Project 3: Container Adapters

Learning Goals:

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

Description:

Continuing with our Book and BookList themes, you are now at a bookstore shopping for the books on your list. As you walk up and down the aisles you place books onto your book cart, one book on top of the other. The last book you place onto your book cart will be on top and will be the first book you remove. In fact, if you want to get to something at the bottom of your cart you'll have to remove everything on top of it first. You're a very smart shopper so you know to put the big heavy books on the bottom, and the small soft breakable popup baby books on the top.

As luck would have it, you've almost completed your shopping and have a pretty full cart when the wheel falls off, rendering your cart unmovable. Determined to complete your book shopping, you grab another cart and begin moving books from the broken cart to the new cart when you realize that your soft baby books, like Ready for Life's "Just Like the Animals" 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 carefully move books between the two new carts so that the soft baby books are always on top.

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

```
START
Procedure carefully_move_books (number_of_books_to_be_moved, broken_cart, working_cart,
spare_cart)

   IF number_of_books_to_be_moved == 1, THEN
      move top book from broken_cart to working_cart
      trace the move

   ELSE
      carefully_move_books (number_of_books_to_be_moved-1, broken_cart, spare_cart, working_cart)
      move top book from broken_cart to working_cart
      trace the move
      carefully_move_books (number_of_books_to_be_moved-1, spare_cart, working_cart, broken_cart)

   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.

Once you fill your working book cart, you'll proceed to the checkout line. When it's your turn, you'll take all your books from your book cart and place them flat on the counter one after the other where they will be scanned in order and a total calculated. As a book is scanned, the ISBN is used to query the store's persistent database for the book's full title, author, and price. Note that the book scanned may not be in the database. You take your receipt and your bags of books and leave the store.

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

```
"9780895656926", "Just Like the Animals", "Ready for Life", 41.97
"54782169785" (131 Answer Key) not found, book is free!
"0140444300", "Les Misérables (1st edition)", "Victor Hugo", 28.96
"9780399576775", "Eat pray love made me do it (1st edition)", "Rebecca Asher", 36.99
"9780545310581", "The Hunger Games - Library Edition", "Suzanne Collins", 67.56
-----------------------
Total $175.48
```

How to Proceed:

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

1. Your Book class from Project 1 needs to be working well before continuing with this assignment. If your Book class failed any unit tests, review the solution walkthrough video and fix your Book code.
2. Implement the database functions first. Details are in BookDatabase.hpp and BookDatabase.cpp.
    a. The constructor opens a text file containing Books 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 an ISBN, searches the memory resident data store, and returns a pointer to the book if found and a null pointer otherwise. Implement this function recursively.
c. The size() function takes no arguments and returns the number of books in the database.

3. Implement the segments in Main.cpp from top to bottom next. Details are embedded there.
   a. Implement the carefully_move_books recursive algorithm. Hint: If you want to stub this out for now so you can make progress elsewhere, simply set `to = from;` in TO-DO(1). Do remember to come back later and actually implement it.
   b. Shop for a while placing books onto a book cart.
   c. A wheel on your cart falls off so carefully move books from the broken cart to a new cart that works.
   d. Checkout and pay for all this stuff by placing your books on the checkout counter. These will be scanned by the cashier in the order the books were placed on the checkout counter.
   e. Scan the books, accumulating the amount due and creating a receipt with full product descriptions and prices obtained from the database. Don't assume the book's ISBN will be in the store's persistent database.

Rules and Constraints:

1. You are to modify only designated TO-DO sections. Designated TO-DO sections are identified with the following comments:

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

   Before submitting the assignment, remove all these TO-DO statements from BookDatabase.hpp, BookDatabase.cpp, and Checkout.cpp. Insert your code between them. In this assignment, there are 12 such sections of code you are being asked to complete: two are in BookDatabase.hpp, three are in BookDatabase.cpp, and seven of them are in main.cpp.

Team Formation:

As stated in the syllabus, you may work in a team of 1-2 students. Your first task is to decide on who will be on your team before you do any of the work described below.

Obtaining and Pushing Code:

We are using:

- GitHub Education to distribute starter code. You save your work by pushing to a github repository. The integrity of files permitting, github is used to evaluate your code using Github Actions. You are free to run Github Actions on your own, but use that far exceeds 10 github actions per month will elicit a discussion and possible deduction.
- Tuffix Linux environment (recommended) to locally run and create your submissions. This is the environment used by Github Actions, and the environment I will use in evaluating any work locally.

This document explains how to obtain starter code and push it to GitHub: GitHub Education / Tuffix Instructions.

This Youtube video by David McLaren explains how to get started with GitHub: https://youtu.be/1a5L_xsGIm8.

Code Layout:

You will need to work with the following files.

- `README.md`: You must edit this file to include your name and CSUF email. This information will be used in the grading process.
- `Book.hpp`: This file declares the Book class, and is complete. Do not modify this file.
- Book.cpp: Definitions for all the Book member functions. Replace with your finished/reviewed work from Project 1 and/or Project 2. Ensure that you **upload your version of Book.cpp to your repository**.
- BookDatabase.hpp: This file declares the BookDatabase interface. Modify this file, completing the TO-DOs in it.
- BookDatabase.cpp: This file defines the BookDatabase implementation. Modify this file, completing the TO-DOs in it.
- Main.cpp: This file performs a sequence of defined operations. Modify this file, completing the TO-DOs in it.
- Local_build.sh is a script that automates updating, compiling, running, and testing the project.

The repository also contains other files, which you must leave unchanged, and may ignore (though you're welcome to look inside if you wish).

- BookDatabase_test.cpp - This file defines the unit tests for the BookDatabase class.
- CMakeLists.txt - used by cmake to automate the build process in a way that can be easily accessed using our repository on a fresh installation of Ubuntu
- c-cpp.yml - this file defines the github actions that are available to this project. Actions emulated by local_build.sh

Locally Building Targets:

You can use the included “local_build.sh” file to run a series of console commands (in a linux environment) that will create and run your code.

To enable the use of the file on your machine, we have to enable **execution** (potentially dangerous with unknown files!):

```
$ chmod +x local_build.sh
```

To run the local build process once permission is granted:

```
$ ./local_build.sh
```

We recommend that you use these make commands inside Tuffix to confirm that your code works, and preview how your submission will be graded, before accepting your work as completed, or checking it in the way it will be evaluated using Github Actions.

Reminders:

- The C++ using directive `using namespace std;` is **never allowed** in any header or source file in any deliverable product.

- It is far better to deliver a marginally incomplete product that compiles error and warning free than to deliver a lot of work that does not compile. The autograder rejects submissions that do not compile. A rejected submission receives an “F” score.
- Filenames are case sensitive on Linux operating systems, like Tuffix.