CPSC 131, Data Structures – Fall 2025
Homework 3: Container Adapters

Student Learning Objectives:

- 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
- Increase recursion proficiency
- Demonstrate the ability to translate requirements into solutions in a timely manner
- Reinforce the software development cycle and practice building error free solutions on Linux
- Reinforce modern C++ object-oriented programming techniques

Description:

Continuing with our grocery item and grocery list themes, you are now at a grocery store shopping for the grocery items on your list. As you walk up and down the aisles you place grocery items into your shopping trolley, one grocery item on top of the other. The last grocery item you place into your shopping trolley will be on top and will be the first grocery item you remove. In fact, if you want to get to something at the bottom of your trolley you'll have to remove everything on top of it 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 trolley when the wheel falls off rendering your trolley unmovable. Determined to complete your grocery shopping you grab another trolley and begin moving items from the broken trolley to the new trolley 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 trolley and carefully move grocery items between the two new trolleys so that the breakable items are always on the top.

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

```
START
Procedure carefully_move_grocery_items( number_of_items_to_be_moved, broken_trolley, working_trolley, spare_trolley )

   IF number_of_items_to_be_moved == 1, THEN
      move top item from broken_trolley to working_trolley

   ELSE
      carefully_move_grocery_items( number_of_items_to_be_moved-1, broken_trolley, spare_trolley, working_trolley )
      move top item from broken_trolley to working_trolley
      carefully_move_grocery_items( number_of_items_to_be_moved-1, spare_trolley, working_trolley, broken_trolley )

   END IF

END Procedure
STOP
```

See Towers of Hanoi in WikiBooks, or Data Structure & Algorithms - Tower of Hanoi, Tower of Hanoi video or Tower of Hanoi recursion game algorithm explained for more about the recursive algorithm.

A sample trace might look like:

```
After   0 moves:     Broken Trolley           Spare Trolley            Working Trolley
                     ------------------------------------------------------------------------
                     eggs
                     bread
                     apple pie
                     hotdogs
                     rice krispies
                     milk
                     ========================================================================

After   1 moves:     Broken Trolley           Spare Trolley            Working Trolley
                     ------------------------------------------------------------------------
                     bread
                     apple pie
                     hotdogs
                     rice krispies
                     milk                     eggs
                     ========================================================================

After   2 moves:     Broken Trolley           Spare Trolley            Working Trolley
                     ------------------------------------------------------------------------
                     apple pie
                     hotdogs
                     rice krispies
                     milk                     eggs                     bread
                     ========================================================================

After   3 moves:     Broken Trolley           Spare Trolley            Working Trolley
                     ------------------------------------------------------------------------
                     apple pie
                     hotdogs
                     rice krispies                                     eggs
                     milk                                              bread
                     ========================================================================

and so on . . .

After  63 moves:     Broken Trolley           Spare Trolley            Working Trolley
                     ------------------------------------------------------------------------
                                                                       eggs
                                                                       bread
                                                                       apple pie
                                                                       hotdogs
                                                                       rice krispies
                                                                       milk
                     ========================================================================
```

Once you fill your shopping trolley you'll proceed to the checkout line. When it's your turn, you'll take all your grocery items from your shopping trolley and place them flat on the counter one after the other where they will be scanned in order and a total calculated. As a grocery item is scanned, the UPC is used to query the store's persistent database for the grocery item's full name, description, and price. Note that the grocery item scanned may not be in the database. 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:

```
"00688267039317",  "Nature's Promise",  "Nature's Promise Naturals Fresh Brown Eggs Omega 3 Large",  77.47
"00835841005255",  "Fiber One",  "Fiber One Bread Country White",  8.73
"09073649000493" (pumpkin pie) not found, so today is your lucky day - You get it free! Hooray!
"00038000291210",  "Rice Krispies",  "Kellogg's Rice Krispies Cereal",  40.37
"00075457129000",  "Kirkland Foods",  "Kirkland Family Farms Dairy Pure Milk 1½% Lowfat",  30.28
-------------------------
Total  $156.85
```

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. When you're ready, replace the GroceryItem.cpp file packaged with this assignment with your (potentially updated) GroceryItem.cpp file from last assignment.
2. Implement the database functions first. This worldwide database of grocery items has hundreds of thousands of entries most grocery stores access when looking up a particular UPC's for a matching brand name, product description and recommended retail price. Details are in GroceryItemDatabase.cppm and GroceryItemDatabase.cpp.
    a. The constructor opens a text file containing Grocery Items and populates a memory resident data store with the contents of the text file. Implement the memory resident data store with a standard vector. You are given a sample database text file to test your work. The actual database file used to grade your work may be different.
    b. The find() function takes a UPC, searches the memory resident data store, and returns a pointer to the grocery item if found and a null pointer otherwise. Solutions with non-recursive solutions earn no credit. Separate interface from implementation by declaring an overloaded helper function in the module interface and implementing both find functions in the module implementation (source) file.
    c. The size() function takes no arguments and returns the number of grocery items in the database.
3. Next, implement the segments in main.cpp from top to bottom. Details are embedded in main.cpp.
    a. Implement the carefully_move_grocery_items items recursive algorithm first, then
        i. Hint: You may want to stub this out for now so you can make progress elsewhere, but do remember to come back later and actually implement it. In TO-DO (2), simply set `to = from;`
    b. Snag an empty trolley
    c. Shop for a while placing grocery items into a shopping trolley
    d. A wheel on your trolley falls off so carefully move grocery items from the broken trolley to a new trolley that works

e. Checkout and pay for all this stuff by choosing a checkout line and placing grocery items on the counter's conveyor belt. Once all the grocery items have been moved from the trolley to the counter the cashier will begin scanning the grocery items in the order you placed them on the counter.

f. Scan the grocery items in the order you placed them on the counter accumulating the amount due and creating a receipt with full product descriptions and prices obtained from the database.

   i. Don't assume the grocery item's UPC will be in the worldwide database of grocery items.

   ii. Do assume the database text file will have formatting errors and that your Grocery Item's extraction operator will handle the errors. Of course, verify your Grocery Item's extraction operator really does handle errors.

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 12 such sections of code you are being asked to complete. 7 of them are in main.cpp, 2 are in GroceryItemDatabase.cppm, and 3 are in GroceryItemDatabase.cpp.

2. Avoid inserting complex expressions into streams. Instead, use the standard format and standard print functions. Instead of `std::cout << … << … <<` and so on, use `std::print( std::cout, "…", …, …,` and so on.

3. Avoid the *to_string(…)* like functions. Instead, use `std::format("…", …, …, …)`.

4. Avoid `#include` directives. We are using C++ Modules exclusively, which are imported not included. Rare exceptions are used in very specific circumstances and are specifically noted.

Reminders:

- Multiple submissions may be available and a waiting period between submissions may be enforced.
- The C++ using directive `using namespace std;` is **never allowed** in any 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.
- Object Oriented programming suggests that objects know how to read, write, and format themselves. Classes you write shall overload the insertion and extraction operators, and specialize the standard formatter.
- Object Oriented programming suggests that objects know how to compare themselves. Classes you write shall overload the spaceship (<=>) and equality (==) relational operators.
- 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 Ubuntu 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, 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 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.cppm GroceryItem.cpp	1. GroceryItem.cppm 2. GroceryItem.cpp	Replace with your (potentially updated) file from the previous assignment.
main.cpp GroceryItemDatabase.cppm GroceryItemDatabase.cpp	3. main.cpp 4. GroceryItemDatabase.cppm 5. GroceryItemDatabase.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.
sample_output.txt	6. output.txt	Capture your program's output to this text file using command line redirection. See command redirection. Failure to deliver this file indicates you could not get your program to execute. Screenshots or terminal window log files are not permitted.
	Readme.txt	Optional. Use it to communicate your thoughts to the grader.
Grocery_UPC_Database-Small.dat		Text file with a worldwide grocery item database. Do not modify this file. It's big and unchanged, so don't include it in your delivery
quoted_string_patch.inc		You shan't modify this file. GCC requires a patch to use the std::quoted manipulator with istreams. This patch will be added to your delivery and included during the build process.
RegressionTests/ CheckResults.cppm GroceryItemDatabaseTests.cppm GroceryItemTests.cppm private_fragement_patch.inc		When you're far enough along and ready to have your classes tested, then place these files 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.