compile options.
- 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**
GroceryItem.hpp GroceryItem.cpp	1. GroceryItem.hpp 2. GroceryItem.cpp	You should replace the provided file stubs with your (potentially) updated files from the previous assignment.
main.cpp GroceryItemDatabase.cpp	3. main.cpp 4. GroceryItemDatabase.cpp	Start with the file provided, make your changes in the designated TO-DO sections (only), and delivery your final solution.
GroceryItemDatabase.hpp	5. GroceryItemDatabase.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.
sample_output.txt	6. 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.
Grocery_UPC_Database_Sample.dat		Text file with a grocery store's database. Do not modify this file. It's big and unchanged, so don't include it in your delivery.
GroceryItemDatabaseTests.cpp GroceryItemTests.cpp CheckResults.hpp		When you're far enough along and ready to have your class tested, then place these files in your working directory. These tests will be added to your delivery and executed during the grading process.