CPSC 350: Data Structures
Assignment 5, Version 1.2
Building a Database with Binary Search Trees
Due: 5-11-2020, 11:59pm.

Overview

In this assignment, you will push your C++ skills to the limit by implementing a simple database system using binary search trees. Though the end product will be a far cry from Oracle or MySQL, your DB will allow the user to insert, delete, and query data. The data itself will be persistent (stored on disk), so that you may process it over several sessions.

The DB itself will contain data that would be commonly found in a university's computer system. In our case, this information consists of student and faculty records. The information for each will be stored in its own tree (or "table" in DB terminology).

Though I will provide you with a general outline of the program, many of the implementation details will be up to you. In the same spirit, I will give you a point in the right direction as far as some of the C++ techniques go, but it will also be your responsibility to research the techniques in more detail.

Details

Tables

The tables that store the records in your DB will be binary search trees. The nodes will consist of Student or Faculty objects, depending on the tree. The tree will be sorted on the primary key value of the nodes, which in our case will be faculty and student Ids.

Your first job will be to build a BST implementation supporting the usual operations (including delete). This should not be difficult in and of itself. Just be sure to use templates to make your implementation generic, and overload operators as required.

Student Records

Student records will be stored in a Student class. Student records contain a unique student ID (an integer), a String name field, a string level field (Freshman, Sophomore, etc), a String major field, a double GPA field, and an integer advisor field, which will contain the Faculty ID of their advisor. These are the only fields the class contains.

The Student class must overload equality, less than, greater than operators, etc. so that we can compare them to one another.

Faculty Records

Faculty records are similar to student records, and will also require overloaded operators.

Faculty records contain an integer Faculty ID, a String name, a String level (lecturer, assistant prof, associate prof, etc), a String department, and a list of integers corresponding to all of the faculty member's advisees' ids. These are the only fields the class contains.

How the Program Should Work

Your program will keep references to both the faculty and student tables in memory. These references are simply BSTree instances. For convenience, we will call them masterFaculty and masterStudent.

When the program starts, it should check the current directory for the existence of 2 files "facultyTable" and "studentTable". These files correspond to the BSTrees containing the faculty and student data. If neither of these files exist, then masterFaculty and masterStudent should be initialized as new, empty trees. If the files do exist, then they should be read into the appropriate variables. (See appendix A)

Once the tables have been set up, a menu should be presented to the user to allow them to manipulate the databases. At a minimum (if you do more you'll get more credit), the choices should include:

1. Print all students and their information (sorted by ascending id #)
2. Print all faculty and their information (sorted by ascending id #)
3. Find and display student information given the students id
4. Find and display faculty information given the faculty id
5. Given a student's id, print the name and info of their faculty advisor
6. Given a faculty id, print ALL the names and info of his/her advisees.
7. Add a new student
8. Delete a student given the id
9. Add a new faculty member
10. Delete a faculty member given the id.
11. Change a student's advisor given the student id and the new faculty id.
12. Remove an advisee from a faculty member given the ids
13. Rollback
14. Exit

When a command is selected, you should prompt the user for the required data, and execute the command. If there are any errors, you should inform the user and abort the command.

All of the above commands should enforce referential integrity. That is to say, a student can not have an advisor that is not in the faculty table. A faculty member can't have an advisee in the student table. If a faculty member is deleted, then their advisees must have their advisors changed, etc. Your commands will be responsible for maintaining referential integrity. If a user issues a command that would break referential integrity, you should warn them and abort the command, or execute the command and fix any violations as appropriate.

After each command is executed, the menu should be displayed again, and the user allowed to continue.

The Rollback command is used if the user realizes they have made a mistake in their data processing. The Rollback command will "undo" the previous action, but only if that action changed the structure of the DB. Your program should allow the user to roll back the last 5 commands that CHANGED the DB. (Commands that simply display data do not count.) This will involve keeping snapshots of the DB before and after commands are issued. The implementation details for this are left up to you.

If the user chooses to exit, you should write the faculty and student tables back out to the "facultyTable" and "studentTable" files (see appendix A), clean up, and quit gracefully.

Programming Strategy
At this point you should realize this is a non-trivial assignment. To successfully complete it, you need to make use of your best OO design and programming skills. Think modularly, sketch out a solution before you start coding, and START EARLY. Sleeping is optional, but not recommended.

Requirements
-You may work in groups of 2 on this assignment. (HIGHLY Recommended)
-All code must be your own
-Develop on any platform you wish, but make sure it compiles/runs with g++ within your Linux Docker container

Grading
As usual, you will be graded on correctness, elegance of solution, and your adherence to the above requirements. Style and comments are also important, so be aware that a well-documented, clean solution will receive more credit than a sloppy solution without comments.

Due Date
Submit the link to your GitHubs repository via the course website on Canvas by 11:59 pm on 5-11-2020. The README should contain your names, student Ids, and any comments you have to make about your solution (special instructions for running, etc).

Appendix A: Object Serialization

Serialization involves converting objects into a form such that they can be written out to disk. There are two options for doing this with your binary search trees.

1) Design a standard format for each node in the tree. BSTs are then saved by writing out these nodes to a text or binary file. The BSTs can be restored by starting with an empty tree and inserting nodes corresponding to the nodes from the file.
2) Design a format so the entire tree can be read from/written to a file. This is done most efficiently with binary files.