Due Date: Mar 2, 2022 @ 11:59:59

STL Introduction – BookDB

This assignment will focus on using a few features of the C++ Standard Library. You will write a program that uses a vector to construct and manipulate a simple database of books. Each book instance will be represented as a struct, and you will be required to traverse, modify, sort, and delete from the database.

Your will submit your work to Gradescope and it will be both autograded and human graded. The autograder will compile your code with g++-7.5.0.

Be sure to watch Piazza for any clarifications or updates to the assignment.

Don't forget the Assignment 2 Canvas Quiz! Your assignment submission is not complete without the quiz.

BookDB Overview

Your book database must consist of 3 files: `BookDB.h`, `BookDB.cpp`, and `demo.cpp`. The first being the header file, the second being the implementation of the database modification functions, and the third being the `main()` function implementation.

For `BookDB.h`:

- Be sure to have include guards.
- Don't import anything that is unnecessary.
- Include file header comments that have your name, student id, a brief description of the library, and be sure to cite any resource (site or person) that you used in implementing the library.
- Each function should have appropriate function header comments.
  - You can use javadoc style (see doxygen) or another style, but be consistent.
  - Be complete: good description of the function, parameters, and return values.
- As in assignment 1, no classes and no inline methods.
- In `BookDB.h`, you need to define a `struct` called `Book`. It has the following members:
  - `unsigned id`: Book ID
  - `unsigned year`: Publication year
  - `double rating`: Book rating

For `BookDB.cpp`:

- All the function bodies for those defined in `BookDB.h` should be here.
- Don't clutter your code with useless inline comments (it is a code smell [Inline Comments in the Code is a Smell, but Document the Why], unless it is the why).
- Follow normal programming practices for spacing, naming, etc: Be reasonable and consistent.
- Be sure to avoid redundant code.

Functions to implement:

`int addBook(unsigned bookID, unsigned year, double rating, std::vector<Book> &db)` First search the database for an entry with the given bookID. If no such entry is found, add a book with the given bookID, year, and rating to the end of the vector and return 1. If there is already an entry with the given bookID, the list should not be changed and 0 should be returned.

`int updateBook(unsigned bookID, unsigned year, double rating, std::vector<Book> &db)` Search the database for an entry with the given bookID. If such an entry is found, update the rating and the year as per the input and return 1. If no such entry is found, return 0.

`int deleteBook(unsigned bookID, std::vector<Book> &db)` Search the database for an entry with the given bookID. If such an entry is found, remove the entry from the database and return 1. If no such entry is found, the list should not be changed and 0 should be returned.

`std::vector<Book>* findBooks(unsigned year, const std::vector<Book> &db)` Returns a pointer to the vector of Books published in the year specified by the argument. Don't forget to allocate and free heap memory for the vector!

`double calculateAverageRating(const std::vector<Book> &db)` Calculate the average rating of all the books in the database. If no books have been added, return -1.0.

`void print(const std::vector<Book> &db)` Prints the contents of the database to `cout`, starting from the beginning entry, print out each book on its own line. For each book, the output should be the book's id, the year, and the rating, separated by a comma and space. The rating should be followed immediately by a newline. For example:

```
12345, 2014, 4.5
```

If there are no entries in the database, output `"No entries\n"`.

`int sortDB(std::vector<Book> &db, short sortingMethod)` This functions should take two arguments: the book database and the user-specified sorting behaviour choice. The three sorting options are:

1. Sort the books in order of increasing ID number.
2. Sort the books in order of increasing year. Break ties by putting smaller book IDs before larger ones.
3. Sort the books in order of increasing rating. Break ties by putting smaller book IDs before larger ones.

If the user enters something other than the three possible numbers for the sort options, return 0. Otherwise return 1.

demo.cpp

For assignment 2, on Canvas, you will find a nearly complete `demo.cpp`. You'll need to add the file header, comments and fix a memory leak.

Makefile

Your submission must include a `Makefile`. The minimal requirements are:

- Compile using the demo executable with the `make` command.
- Use appropriate variables for the compiler and compiler options.
- It must contain a clean rule.
- The demo executable must be named something meaningful (not a.out).

README.md

Your submission must include a `README.md` file (see https://guides.github.com/features/mastering-markdown/). The file should give a brief description of the program, the organization of the code, how to compile the code, and how to run the program.

Submission

You will upload all the required files to Gradescope. The detail on how to do that will become available to you once you have completed the Assignment 1 Canvas quiz.

There are no limits on the number of submissions. See the syllabus for the details about late submissions.

Grading

For this assignment, half of the marks will come from the autograder. For this assignment, none of the test details have been hidden, but there may be hidden tests in future assignments.

The other half of the marks will come from human-grading of the submission. Your files will be checked for style and to ensure that they meet the requirements described above. In addition, all submitted files should contain a header (comments or otherwise) that, at a minimum, give a brief description of the file, your name, and wisc id.

Rubric

- Autograder Tests - 42
- README File - 5
  - Name / student id
  - Typeset/organized with markdown
  - Program description
  - Compilation instructions
  - Run instructions
  - Code organization description
- Makefile - 5
  - File header comments including name/id
  - Demo build rule with appropriate named executable

  - BookDB.o build rule
  - Clean build rule
  - Correct file name

- demo.cpp - 5
  - File header comments including name/id
  - Fix memory leak

- BookDB.h - 10
  - File header comments including name/id
  - Include guards
  - All includes are required for header file
  - Function header comments with function description, return value and parameters
  - Comment detailing other elements like structs
  - Function prototypes include default values as per the specification
  - Only inline function bodies are defined in header

- BookDB.cpp - 13
  - File header comments including name/id
  - No unused/unneeded includes
  - Consistent style (braces, spacing, etc)
  - Avoid inline comments
  - Avoid duplicate code, use helper functions if needed
  - Avoid c-style casting
  - Avoid gotos
  - Well-designed functions

Tackling Assignment 2

1. Create a directory for your assignment 2 project, and create the required files: `BookDB.cpp`, `BookDB.h`, `README.md`, and `Makefile`. Download the `demo.cpp`, and the sample output file from Canvas.

2. Get things compiling: Put the function prototypes in `BookDB.h`, the function stub details in `BookDB.cpp`, and set up your makefile. (Pro tip: When adding you function prototypes, also do the function header comment).

3. Finish `demo.cpp`: Add the file header comments, function comments, and fix the memory leak.

4. Implement and test the `AddBook` and `Print` functions. Do they pass the autograder tests?

5. Implement and test the `Update`, `Delete`, and `CalculatedAvg` functions. Do they pass the autograder tests?

6. Implement and test `FindBooks`. Does it pass the autograder?

7. Implement and test `SortDB`. Does it pass the autograder?
8. Don't forget to do the `README.md` file.
9. Final submission: make sure you have file comments, and name on all files. Double check the rubric, and do your final submission to Gradescope.

HAPPY CODING!