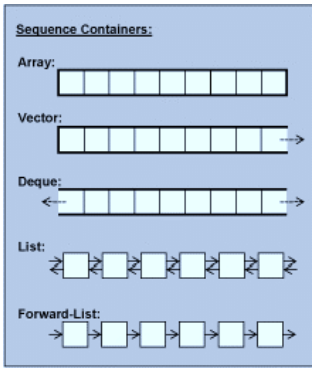
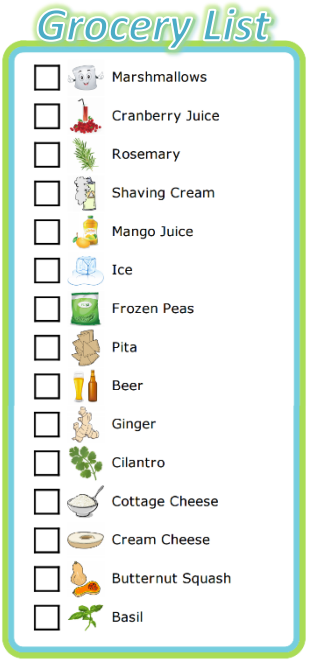
CPSC 131, Data Structures – Spring 2020
Grocery List: Sequence Containers Homework

Learning Goals:



- Familiarization with fixed sized arrays, extendable vectors, (doubly linked) lists, and (singly linked) forward lists insertion, deletion, traversals, and searches.
- Reinforce the similarities and differences between the sequence data structures and their interfaces.
- Analyze and understand the differences in the sequence container's complexity for some of the more common operations
- Familiarization and practice using the STL's sequence container interface
- Reinforce modern C++ object-oriented programming techniques

Description:

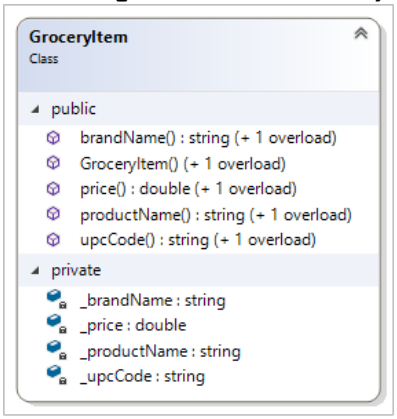


This Grocery List assignment builds on the Grocery Item from the previous assignment. Here you create and maintain a collection of grocery items to form a grocery list. Grocery items are placed on the list, removed from the list, and reordered within the list. A complete `GroceryList` class interface and partial implementation have been provided. You are to complete the implementation.

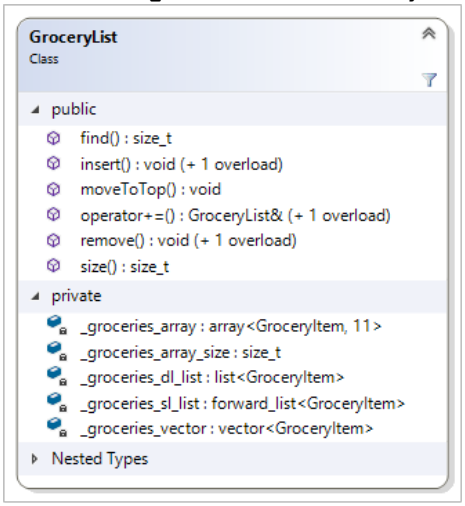
To reinforce the four data structure concepts (fixed sized arrays, extendable vectors, doubly linked lists, and singly linked lists) discussed in class, your grocery list implementation mirrors grocery item insertion, removal, and reordering requests to each of four STL containers (`std::array`, `std::vector`, `std::list`, and `std::forward_list` respectively). At the end of each operation the four STL containers must be consistent. For example, if a request to insert a grocery item is received, then that grocery item is inserted into all four STL containers such that each container holds the same grocery items in the same order.

The following class diagrams should help you visualize the `GroceryList` interface, and to remind you what the `GroceryItem` interface looks like.

`GroceryItem` *class summary*



`GroceryList` *class summary*



Grocery list function summaries:

1. `find()` takes a grocery item as a parameter and returns the zero-based offset of that item, or the total number of items in the grocery list if the grocery item was not found. For example, if your grocery list contains the grocery items in the picture above, a request to find “Marshmallows” returns 0, “Ice” returns 5, and “Milk” returns 15.
2. `insert()` takes a grocery item and either a position (TOP or BOTTOM) or an offset into the list as parameters and inserts the provided grocery item before the insertion point. For example, again if your grocery list contains the grocery items in the picture above, inserting “Milk” with an offset of 5 places “Milk” between “Mango Juice” and “Ice”. Silently discard duplicate items from getting added to the grocery list.
3. `moveToTop()` takes a grocery item as a parameter, locates and removes that grocery item, and then places it at the top of the list. For example, a request to move “Beer” to the top removes “Beer” from its current location and places it before “Marshmallows”. Of course, “Ginger” would then immediately follow “Pita”. The grocery list remains unchanged if the provided grocery item is not in the grocery list.
4. `operator+=` concatenates the grocery list provided as a parameter to this grocery list. Silently discard duplicate items during concatenation.
5. `remove()` takes either a grocery item or a zero-based offset from the top as a parameter and removes that item from the grocery list. No change occurs if the given grocery item is not in the grocery list, or the offset is past the size of the grocery list. For example, a request to remove “Beer” from your above pictured grocery list reduces the size of the grocery list by one and causes “Ginger” to immediately follow “Pita”.
6. `size()` takes no parameters and returns the number of grocery items in the grocery list. For example, the size of your above pictured grocery list is 15. The size of an empty grocery list is zero.

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.

   GroceryItem's constructor's parameter order is important for this assignment. You must allow grocery items to be constructed with at least zero, one, or two arguments. If one argument is given, it must be the product name. If two arguments are given, the first one must be the product name, and the second one must be the brand name. Take a close look at last assignment's posted solution and double check your grocery items constructor's parameter order.

   When you're ready, replace the GroceryItem.hpp and GroceryItem.cpp file stubs packaged with this assignment with your updated GroceryItem.hpp and GroceryItem.cpp files from last assignment.
2. Compile your program using Build.sh. There will likely be warnings, after all it's only partial solution at this point. If there are errors, solve those first. For example, implement groceries_sl_list_size() first to remove the "must return a value" error. Your program should now execute.
3. Once you have an executable program, start implementing functions in the order called from function main(). main() drives the testing, so working the functions in called order allows you to see your progress. That is, implement the functions in this order:
   a. moveToTop()
   b. find()
   c. remove()
   d. insert()
   e. operator+=()
4. Implementing insert() and remove() is really implementing insert and remove on each of the four STL containers. You may want to:
   a. Work the insert() and remove() functions together for arrays, then for vectors, lists, and finally forward_lists. Insertion and removal (or as the STL calls it, erasure) are very close complements of each other.
   b. While working insert and remove for each container, you may want to temporary turn off container consistency checking by commenting out those functions. But don't forget to uncomment them before you're finished.

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 14 such sections of code you are being asked to complete. All of them are in GroceryList.cpp.

Hint: In many cases, the requested implementation requires only a single line of code. Of course, finding that single line is non-trivial. Most all can be implemented with less than 5 or 6 lines of code. If you are writing significantly more than that, you may have gone astray.

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.
- Object Oriented programming suggests that objects know how to read and write themselves. Classes you write should overload the insertion and extraction operators.
- Object Oriented programming suggests that objects know how to compare themselves. Classes you write should overload the equality and inequality operators.
- Always initialize your class's class (global) and instance attributes. Instance attributes should always be initialized with member initialization or within the constructor's initialization list. Avoid assigning initial values within the body of constructors.
- 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
main.cpp GroceryList.hpp	1. main.cpp 2. GroceryList.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.
GroceryItem.hpp GroceryItem.cpp	3. GroceryItem.hpp 4. GroceryItem.cpp	You should replace the provided file stubs with your (potentially) updated files from the previous assignment.
GroceryList.cpp	5. GroceryList.cpp	Start with the file provided, make your changes in the designated TO-DO sections (only), and delivery your final solution.
	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.
GroceryListTests.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.