Enrollment System

You are tasked to program a simple enrollment system where students can enroll several courses to be taken on the following term while faculty members can choose a list of offered courses by the system. The system should be able to allow the enrolled courses only when they took the pre-requisite courses of such courses. Also, the system should automatically assign a room and schedule for each course.

Your program should be able to

1. Manually input the information for students, courses, and faculty to an appropriate data structure during run-time; and
2. To read and store all the information in #1 from and to a text~~/binary~~ file with the use of arrays, structures and strings.

There are three (3) users of the system: student, faculty and an academic assistant to appear in the **Main Menu**.

Below are the properties and operations of each user:

Student

- Maximum # of students is 20.
- Enter student ID
- Enroll courses. The system should automatically display the available courses with their schedule, no. of units, room assignment and the assigned faculty member. Students should be able to choose which courses to take. However, the system should be able to check if the student has already taken the pre-requisite subject of a chosen course. If not, the system should not allow the enrollment of such courses.
    - Add, Edit and Delete of Courses must be possible

- Prints list of enrolled courses (see your EAF for the format)
    - Displays on the screen and in a text file the student info and the approved enrolled courses (EAF). When outputted to a text file (EAF file), proper spacing/tabs in displaying student info and enrolled courses must be observed. Output must follow the format of your EAF. Note that this text file is just for the EAF. Another text file must be used to store information for all the students. See format of this file (**students.txt**) below.

      EAF output/text file must include Course code-section- units -day – time – room – faculty member with grand total of #units
      Name, ID No, Program, EAF # (a student is given a number once the enrollment is finished) , current date and time

- Below is the format of the **students.txt** containing all the information of all the students – courses taken and newly approved enrolled courses

```
STUDENT 1
<student 1 name> <space> <IDNo> <newline>
<course 1 taken>
<course 2 taken>
:
EAF
<course code> <space> <section> <space> <units> <space> <day> <space>
<time> <space> <room> <space> <faculty> <newline>
<course code> <space> <section> <space> <units> <space> <day> <space>
<time> <space> <room> <space> <faculty> <newline>

……..
STUDENT 2
<student 2 name> <space> <IDNo> <newline>
<course 1 taken>
<course 2 taken>
:
EAF
<course code> <space> <section> <space> <units> <space> <day> <space>
<time> <space> <room> <space> <faculty> <newline>
. . .
<course code> <space> <section> <space> <units> <space> <day> <space>
<time> <space> <room> <space> <faculty> <newline>
<eof>
```

Faculty

- Maximum no. of faculty is 20.
- Enters faculty information - ID# and deloading units (max of 12 units)
- Selects course load. The system should be able to display all the available courses from which the faculty can select the preferred course load. However, the system should notify the faculty member if he/she is underload (<12 units) or overload (>12 units) based on the deloading units and total number of units for the preferred course load.
    - Add, Edit and Delete of Teaching Course Load must be possible
        - Before deleting the course, your program must check first if there are no students enrolled yet. If there are, deletion should not be allowed.
- Prints faculty load including all the subjects assigned to the faculty and loading status (underload, overload, with deloading units) on screen and into a text file
- Prints list of students per subject on screen and into a text file
- Another text file must be created for faculty members where all faculty information and current courses are stored (**faculty.txt**)

```
FACULTY 1
<faculty name 1><space><IDNo><space><dept><deloading units><newline>
<course code 1> <space> <section> <newline>
. . . .
<course code M> <space> <section> <newline>
<newline>
```

```
 .. ..
FACULTY N
<faculty name N><space><IDNo><space><dept><deloading units><newline>
<course code 1> <space> <section> <newline>
. . . .
<course code M> <space> <section> <newline>
<newline>
<eof>
```

Academic Assistant/Enrollment System

- Enters student information (name, id#, course, courses/subjects taken)
    - Add, Edit and Delete of student information and courses must be possible
- Enters faculty information(name, id#, department)
- Enters courses with their pre-requisite courses.
    - Add, Edit and Delete of courses and their pre-requisite must be possible
- Offers courses for student enrollment
- Handles room assignment of courses (at least 10 rooms) and scheduling of courses (follow the MW and TH 1.5 hours)
    - Add, Edit and Delete of room assignment must be possible
- Maintains and displays a list of faculty members, their info including their department.

courses.txt

```
<course code1> <space> <section> <space> <units> <space> <day> <space>
<time> <space> <room> <space> <faculty> <newline>
. . .
<course codeN> <space> <section> <space> <units> <space> <day> <space>
<time> <space> <room> <space> <faculty> <newline>
<eof>
```

prerequisite.txt

```
<pre-requisite course code1> <space> <course code> <newline>
<pre-requisite course code2> <space> <course code> <newline>
:
<eof>
```

rooms.txt

```
<room no.><space><day><space><time><space><course code1><space><section><space><#of
occupancy><newline>
:
<room no.><space><day><space><time><space><course code1><space><section><space><#of
occupancy><newline>
<eof>
```

SAMPLE OUTPUT:

Main Menu

1] Student
2] Faculty
3] Academic Assistant
Answer: 3

Academic Assistant Menu

1] Enter student information
2] Enter faculty information
3] Enter courses and prerequisites
4] Schedule and Room Assignment
5] Display Faculty List
6] Exit to Return to Main Menu
Answer: 1

Enter student ID: 123456
Enter name: Juan dela Cruz
Enter course: BSCS
Enter subjects taken (Type Exit to stop the entry)
CCPROG1
CCICOMP
MTH101A
STT101A
GEPCOMM
NSTP101
Exit

Main Menu

1] Student
2] Faculty
3] Academic Assistant
Answer: 1

Student Menu
Enter Student ID: 123456

Choose your transaction:
1] Enroll Courses
2] Print EAF
3] Exit to Return to Main Menu
Answer: 1

Enroll Courses
1] Add Courses
2] Edit Courses
3] Delete Courses

Answer:1

Adding Courses
You have no enrolled courses yet.
Here are the Offered Courses:

SUBJECT	Section	Units	Day	Time	Room	Faculty
CCPROG2	S11A	3	MW	0915-1045	G306A	
CCPROG2	S12A	3	TH	1245-1415	G302A	Santos, Miguel
FORMDEV	S13	3	TH	1100-1230	G211	
CCDSTRU	S11	3	TH	0730-0900	G204	Rodriguez, Tom
CSALGCM	S14	3	TH	1245-1415	G207	Lee, King
CSARCH1	S11	3	TH	1430-1600	G206	
CCAPDEV	S17	3	TH	1245-1415	G302B	Tan, Timothy
:						
ST-MATH	S15	3	TH	1615-1745	G204	

Enter your courses and section to enroll. For example: CCPROG1 S11. Type EXIT once you are done with the entry.
CCDSTRU S12
CCPROG2 S12A
ST-MATH S15

CCAPDEV S17
EXIT

Below are the approved enrolled courses.
CCPROG2 S12A 3 TH 1245-1415 G302A Santos, Miguel

Below are not allowed to enroll.

CCAPDEV. You have not taken CCINFOM yet.
ST-MATH. You have not taken CSMATH1 yet.
CCDSTRU S12. Incorrect section.

Main Menu

1] Student
2] Faculty
3] Academic Assistant
Answer: 2

Faculty Menu
1] Enter Faculty Deloading Units
2] Select Course Load
3] Print Course Load
4] Print Student List
5] Exit to Return to Main Menu
Answer: 1

Faculty Information
Enter ID#: 102345
Enter Deloading Units: 3
You can teach 9 units of courses.

Faculty Menu
1] Enter Faculty Deloading Units
2] Select Course Load
3] Print Course Load
4] Print Student List
5] Exit to Return to Main Menu
Answer: 2

Here are the available courses:

CCPROG2 S11A 3 MW 0915-1045 G306A

FORMDEV	S13	3	TH	1100-1230	G211
CSARCH1	S11	3	TH	1430-1600	G206
ST-MATH	S15	3	TH	1615-1745	G204

Enter your preferred courses and section (Type EXIT to stop the entry)
CCPROG2 S11A
CSARCH1 S11
ST-MATH S15
EXIT

Bonus

A **maximum** of **10 points** may be given for features **over & above** the requirements, like using a binary file in your program or other features not conflicting with the given requirements or changing the requirements) subject to **evaluation** of the teacher. **Required features** must be **completed first** before bonus features are credited. Note that use of conio.h, or other advanced C commands/statements may **not** necessarily merit bonuses.

Note though that you are NOT ALLOWED to do the following:

- to declare and use global variables (i.e., variables declared outside any function),
- to use goto statements (i.e., to jump from code segments to code segments), [gotoxy() is allowed]
- to use the break or continue statement to exit a block. Break statement can only be used to break away from the switch block,
- to use the return statement or exit statement to prematurely terminate a loop or function or program,
- to use the exit statement to prematurely terminate a loop or to terminate the function or program, and
- to call the main() function to repeat the process instead of using loops.

It is best that you perform your coding "incrementally." This means:

- Dividing the program specification into subproblems, and solving each problem separately according to your algorithm;
- Code the solutions to the subproblems one at a time. Once you're done coding the solution for one subproblem, apply testing and debugging.

Submission & Demo

- **Final MP Deadline: March 31, 0730 AM via AnimoSpace**. No project will be accepted anymore after the submission link is locked and the grade is automatically 0.

  Requirements: Complete Program

  - Make sure that your implementation has considerable and proper use of arrays, structures, files, and user-defined functions, as appropriate, even if it is not strictly indicated. See details on the note regarding "Not Allowed" above.

  - It is expected that each feature (except for the exit or back to menu features) is supported by at least one function that you implemented. Some of the functions may be reused (meaning you can just call functions you already implemented [in support] for other features. There can be more than one function to perform tasks in a required feature.

  - Debugging and testing was performed exhaustively. The program submitted has

    a. NO syntax errors

    b. NO warnings - make sure to activate -Wall (show all warnings compiler option) and that C99 standard is used in the codes

c. NO logical errors -- based on the test cases that the program was subjected to

Important Notes:

1. Use **gcc -Wall** -std=C99 to compile your C program. Make sure you **test** your program completely (compiling & running).
2. Do not use brute force. Use **appropriate conditional** statements **properly**. Use, **wherever appropriate**, **appropriate loops** & **functions properly**.
3. You **may** use topics outside the scope of CCPROG2 but this will be **self-study**. Refer to restrictions stated above (regarding "Not Allowed").
4. Include **internal documentation** (comments) in your program.
5. The following is a checklist of the deliverables:

Checklist:

- ❒ Upload via AnimoSpace submission:
  - ❒ source code*
  - ❒ test script**
  - ❒ sample text file exported from your program
- ❒ email the softcopies of all requirements as attachments to **YOUR** own email address on or before the deadline

Legend:

***** Source code exhibit readability with supporting inline documentation (not just comments before the start of every function) and follows a coding style that is similar to those seen in the course notes and in the class discussions. The first page of the source code should have the following declaration (in comment) [replace the pronouns as necessary if you are working with a partner]:

```
/*********************************************************************************
**********************
This is to certify that this project is my own work, based on my personal efforts in studying and applying
the concepts learned.  I have constructed the functions and their respective algorithms and
corresponding code by myself.  The program was run, tested, and debugged by my own efforts.  I
further certify that I have not copied in part or whole or otherwise plagiarized the work of other
students and/or persons.
                                                        <your full name>, DLSU ID# <number>
*********************************************************************************
**********************/
```

Example coding convention and comments before the function would look like this:

```
/* funcA returns the number of capital letters that are changed to small letters
   @param strWord - string containing only 1 word
   @param pCount - the address where the number of modifications from capital to small are
                   placed
   @return 1 if there is at least 1 modification and returns 0 if no modifications
   Pre-condition: strWord only contains letters in the alphabet
*/
int     //function return type is in a separate line from the
funcA(char strWord[20] ,    //preferred to have 1 param per line
     int * pCount)          //use of prefix for variable identifiers
{  //open brace is at the beginning of the new line, aligned with the matching close brace
   int    ctr;       /* declaration of all variables before the start of any statements –
                        not inserted in the middle or in loop- to promote readability */

   *pCount = 0;
   for (ctr = 0; ctr < strlen(strWord); ctr++)  /*use of post increment, instead of pre-
                                                                    increment */
   {  //open brace is at the new line, not at the end
      if (strWord[ctr] >= 'A' && strWord[ctr] <= 'Z')
      {    strWord[ctr] = strWord[ctr] + 32;
           (*pCount)++;
      }
      printf("%c", strWord[ctr]);
   }

   if (*pCount > 0)
      return 1;
   return 0;
}
```

Test Script should be in a table format. There should be at least 3 categories (as indicated in the description) of test cases **per function. There is no need to test functions which are only for screen design (i.e., no computations/processing; just printf).

Sample is shown below.

Function	#	Description	Sample Input Data	Expected Output	Actual Output	P/F
sortIncreasing	1	Integers in array are in increasing order already	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P
	2	Integers in array are in decreasing order	aData contains: 53 37 33 32 15 10 8 7 3 1	aData contains: 1 3 7 8 10 15 32 33 37 53	aData contains: 1 3 7 8 10 15 32 33 37 53	P

	3	Integers in array are combination of positive and negative numbers and in no particular sequence	aData contains: 57 30 -4 6 -5 -33 -96 0 82 -1	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	aData contains: -96 -33 -5 -4 -1 0 6 30 57 82	P

Test descriptions are supposed to be unique and should indicate classes/groups of test cases on what is being tested. Given the sample code in page 7, the following are four distinct classes of tests:

i.) testing with strWord containing all capital letters (or rephrased as "testing for at least 1 modification")

ii.) testing with strWord containing all small letters (or rephrased as "testing for no modification")

iii.) testing with strWord containing a mix of capital and small letters

iv.) testing when strWord is empty (contains empty string only)

The following test descriptions are **incorrectly formed**:

- Too specific: testing with strWord containing "HeLlo"
- Too general: testing if function can generate correct count OR testing if function correctly updates the strWord
- Not necessary -- since already defined in pre-condition: testing with strWord containing special symbols and numeric characters

6. Upload the softcopies via Submit Assignment in AnimoSpace. You can submit multiple times prior to the deadline. However, only the last submission will be checked. Send also to your **DLSU GMail account** a **copy** of all deliverables.

7. You are allowed to create your own modules (.h) if you wish, in which case just make sure that filenames are descriptive.

8. During the MP **demo**, the student is expected to appear on time, to answer questions, in relation with the output and to the implementation (source code), and/or to revise the program based on a given demo problem. Failure to meet these requirements could result to a grade of 0 for the project.

9. It should be noted that during the MP demo, it is expected that the program can be compiled successfully and will run in the command prompt using gcc -Wall -std=C99. If the program does not run, the grade for the project is automatically 0. However, a running program with complete features may not necessarily get full credit, as implementation (i.e., code) and other submissions (e.g., non-violation of restrictions evident in code, test script, and internal documentation) will still be checked.

10. The MP should be an HONEST intellectual product of the student/s. For this project, you are allowed to do this individually or to be in a group of 2 members only. [Note that if you decide to work individually, you may still be subject to the Individual Demo Problem.] Should you decide to work in a group, the following mechanics apply:

Individual Solution: Even if it is a group project, each student is still required to create his/her INITIAL solution to the MP individually without discussing it with any other students. This will help ensure that

each student went through the process of reading, understanding, solving the problem and testing the solution.

Group Solution: Once both students are done with their solution: they discuss and compare their respective solutions (ONLY within the group) -- note that learning with a peer is the objective here -- to see a possibly different or better way of solving a problem. They then come up with their group's final solution -- which may be the solution of one of the students, or an improvement over both solutions. Only the group's final solution, with internal documentation (part of comment) indicating whose code was used or was it an improved version of both solutions) will be submitted as part of the final deliverables. It is the group solution that will be checked/assessed/graded. Thus, only 1 final set of deliverables should be uploaded by one of the members in the AnimoSpace submission page. [Prior to submission, make sure to indicate the members in the group by JOINing the same group number.]

Individual Demo Problem: As each is expected to have solved the MP requirements individually prior to coming up with the final submission, both members should know all parts of the final code to allow each to INDIVIDUALLY complete the demo problem within a limited amount of time (to be announced nearer the demo schedule). This demo problem is given only on the day/time of the demo, and may be different per member of the group. Both students should be present during the demo, not just to present their individual demo problem solution, but also to answer questions pertaining to their group submission.

Grading: the MP grade will be the same for both students -- UNLESS there is a compelling and glaring reason as to why one student should get a different grade from the other -- for example, one student cannot answer questions properly OR do not know where or how to modify the code to solve the demo problem (in which case deductions may be applied or a 0 grade may be given -- to be determined on a case-to-case basis).

11. Any form of **cheating** (**asking other people not in the same group for help**, **submitting as your [own group's] work part of other's work**, **sharing your [individual or group's] algorithm and/or code to other students not in the same group, etc.**) can be punishable by a grade of **0.0** for the **course** & a **discipline case**.

Any requirement not fully implemented or instruction not followed will merit deductions.