COP 3502C Programming Assignment # 1
Dynamic Memory Allocation
Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will heavily use dynamic memory allocation. Your solution should follow a set of requirements to get credit.

What should you submit?
Write all the code in a single file and upload the .c file to Webcourses.

Please include the following commented lines in the beginning of your code to declare your authorship of the code:

```
/* COP 3502C Assignment 1
This program is written by: Your Full Name */
```

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.
Caution!!!
Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:
See the deadline in Webcourses. The assignment will accept late submission up to 24 hours after the due date time with 10% penalty. After that the assignment submission will be locked. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification or need help?

Write an email to the TA and put the course teacher in the cc for clarification on the requirements. I will also create a discussion thread in webcourses and you can ask questions there too.

How to get help if you are stuck?

According to the course policy, all the helps should be taken during office hours. Occasionally, we might reply in email.

Problem

WOW training center offers certain number of courses. A course can have one or more sections. A section can have one or more students (assume that a student cannot enroll in multiple sections of the same course). A student has to do a certain number of assignments in the course. In order to pass the course, the average score of the assignment has to be >=70.

You will write a program that will read data about the courses, sections, students, and scores of the students and will produce some summarized results from the data.

Input Specification (read data from a file in.txt. Do not use any other file name in your code.):

The first line of the file contains a single positive integer, *t (t<=25)*, that represents the number of test cases to process. The test cases follow.

The first line of each test case contains a single positive integer, *c (c<=500),* that represents the number of courses. After that the file contains data for c number of courses.

For each course:

The first line of a course contains a string, *cn (all lower case, single word, max length <=20)*, that represents the course name.

The next line contains a single positive integer, *s (s<=10),* that represents the number of sections in the course *cn*. After that the file contains data for *s* number of sections for the course as follows.

For each section:

The first line contains two positive integers, *st (st<=500)* and *m (m<=20)*, where *st* represents the number of students in the section and *m* represents the number of assignments in the section.

The following *st* number of lines represents data for the *st* number of students (each line for each student).

A student data line contains a positive integer, *id (id<=50,0000),* a string *lname (all lower case, single word, max length <=20)*, and *m* positive float numbers *(each float number is <=100)* separated by spaces, where *id* represents the id number of the student, *lname* represents the last name of the student, and *m* positive float numbers represent the score of the students in *m* assignments for the course. Assume that the id numbers are unique for each student.

Similarly, the file contains data for *s* number sections for the course *cn*. Similarly, the file contains data for all the courses for each test case.

Output (output should be in console. No need to write the output to a file):

For each course, display course name (in the same order they appear in the file), total number of students passed, average scores for each section of the course (in the same order they appear in the file), id, last name and average score of the student who achieved highest average score in the assignments of the entire course regardless of section (print the first student only if multiple students achieved the highest score). Display the result in one course per line in the following format:

course_name pass_count list_of_averages_section(separated by space up to two decimal places) id lname avg_score (up to two decimal places)

Where course_name is the name of the course, pass_count is the total number of students passed the course, list_of_averages_section is the list of average scores per section of the course, id, lname and avg_score is the id, last name, and average score of the student who achieved the highest score in the course.

Sample in.txt

```
2 //number of test cases
3 //num of courses for test case1. The following lines are for test case 1
cs1 //name of course 1
2 //number of sections for course 1 (cs1)
3 4 //no. of students and assignments for sec1 of course 1 (cs1)
101 john 70 60.5 95.2 50.6 //id lname scores
102 tyler 80 60.5 95.2 66.6
103 nusair 70 60.5 85.2 50.6
2 3 //no. of students and assignments for sec2 of course 1 (cs1)
105 edward 90.5 60.5 98.2
104 alan 40 60.5 95.2
math2 //name of course 2
3 //number of sections for course 2 (math2)
2 2 //no. of students and assignments for sec1 of course 2 (math2)
101 john 95.2 53.6
103 nusair 86.2 56.6
2 3 //no. of students and assignments for sec2 of course 2 (math2)
105 edward 90.5 60.5 98.2
104 alan 40 60.5 95.2
3 2 //no. of students and assignments for sec3 of course 2 (math2))
110 kyle 90.5 98.2
108 bob 45 85.2
109 smith 75.5 65.9
physics3 //name of course 3
1 //number of sections of course 3
4 2 //num of students and assignments for sec1 of course3 (physics3)
```

```
105 edward 60.5 98.2
104 alan 40.5 95.2
108 bob 55 85.2
109 smith 65.5 68.9
2//num of courses for test case2. The following lines are for test case2
cs1 //name of course 1
2 //number of sections for course 1 (CS1)
2 3 //no. of students and assignments for sec1 of course 1 (cs1)
102 habib 90.5 60.5 98.2
101 mohamed 40 60.5 95.2
4 3 //num of students and assignments for sec2 of course 1 (cs1)
104 manha 85.5 60.5 95.2
102 habib 80 60.5 95.2
103 hussain 70 60.5 85.2
109 ali 78.0 63.5 85.5
physics2 //name of course 2
3 //number of sections for course 2 (physics2)
2 2 //num of students and assignments for sec1 of course 2 (physics2)
101 mohamed 95.2 53.6
103 hussain 86.2 56.6
2 3 //num of students and assignments for sec2 of course 2 (physics2)
105 aziz 90.5 60.5 98.2
104 manha 40 60.5 95.2
3 2 //num of students and assignments for sec3 of course 2 (physics2)
110 ahmed 90.5 98.2
108 hasan 45 85.2
109 nadia 75.5 65.9
```

Sample Output

```
test case 1
cs1 2 70.41 74.15 105 edward 83.07
math2 5 72.90 74.15 76.72 110 kyle 94.35
physics3 2 71.12 105 edward 79.35

test case 2
cs1 5 74.15 76.63 102 habib 83.07
physics2 5 72.90 74.15 76.72 110 ahmed 94.35
```

Requirements for the assignment:

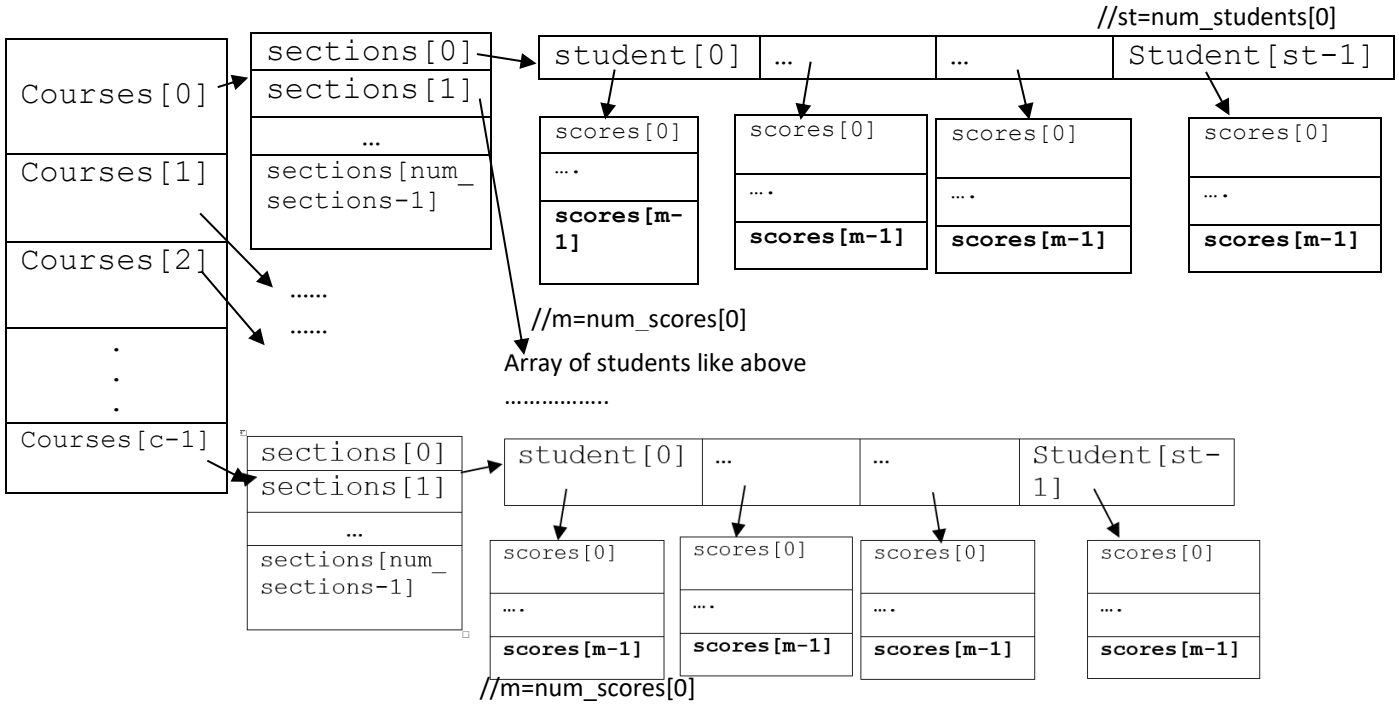
1. In order to store the students, courses and sections, you have to use the following structure definition. You are not allowed to add or delete any member for the structure.

```
typedef struct student{
   int id;
   char *lname; //stores last name of student
   float *scores; //stores scores of the student. Size is taken from num_scores array.
```

```
    float std_avg; //average score of the student (to be calculated)
}student;

typedef struct course{
   char *course_name; //stores course name
   int num_sections; //number of sections
   student **sections;//stores array of student arrays(2D array). Size is num_sections;
   int *num_students;//stores array of number of students in each section. Size is num_sections;
   int *num_scores; //stores array of number of assignments in each section. Size is num_sections;
} course;
```

A partial view of the memory map of the arrays will be like the following figure. *Courses is an array of course. Sections of a course is an array of arrays of students (a.k.a 2D array of students)* .



2. You have to use dynamic memory allocation for the arrays and strings based on the size and data you will receive from the file.
3. After processing the job of a test case, you must have to free memory appropriately.
4. Your code should implement **<u>at least</u>** the following functions:
    a. `course *read_courses(FILE *fp, int *num_courses)`: This function takes a file pointer and reference of an integer to track how may courses the file has. Then it reads the data for an entire test case and return the allocated memory for all the courses (including sections) for a test case. Note that you can call other functions from this function as needed.
    b. `student **read_sections(FILE *fp, int num_students[], int num_scores[], int num_sections)`: This function takes the file pointer, references of two arrays, one for

number of students, and one for number of scores for a course. The function also takes an integer that indicates the number of sections the course has. The function reads all the data for all the sections of a course, fill-up the num_students and num_socres array of the course and returns 2D array of students that contains all the data for all the sections of a course. A good idea would be calling this function from the read_course function.

c. `void process_courses(course *courses, int num_courses):` This function takes the array of courses produced and filled by all the courses of a test case and also takes the size of the array. Then it displays the required data in the same format discussed in the sample output. You can write and use more functions in this process as you wish.

d. `void release_courses( course *courses, int num_courses):` This function takes the array of courses produced and filled by all the courses of a test case and also takes the size of the array. Then it free up all the memory allocated within it. You can create more function as needed to ease the process.

e. You must have to include #include "leak_detector_c.h", in your code and must add the line atexit(report_mem_leak); in the beginning of your main function for memory leak detection. Keep the uploaded leak_detector_c.c and leak_detector_c.h file in the same directory while testing your code

Rubric:

The code will be compiled and tested in Eustis server for grading. If your code does not compile in Eustis, we conclude that your code is not compiling and it will be graded accordingly. We will apply a set of test cases to check whether your code can produce the expected output or not. Failing each test case will reduce some grade based on the rubric given bellow. If you hardcode the output, you will get -200% for the assignment.

1. If a code does not compile: 0
2. If you modify the required structure or do not use them: 0
3. Not using dynamic memory allocation for storing courses, strings, etc., will receive 0
4. There is no grade for a well indented and well commented code. But a bad indented code will receive 20% penalty. Not putting comment in some important block of code -10%
5. We will apply three test cases in a single in.txt file:
    a. Each test case with exact output format is: 20% (total 60%)
    b. Freeing up memory properly with zero memory leak (if all the required malloc implemented): (30%)
    c. Writing and using all the required functions: 10%

Study the lecture notes and labs for learning dynamic memory allocation and fie I/O. Use fscanf with appropriate format specifier (%d/%f/%s, etc) for reading file. It will make the process easier.

Good Luck!