Lab 10: Classes and Data Abstraction

Overview

This lab will give you practice designing and implementing a simple class. It is similar to Lab 9, except that I have written most of a main program that you need to solve this problem. I include the same review of data abstraction that was contained in Lab 9's description.

Review of Data Abstraction and Abstract Data Types

Procedural abstraction is the separation of what a function does from how it does it. The idea is to write descriptions of what functions do without actually writing the functions, and separate the *what* from the *how*. The client software, i.e., the software calling the function, only needs to know the parameters to pass to it, the return value, and what the function does; it should not know how the function works. In this sense, functions become like black boxes that perform tasks.

Data abstraction separates what can be done to data from how it is actually done. It focuses on the operations of data, not on the implementation of the operations. For example, in data abstraction, you might specify that a set of numbers provides functions to find the largest or smallest values, or the average value, but never to display all of the data in the set. It might also provide a function that answers yes or no to queries such as, "is this number present in the set?" In data abstraction, data and the operations that act on it form an entity, an object, inseparable from each other, and the implementation of these operations is hidden from the client.

Information hiding takes data abstraction one step further. Not only are the implementations of the operations hidden within the module, but the data itself can be hidden. The client software does not know the form of the data inside the black box, so clients cannot tamper with the hidden data.

An **abstract data type** is a representation of an object. It is a collection of data and a set of operations that act on the data. An ADT's operations can be used without knowing their implementations or how data is stored, as long as the interface to the ADT is precisely specified.

There are two views of an abstract data type: its **public view**, called its **interface**, and its **private view**, called its **implementation**. The parts of a class that are in its public view are said to be *exposed by the class*.

The `student` class

The `student` class represents a student record in a single college class. It contains the student's last name, first name, email address, a set of five homework grades, the number of graded homeworks, a set of three exam scores, and the number of graded exams. The homework and exam scores are whole numbers. If the number of graded homeworks is N < 5, then the first N homeworks are the ones with grades, and the same holds true for exams.

The member functions that the `student` class should make available to client software (i.e., should expose) are:

1. A function which, when given a first name, last name, and email address, initializes the student record with those values, setting all homework and exam scores to zero.
2. A function which returns the student's first name.
3. A function which returns the student's last name.
4. A function which returns the student's email address.
5. A function which returns the list of graded homework scores.
6. A function which returns the list of graded exam scores.

7. A function which returns the student's current composite grade, which is 0.6 times the average of all graded exams plus 0.4 times the average of all graded homeworks

8. A function to add a new homework grade to the student's record, provided the total number of homework grades is no more than 5. If the total is already 5, it should return an error value.

9. A function to add a new exam grade to the student's record, provided the total number of exam grades is no more than 3. If the total is already 3, it should return an error value.

10. A function to output to a given stream, the student data in the following format:

    ```
    lastname firstname email N hwkscore1 ...hwkscoreN M examscore1 ... examscoreM
    ```

    where `N` is the number of graded homeworks and `M` is the number of graded exams, and `hwkscore1`, etc are the scores on the homeworks and `examscore1`, etc are the exam scores.

Exercise

You will write a program that implements the `student` class described above. Specifically, you will design the `student` class header and implementation files, named `student.h` and `student.cpp` respectively. Each member function must be thoroughly documented. in the header file.

I will supply most of a main program. This program will prompt the user to enter the name of an input file, and then read input from that file. The file will consist of lines that represent student records, and these will be stored in an array of such records by the main program. The program will display a menu-driven interface that will allow the user to interactively add new grades to a given student's record, display a student's grades and/or grade average, and display the set of all student records. The main program will be missing various pieces, which have comments as placeholders. You will have to fill in the missing pieces.

The

What to Submit

Submit your program, however complete it is, by the end of today's lab, i.e., before the end of the class at 2:00 P.M. ***There is no grace period for this. Programs submitted after 2:00 PM will not be accepted.*** The instructions for submitting are:

1. Create a directory in
   `/data/biocs/b/student.accounts/cs135_sw/cs136labs/lab10/submissions`
   whose name is your *username*. For example, I would create the directory `sweiss`.

2. Copy your three files, which should be named `student.h`, `student.cpp`, and `lab10_main.cpp`, to that directory. You will lose 5% of the grade if you misname the files! Make sure that each file has a preamble with your name and other appropriate information in it.

3. Change the permission on the directory that you created so that no one else can read or modify it.

Do not submit executable files. Remember to document your code (both preamble and comments in the code) and make it easy to read. Your work will be graded based on the rubric outlined in the Programming Rules document.