CS368 Programming Assignment 1

Section 1, Fall 2015

Due by 11:59 pm on Friday, September 25

In this page: Announcements | Overview | Specifications | Handing in   Related pages: Assignments

Announcements

Check here periodically.

9/11/2015

- Program released. To ask questions about the homework and see questions posed by other students and their answers, go to: http://www.piazza.com/wisc/fall2015/cs3681 and sign-in using your wisc.edu account.
- Here are some problems commonly seen in the first programming assignment that we want to warn you about:
  - handing in extra files or files with the wrong names and/or capitalization
  - (for pair programmers) both partners turning in source code files or one or both partners forgetting to turn in the `README.txt` file
  - not following the commenting and style guidelines
  - not following the program output format exactly as specified,
  - having additional output other than what is specified (especially debug related)
  - having poorly-written code, even if it works (e.g., redundancy, convoluted functions, poor modularity)

Overview

Why are we doing this program?

Description

In this assignment you will be using an array to implement an unordered list of structures. The list will have some very basic functionality (add, delete, update, print).

Goals

The goals of this assignment are to gain experience:

- compiling and running C++ programs in Linux
- creating and manipulating an array of fixed size
- defining and using `structs`
- doing console (terminal) input and output
- writing C++ free functions

Specifications

What are the program requirements?

Start with the code given in the file studentDB.cpp. You can find this file at

```
~cs368-1/public/html/assignments/p1/files/studentDB.cpp
```

Note you can copy the file using the Linux `cp` command.

The Database

Implement a basic database containing student records. The database will be an unordered list implemented as a fixed-size array of 7000 elements. Each element of the array will be a structure (`struct`) containing the following student information:

- student ID - a 6-digit integer (e.g., 123456); a valid student ID must be a 6-digit positive integer, starting with any digit other than zero
- number of credits - a non-negative integer value
- overall GPA - a floating point number (use a `double`) between 0.0 and 4.0 (inclusive)

Note: you must use an array to implement your database (e.g., you may not use a vector).

User Interface

The program will use console input and output (`cin` and `cout`). Your `main` function will read user commands from the console (`cin`) and process the student database accordingly. Each command will be on a separate line. The commands are:

`a ID credits GPA`

Adds a student with the given student ID (`ID`), number of credits (`credits`), and overall GPA (`GPA`) to the database. If the student is already in the database, an error message should be printed indicating this.

Example: `a 987654 42 3.24`

`d ID`

Deletes the student with the given student ID (`ID`) from the database. If the student is not in the database, an error message should be printed indicating this.

Example: `d 456789`

`u ID grade N`

Updates the student with the given student ID (`ID`) so that both the total number of credits and the overall GPA reflect the addition of a letter grade of `grade` in a course with `N` credits. In this college, the only letter grades allowed are A, B, C, D, and F (so `grade` will be a single character). Your program should accept both upper-case and lower-case versions of allowed grades (e.g., `b` and `B` are both valid grades).

After this command is processed, the program should output the updated student information to the console (`cout`). If the student is not in the database, an error message should be printed indicating this.

Example: `u 987654 B 3`

`c GPA`

Prints all the students who have a GPA greater than or equal to the given `GPA` in the same format as the `p` command (described below).

Example: `c 3.15`

`p`

Prints the current contents of the student database. The format is one student per line and each line should contain the student ID, number of credits, and GPA separated by commas.

q

Quits the program.

Sample Execution

A brief sample execution of the program is given below (both user input and the resultant output are shown):

```
Enter your commands at the > prompt:
> a 123456 4 3.0
> d 987654
error - student 987654 not found
> c 3.15
no students found
> a 987654 20 3.68
> p
123456,4,3
987654,20,3.68
> c 3.15
987654,20,3.68
> u 123456 A 3
123456,7,3.42857
> d 987654
> p
123456,7,3.42857
> q
quit
```

Required Functions

You must implement the following functions:

addStudent(student ID, number of credits, GPA)

Search the list for an entry with the given student ID. If no such entry is found, add the student with the given student ID, number of credits, and GPA to the end of the list. If there is already an entry on the list with the given student ID, the list should not be changed.

deleteStudent(student ID)

Search the list for the entry with the given student ID. If such an entry is found, remove the entry from the list. If there is no entry on the list with the given student ID, the list should not be changed. Make sure you don't leave any "gaps" in the array when you remove an entry; an easy way to fill a gap is to move the entry at the end of the list into the gap.

updateStudent(student ID, letter grade, course credits)

Search the list for the entry with the given student ID. If such an entry is found, update the total credits and overall GPA to take into account the addition of the given letter grade (a single character) in a course with the given number of credits.

count(GPA)

Search the list for all the students having a GPA greater than or equal to the given GPA and print them out in the format specified in the description of the `p` command above. If there are no students who meet that criterion, print `no students found`

print

Prints the list in the format specified in the description of the `p` command above.

Important note: The parameters listed in the functions above are **not** exhaustive. You can (and should) pass more information to the functions than what is listed. You also can (and will probably want to) create more functions than the ones listed.

Error Checking

Your program should give reasonably informative error messages (for example, "`error - student 987654 not found`" is reasonable, whereas simply "`error`" is not). Some situations for which you will need to give error messages have been mentioned above.

You can assume that all commands will have the correct format, that is, that any command we use to test your program will start with one of the acceptable characters and that commands expecting additional information will have them and they will be of the correct type. **You will, however, need to check that values are in the appropriate range.** For example, your program does not need to be able to handle the command "`u 987654 AB 3.5`" but it should handle "`a 123 20 2.3`" and "`u 123456 I 4`". Your program should handle these error situations by printing out a reasonable error message and going on to the next command.

Handing in

What should be handed in?

Make sure your code follows the style and commenting standards used in CS 302 and CS 367. Note: the commenting standards use javadoc comments for class, method, and constructor headers. You do not need to use javadoc comments in your programs for CS 368; however, your comments should include the same information as the javadoc comments. For example, your function header comments should include a description of what the function does, the name and a short description of each parameter, and a description of the return value.

Electronically submit the following file to your **`In`** "handin" directory by the due date and time (or refer to the late policy):

- Your modified version of **`studentDB.cpp`**
  All your code for this program must be in this one file.

- Pair Programming students must also submit a `README` file: All students working in pairs must read the collaboration policy and submit a `README` file. Each student working in a pair must hand-in this file to his/her own hand-in directory. Copy, complete, and hand-in the file that is linked here:
  - `README.txt`

Please turn in only the files named above. Extra files clutter up the "handin" directories.