CPSC 131, Data Structures – Spring 2020
Book: Introduction & Review Project

Learning Goals:

- Become familiar with creating, compiling, running, and submitting programming assignments
- Demonstrate mastery of basic C++ skills, including
  - allocating and releasing dynamic memory
  - reading from standard input and writing to standard output
  - working with standard vectors
  - overloading insertion and extraction operators
- Demonstrate the ability to translate requirements into solutions
- Refresh your memory and normalize our point of departure. Depending on your background and how long ago you actively practiced programming in C++, some of this may be review and some may seem new to you

Description:

In this assignment, you will play both the role of class designer and the role of class consumer. As class designer you will implement the provided class interface, and then then as class consumer you will instantiate objects of this class to solve a simple problem. The class itself has a few private attributes, and a multiparameter constructor. Objects of the class have the fundamental capability to insert, extract, and compare themselves. The problem being solved is simply to read several objects storing then dynamically, and after you've read them all print them in reverse order. Specifically, you are to implement:

1. **A class named Book**. You will reuse this class in all future projects, so effort you apply getting it right now will greatly benefit you later.
   a. Attributes
      i. Title – the name of the book (Ex: An Introduction to Programming with C++, Data structures for particle physics experiments)
      ii. Author – the book's author (Ex: Diane Zak, Alison "Ally" Uttley)
      iii. ISBN – a 10 or 13 character international standard book number uniquely identifying this book (Ex: 9790619213090, 979010181X).
      iv. Price – the cost of the book in US Dollars (Ex: 74.99, 115.50). Code this as type `double`.
   b. Construction
      i. Allow books to be constructed with zero, one, two, three, or four arguments[1]. Title must be the first argument, author the second, ISBN the third, and price the fourth.
      ii. Initialize each attribute with member initialization[2] and in the constructor's initialization list[3]. Do not set the attribute's value in the body of the constructor.
   c. Operations
      i. Set and retrieve each of the attributes. Name your overloaded functions isbn, title, author, and price. For example:

```
void        isbn( const std::string & isbn); //mutator
std::string isbn() const;                    //query
```

[1] See Rational.hpp @ lines 91-92 and RationalArray.hpp @ lines 72-74 for constructors with multiple arguments, some with defaulted values
[2] See Rational.hpp @ lines 288-289 and RationalArray.hpp @ lines 211-216 for member initialization examples
[3] See Rational.hxx @ lines 45, 58 and RationalArray.hpp @ lines 58-59 for constructor's initialization list examples

ii. Overload the insertion and extraction operators[4]. For example, main() may read from standard input and write to standard output a Book object like this:

```
Book book;
std::cin  >> book; // extraction (reading)
std::cout << book; // insertion  (writing)
```

Insertion and extraction shall be symmetrical. That is, you shall be able to read what your write. Assume fields are separated by commas and string fields are always enclosed with double quotes. The first field must be the ISBN, the second the title, the third the author, and the fourth the price. For example:

"0001005340", "Little Grey Rabbit's Birthday", "Alison \"Ally\" Uttley", 31.57

Incomplete books or books with errors shall be ignored.

Don't try to add and remove the quotes yourself. See and use std::quoted()[5])

iii. Overload the 6 relational operators[6]. For example, function main() may compare Book objects like this:

```
Book interestingNovel, boringNovel;
if( interestingNovel == boringNovel) ...
if( interestingNovel <  boringNovel) ...
```

Books interestingNovel and boringNovel are equal if all attributes are equal (or within 0.0001 for floating point numbers, like price). Books are to be sorted by ISBN, author, title, then price. For example, if interestingNovel's and boringNovel's ISBN are equal but interestingNovel's author is less than boringNovel's author, then interestingNovel is less than boringNovel.

d. Separate interface from implementation using header and source files. Implement all functions in the source (.cpp) file, not the header (.hpp) file. Header file only solutions will not be accepted.

2. **Function main()** to use the Book class above:
   a. Read a book[7] from standard input (std::cin) until end of file[8]. For each book read:
      i. Store the book in a dynamically allocated object
      ii. Store the pointer to the book in a standard vector
   b. After you have reached the end of file, write the books to standard output (std::cout) in reverse order.
   c. Be sure to release the dynamically allocated objects before exiting the program

[4] See http://www.cplusplus.com/doc/tutorial/templates/. Also see Rational.hpp @ lines 50-51 and Rational.hxx @ lines 155-156 for insertion and extraction overloading examples. Additional examples at operator overloading (scroll down to Stream extraction and insertion)

[5] Additional example at std::quoted and the "friendly" delimiters

[6] Also see Rational.hpp @ lines 48, 361-377 and Rational.hxx @ lines 128-143 for relational operator overloading examples. Additional examples at operator overloading (scroll down to Relational operators)

[7] Do not read a book's attributes (three strings and a number), use Book's extraction operator to read a book. You know you have an incorrect solution if you have defined variables to hold title, author, ISBN, or price.

[8] This program requires you to not explicitly open files. Simply write your program extracting data from std::cin. Enter Cntl-D (Linux) or Cntl-Z (windows) to indicate end-of-file. Better yet, create a text file with your input and then simply redirect input from that text file (see below). You know you have an incorrect solution if you have included <fstream> or call the ifstream::open function.)

Reminders:

- The C++ using directive `using namespace std;` is **never allowed** in any header or source 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.
- Object Oriented programming suggests that objects know how to read and write themselves. Classes you write shall overload the insertion and extraction operators.
- Object Oriented programming suggests that objects know how to compare themselves. Classes you write shall overload the equality and inequality 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 Tuffix 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. Using Build.sh is not required, but strongly encouraged. The grading tools use it, so if you want to know if you compile error and warning free (required to earn any credit) than you too should use it.
- Using std::system("pause") is not permitted. If you don't know what this is, good!
- Don't directly compare floating point numbers for equality. Instead of `x == y`, say `std::abs(x-y) < EPSILON`.
- 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
Book.hpp	1. Book.hpp	You shall not modify this file. 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 this file in your delivery.
	2. Book.cpp 3. main.cpp	Create these files as describe above.
	4. 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.
BookTests.cpp CheckResults.hpp		These files contain code to regression test your Book class. When you're far enough along and ready to have your class regression tested, then place these files somewhere in your working directory and Build.sh will find them. Simply having these files in the directory (or sub directory) will add it to your program and run the tests – you do not need to #include anything or call any functions. These tests will be added to your delivery and executed during the grading process.
sample_input.txt		A sample set of data to get you started.
sample_output.txt		A sample of a working program's output. Your output may vary.