Yen Pham

EUD: yhp0005

CSCE1040

Instructor: David Keathly

Homework 2 Design & Algorithm

I. Data Description for each class

1. Class Student: integer student ID, string student name, and string classification (freshman, sophomore, junior, senior)

2. Class Students: integer student count (to count how many students in the student database), int student capacity (to determine the space in the list in add more students), and dynamic array student list with type Student (array is set to be dynamic to change the array size if needed)

3. Class Course: string course ID (must be entered without space), string name (can be entered with spaces), string location (can be entered with space), string meeting time (combination of multiple string to set up the meeting time, can be entered with space)

4. Class Courses: integer course count (to count how many courses in the course list), int course capacity (to decide how many space should an array list can be); dynamic array course list with type Course (set this array as dynamic array so that we can resize the array list as much as we want)

5. Class Enrollment: integer enrollment ID, integer student ID, string student name, string course ID, dynamic array grade as type integer, integer grade count, float student's grade average, char letter grade (decide grade scale A, B, C, D)

6. Class Enrollments: integer enrollment count, integer enrollment capacity (to decide the size of the enrollment array), dynamic array enrollment list of type Enrollment

II. Function Definition

1. Class Student:

   void set_ID(int num): set student ID with the input variable

   int get_ID(): return the student ID

   void set_name(string name): set student name with the input variable

   string get_name(): return student name

   void set_classification(string classification): set student's classification with the input variable

   string get_classification(): return student's classification

2. Class Students:

Students(): initialize all variables in class Students with CHUNKSIZE for student capacity variable, 0 for other integer variable and “none” for string variables; set dynamic array with the beginning size of CHUNKSIZE

void add_student(): add students to the list

void print_student(): print student list

void set_student_count(int num): set student count variable with the input number

int get_student_count(): return student count variable

void set_student_list(Student* student_list): set array student list with the input array

Student* get_student_list(): get student list

3. Class Course

void set_ID(string ID): set course ID

string get_ID(): return course ID

void set_name(string name): set course name

string get_name(): return course name

void set_location(string location): set course location

string get_location(): return course location

void set_meeting_time(string time, string AM_PM, string meeting_date): set meeting time with the input variables

string get_meeting_time(): return meeting time

4. Class Courses

Courses(): initialize course capacity with CHUNKSIZE, all other integer variables in class Courses with 0, set dynamic array course list with size CHUNKSIZE

void add_course():add course

void print_course():print course list

void set_course_count(int num): set number of courses in the course list

int get_course_count(): return number of courses in the course list

void set_course_list(Course* course_list): set course list array with the input array from main

Course* get_course_list(): get course list array

5. Class Enrollment

Enrollment(): initialize all integer variables in class Enrollment with 0

void set_enrollment_ID(int ID): set enrollment ID with the input value

int get_enrollment_ID(): return enrollment ID

void set_student_ID(int ID): set student ID with the input value

int get_student_ID(): return student ID

void set_student_name(string name): set student name with the input value

string get_student_name(): return student name

void set_course_ID(string ID): set course ID with the input value

string get_course_ID(): return course ID

void set_grade(int *student_grade): set grade list with the dynamic array student_grade

int* get_grade(): return grade list

void set_grade_count(int num): set the number of grades in the grade array

int get_grade_count(): return grade count

void set_student_average(float average): set student average with the input value

float get_student_average(): return student average

void set_letter_grade(char letter): set letter grade with the input value

char get_letter_grade(): return letter grade

6. Class Enrollments

Enrollments(): initialize enrollment capacity with CHUNKSIZE, all other integer variables in class Enrollments with 0, set dynamic array enrollment list with size CHUNKSIZE

void set_enrollment_count(int num): set number of enrollments with the input value

int get_enrollment_count(): return number of enrollment

void add_enrollment(Students student, Courses course): add student to a course

void print_enrollment(): print all students in a course

void add_grade(): add grades for a student in a course

void print_grade(): print a list of all grades for a student in a course

void compute_student_average(): compute the average for a student in a course

void compute_course_average(): compute the average for a course

void set_enrollment_list(Enrollment* enrollment_list): set enrollment list with the input array

Enrollment* get_enrollment_list(): return enrollment list

7. Main

int number_of_students(): calculate how many students in the student list in the file students

int number_of_courses(): calculate how many courses in the course list in the file courses

int number_of_enrollment(): calculate how many enrollments in the enrollment list in file enrollments

void store_GradeBook(Students student, Courses course, Enrollments enrollment): store Grade Book

void load_GradeBook(Students* student, Courses* course, Enrollments * enrollment): load Grade Book

III. Algorithm

1.Add a new course

a. If course count is less than course capacity, create a temporary dynamic array with the size of course capacity. Copy all the elements in the course list array into the temporary array, then delete the course list array and set the course list array equal to the temporary array.
b. If course count is more than course capacity, update the course capacity by adding it to the CHUNKSIZE. Do the same steps as 1a
c. Compare the entered course ID to the course ID in the course list. If found, display a message and come back to the main menu. If not, continue to add course name, meeting time, etc. Add course count to 1 after course is added successfully.
d. Sort the course based on course ID

2.Add a new student

a. If student count is less than student capacity, create a temporary dynamic array with the size of student capacity. Copy all the elements in the student list array into the temporary array, then delete the student list array and set the student list array equal to the temporary array.
b. If student count is more than student capacity, update the student capacity by adding it to the CHUNKSIZE. Do the same steps as 2a
c. Compare the entered student ID to the student ID in the student list. If found, display a message and come back to the main menu. If not, continue to add student name and classification. Add student count to 1 after student is added successfully.
d. Sort the student based on student ID

3.Add students to a course (add enrollment)

a. If enrollment count is less than enrollment capacity, create a temporary dynamic array with the size of enrollment capacity. Copy all the elements in the enrollment list array into the temporary array, then delete the enrollment list array and set the enrollment list array equal to the temporary array.

b. If enrollment count is more than enrollment capacity, update the enrollment capacity by adding it to the CHUNKSIZE. Do the same steps as step 3a
c. Compare the entered course ID to the course ID in the course list. If found, continue to add enrollment. If not, display message and go back to the main menu. If student ID was found compare the entered student ID to the one in the student list. If found, display a message and come back to the main menu. If not, check number of courses that student have been taken. Continue to add enrollment ID number if the course count is less than 5. Check if enrollment ID has been used by another other student. Add enrollment count to 1 after enrollment is added successfully.
d. Prompt user to add more student. If yes, check the student count one course have. If that course reaches 48 student, display message and come back to the main menu. If not, continue to add course.
e. Sort the enrollment based on course ID and enrollment ID

4. Add grades for a student in a course

a. Prompt user to enter course ID and student ID. If course ID and student ID matches the ones in the enrollment list, continue to add grades. If not, display message and go back to the main menu.
b. To add grade, if grades have been added to a student (grade count > 0), copy all the element from the grade list to a new array with size 10, then continue to add grades. If grade count = 0; add new grade without copying the list. Whenever grade is added, add grade count to 1
c. Prompt user to continue to add grades or not

5.Print a list of all grades for a student in a course

a. Prompt user to enter student ID and course ID. Compare both of them with the ones in enrollment list. If matches, check if the grade count is larger than 0 or not. If grade count is larger than 0, print the grade. Else, display error message and come back to the main menu

6.Print a list of all student in a course

a. Prompt user to enter course ID. Compare it to the one in the enrollment list. If matches, print the list of student enrolled in a course.

7.Compute the average for a student in a course

a. Prompt user to enter the student ID and course ID. Check if it matches the ones in the enrollment list
b. If matched, add up all grades of that student, then divide them by the grade count to get the average.

8.Print a list of all students

a. Print list of students using for loop

9.Print a list of courses

a. Print list of courses using for loop

10.Compute the average for a course

a. Prompt user to enter the course ID. Check if it matches one in the course list
b. If matched, add all the computed student's average and divide by student number

11.Store Grade book(to a disk file)

a. Store course list, student list and enrollment list to 3 files

12.Load Grade book(from a disk file)

a. Load information from the students, enrollments and courses files