CSCE 1040 Homework 2

DESCRIPTION: Console-based gradebook program with support for unlimited classes and students, as well as file saving.
AUTHOR: Andrew P. Sansom
EMAIL: andrewsansom@my.unt.edu
VERSION: 1.0.0
VERSION DATE: 20 Feb 2019
COURSE: CSCE 1040

Classes

Class Student

This represent's a single student's record.

Variables

- int ID representing the ID number of the student
- string name representing the name of the student
- int numberOfClasses representing the number of classes in which the student is currently enrolled. Clearly this value should not be negative. This cannot be more than 5.
- int classification representing the student's classification. 0=Freshman, 1=Sophomore, 2=Junior, and 3=Senior.

Functions

- bool addCourse(int courseID); //adds a course to "courses" and updates the numberOfClasses. Returns true if succesful (i.e. the student is not already at maximum enrollment and the class is not already full) but false if not.
- void print(); //Prints all information about the student to the console. Simply prints everything using a formatted cout string.

Class course

This represents the data for a course.

Variables

private:

- int ID representing the ID number of the course
- string name representing the name of the course
- string location representing the location of the course as C-string

Functions

public:

- void print(); //prints the course's information to the console.

Class enrollment

Represents a single student's grade data for a single class.

Variables

private:

- int ID representing the ID number of the enrollment data
- int studentID representing the student's ID
- int courseID representing the course's ID
- int grades[10] an integer array of size ten representing the 10 grades in the course
- int numberOfGrades an integer representing the number of grades in the course.
- float average representing the average of all of the grades
- char letterGrade representing the letter grade (Intervals written in mathematical interval notation [90,100] => 'A'; [80,90) => 'B'; [75,80) => 'C'; [70,75) => 'D'; [0,70) => 'F')

Functions

- bool addGrade(int grade); //adds a grade to the next available spot in the grades array. Also increments numberOfGrades. Returns true if successful, false otherwise
- float calculateAverage(); //returns the average of all of the grades in an enrollment object as a float. Returns -1 if the average is undefined (due to a division by zero error).
- char calculateLetterGrade(); //returns the letter grade of the student as a char.
- void printGrades(); //prints all grades to the console.
  //Special Setter and Getter functions
- int getGradeNumber(int gradeNumber); //Returns the gradeNumber-th element in the grades array.
- void setGradeNumber(int gradeNumber, int grade); //Sets the gradeNumber-th element of the grades array to grade.

Class Courses

This contains the master list of courses and associated functions

Variables

private:
Course[] COURSES dynamic array representing the master list of courses
int currentNumberOfCourses the current number of courses in the system
int courseCapacity the current size of COURSES

Functions

public:

- Courses(); Constructor
- ~Courses(); Deconstructor
- int makeNewCourse(std::string name, std::string location); //creates a new course by calling the Course constructor and adding it to the COURSES array. If COURSES is not large enough, it allocates more space. Returns the ID number if successful, or -1 if not.
- bool isCourseIDValid(int courseID); //Loops over each course instance in COURSES. Returns true if there is a class with that courseID. False otherwise.
- void printCourses(); //Prints a list of classes to the console.
- std::string getNameFromID(int courseID); //returns the course name of the course instance with the same courseID.
- void storeCoursesData(); //Saves all courses data to the file "courses.dat" in the working directory.
- void loadCoursesData(); //Loads all courses data to the file "courses.dat" in the working directory.

Class Students

This contains the master list of students and associated functions

Variables

private:

- student* STUDENTS; //dynamic array representing the master list of students
- int currentNumberOfStudents; //representing the current number of student records
- int studentsCapacity; //representing the current capacity of student records.

Functions

public:

- Students();
- ~Students();
- int makeNewStudent(std::string name, int classification); //creates a new student record by calling the Student Constructor and adds it to the STUDENTS array, allocating memory if necessary. Returns with the id number if succesful, but -1 if not.
- void printStudents(); //prints a list of all students to the console
- void printStudentInfo(int studentID); //prints the student's information to the console.
- bool enrollStudentInCourse(int studentID, int courseID); //enrolls student with studentID in course with courseID. Returns false if student's numberOfCourses is equal to 5. Then calls Courses's addStudentToCourse method. Adds the courseID to the student's enrolledCourses array. Iterates the student's numberOfCourses by 1. Returns true if successful, and false if not.
- int getStudentIdFromName(std::string name); //returns the ID number of the student with a given name.
- bool isStudentInTooManyClasses(int studentID); //returns true if the student is in too many classes already.
- bool isStudentIDValid(int studentID); //Returns true if there is a student with that studentID. False otherwise.
- std::string getNameFromID(int studentID); //returns the student name of the student instance with the same studentID.
- void storeStudentData(); //stores student data to the file "students.dat"
- void loadStudentData(); //loads student data from the file "students.dat"

Class Enrollments

This contains the master list of enrollments and associated functions

Variables

private:
Enrollment[] ENROLLMENTS dynamic array representing the master list of enrollments
int currentNumberOfEnrollments representing the current number of enrollment items
int enrollmentCapacity representing the current capacity of enrollments.

Functions

public:

Memory Allocation

- Enrollments();
- ~Enrollments();
- int addEnrollment(int studentID,int courseID); //generates a new Enrollment instance and adds it to ENROLLMENTS, allocating memory if necessary. Returns the id of the enrollment instance.

Verification functions.

These are used to check if the enrollment item really can be made.

- bool isCourseFull(int courseID); // returns true if the course has 48 students enrolled in it.
- int countClassesStudentIsEnrolledIn(int studentID); //Counts the number of classes a student is enrolled in, based on the number of enrollment items with the proper student ID. Loops over each element in ENROLLMENTS and adds one to a counter each time the studentID matches that enrollment instance's.
- int countStudentsInClass(int courseID); //Counts the number of students in a class, based on the number of enrollment items with the proper class ID. Loops over each element in ENROLLMENTS and adds one to a counter each time the courseID matches that enrollment instance's.

Grade Manipulation functions.

These either manipulate grades or run some statistic on them.

- bool addGrade(int studentID, int classID, int grade); //adds a grade for student with id studentID who is enrolled in class with id classID. Iterates over the ENROLLMENTS array for the object with matching studentID and classID. Calls that Enrollment instances AddGradeForStudent function. Returns true if successful, returns false if not.
- float computeAverageOfStudent(int studentID, int courseID); //Computes the average of a student in a course.
- float calculateAverageOfStudentsInCourse(int courseID); //computes the average of the average of every student in a particular class.

Print functions.

These print some sort of data to the console.

- void printAllStudentsInClass(int courseID, Students *students, Courses *courses); //prints a list of all of the students in the class to the console, including their names. Loops over each enrollment and prints the student's name if they are in that class.
- void printAllGradesOfStudentInCourse(int studentID, int courseID); //Loops over the

ENROLLMENTS array, finds the first enrollment instance that matches the studentID and courseID. Then prints the grades to the console.

File storage functions.

These well… store files.

- void storeEnrollmentsData(); //stores student data to the file "enrollments.dat" in the working directory
- void loadEnrollmentsData(); //loads student data from the file "enrollments.dat" in the working directory

Global Singleton Collections

These are just singleton collections. There's nothing special about them. They just hold data and functions that manipulate said data.

Enrollments Enrollments;
Students Students;
Courses Courses;

Global Functions

- bool storeData(); //converts the gradebook to a file.
- bool loadData(); //converts the gradebook from a file.
- bool enrollStudentInCourse(int studentID, int courseID); //Checks if the student is enrolled in too many classes and if there are too many students in the class already. If these conditions are both alright, it attempts to enroll the student in the course (using Enrollment's enroll function) and then update the numberOfClasses in the student's student instance in the Students collection. Returns false if any of the above checks fail. True otherwise.

Main Function

We will want to make a menu with an infinite loop. It displays the list of options, then expects a user input to the console as a string. Using a chain of if statements, it runs the appropriate function and displays any output (if appropriate) to the console.