CSC30500 Project #2
DUE: Monday, November 21, 2022, 11:59 PM

1 Objectives

- Writing a program that connects and interacts with a MySQL database.

2 Problem Statement

This project is the same as project #1, with the exceptions/additions noted in the next subsection.

2.1 New Portions

1. your program should now make no references to any files.
2. Your program should first prompt for several values:
   - the *hostname* of the MySQL server to use.
   - the *port number* of the MySQL server to use. A value of 0 should indicate using the default port.
   - the *username* for the MySQL account to use.
   - the *password* for the MySQL account to use.
   - the *database name* for the MySQL account to use.
3. your program should then use the above values to connect to the associated MySQL server.
4. In addition to project #1's options (which should *all* still be supported), you should also:
   - allow for a `'d'` (for "delete") command, which should just immediately read in a student last name and first name. *All* records relating to this student should be removed from the database. An example of this might be:

     ```
     d Blythe Stephen
     ```

     which should remove all record of `Blythe Stephen` from your database. Make sure that if `Blythe Stephen` was to be subsequently added to your database again, a transcript would be empty for them until classes were once again added as taken.
   - any duplicates found during any add (`'a'`) command should be rejected and a message indicating such should be printed.

2.2 Original Project #1 Functionality – no changes made here, just included here for clarity's sake.

You will be writing a computer program that keeps track of students, courses, grade types, semesters, and the relationship(s) between them. Your program should essentially prompt the user to enter a one letter command, and process the command as follows:

- **a** (for add), which should then read in one of:

- **c**, which should then read a string representing a course prefix, followed by a positive integer representing a course number, followed by a string representing a course title, and finally a positive integer representing the number of credits for a new course. So, the following:

  ```
  a c CSC 30500 Principles_Of_Databeses 3
  ```

  should add the course `CSC 30500` titled `Principles_of_Databases` worth 3 credit hours.
- **g**, which should then read a string representing a grade type and a floating point number representing a point value. So, the following:

  ```
  a g B+ 3.5
  ```

  should add a grade type of B+ with a value of 3.5.
- **m**, which should then read a string representing a semester code as a string, followed by a year number as a positive integer, and finally a string as a description. These items describe a Semester. So, the following:

  ```
  a m Sp89 1989 Spring
  ```

  should add a semester with code Sp89 representing year 1989 with description Spring. You should assume that only `Spring`, `Summer`, or `Fall` are possibilities for the description.
- **s**, which should then also read the next string as a last name, the following string as a first name, and another string as a phone number. These three items together represent a new student. So, the following:

  ```
  a s Blythe Stephen 636-949-4681
  ```

  should add a student named Stephen Blythe with a phone number of 636-949-4681.
- **t**, which should then also read the next string as a last name, the following string as a first name, a string as a course prefix, an integer as a course number, a string as a course grade, and a semester as a string. These items together represent a student having completed a course. So, the following:

  ```
  a t Blythe Stephen CSC 30500 B+ Sp89
  ```

  should add a course taken for Stephen Blythe of CSC 30500, with a grade of B+ in Sp89.

- **l** (that is, the letter 'l' for "list"), which should then read in one of:
  - **c**, which should list *all* of the courses. Each course should have its prefix, number, title and number of credit hours printed.
  - **g**, which should list *all* of the grade types. Each grade type printed out should include its code and point value.
  - **m**, which should list *all* of the semesters. Each semester printed out should include its code, description, and year.
  - **s**, which should list *all* of the students. Each student printed out should include their last name, first name, and phone number.
  - **t**, which should list *all* of the courses that have been taken. For each such course taken, the student last name, the student first name, the semester code, the course prefix, the course number, *the course name*, and the grade code earned should be printed.

- **t** (for "transcript"), which should read in a student last name followed by a student first name. A transcript should then be printed for this student. The transcript should print out the student named, followed by the list of courses the student has taken and their corresponding grades in each course. These courses should be grouped by *ordered* semester taken. At the end of the student's transcript, the total number of credit hours taken should be printed as well as the student's overall GPA.

- **q** (for "quit"), which should stop the running program.

Some notes:

- After processing one command, the program should continue to prompt the user for another command (and process what the user enters). This repeats until the user enters 'q'.
- *Data should be persistent between runs!* That is, if you add a course in one run, it should appear in another run without re-entering it in the subsequent run.

3 What To Hand In

You will be submitting your source code and a `read.me` file via Canvas. The `read.me` file should include information about your project including (but not limited to):

- your name
- the date
- the platform you developed your code on (Windows, Linux, ...)
- any special steps needed to compile your project
- any bugs your program has
- a brief summary of how you approached the problem

You might also want to consider adding things like a "software engineering log" or anything else you utilized in getting the project done.

4 Grading Breakdown

Code Compiles	20%
Following Directions	25%
Correct Execution	45%
Code Formatting/Comments	10%
Early Submission Bonus	*5%*

5 Notes & Warnings

- This is *not* the kind of project you will be able to start the night before it is due - instead, *START NOW!!!*
- Projects may *not* be worked on in groups or be copied from *anyone*. Failure to abide by this policy will result in disciplinary action(s). See the course syllabus for details.
- Have you started working on this project yet? If not, then *START NOW !!!!!*