2020 Summer Computer Science I Section 2
Program 1: Dynamic Memory Allocation
Please Consult WebCourses for the due date/time
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 must use dynamically allocated memory and appropriately free the memory in the method prescribed to get full credit.

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 all of her assignments must be at least 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.

For the purposes of this problem, we define the average score of a student in the course to be the average of all of the student's assignment scores. We define the average score of a section to be the average of the average scores of all of the students.

Consider the following example: if one section of a course has three students, alia, barney and carmen. Let alia's scores on the four assignments in the course be 100, 90, 80 and 95, for an average of 91.25. Let barney's scores on the four assignments in the course be 83, 93, 90 and 95, for an average of 90.25. Let carmen's scores on the four assignments in the course be 100, 70, 85, and 93, for an average of 87.00. The average score for this section would be (91.25 + 90.25 + 87.00)/3 = 89.50, rounded to two decimal places.

Input Specification (please read from standard input, with no prompts for entering information, test using file redirection):
The first line of the file contains a single positive integer, t ($t \leq 25$), 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 ($c \leq 500$), that represents the number of courses. After that, the input contains data for c number of courses.

For each course:

The first line of a course contains a string, *cn* (all lower case letters or digits, max length ≤ 20), that represents the course name.

The next line contains a single positive integer, s ($s \leq 10$), 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* ≤ 500,000), a string *lname* (all lower case letters, max length ≤ 20), and *m* positive floating point numbers (each number is ≤ 100.0) separated by spaces, where *id* represents the id number of the student, *lname* represents the last name of the student, and the *m* positive floating pointing numbers represent the scores of the student on the *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 (to standard output):

For each test case, output a single line header with the following format:

```
test case x
```

where x is the test case number, starting with one.

Within each test case, produce a single line of output for each course. For each course, display the course name (in the same order they appear in the file), total number of students who passed the course, the 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, in terms of where they appear in the file, 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 rounded to exactly two decimal places) id lname avg_score (rounded to exactly 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 Input

```
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
```

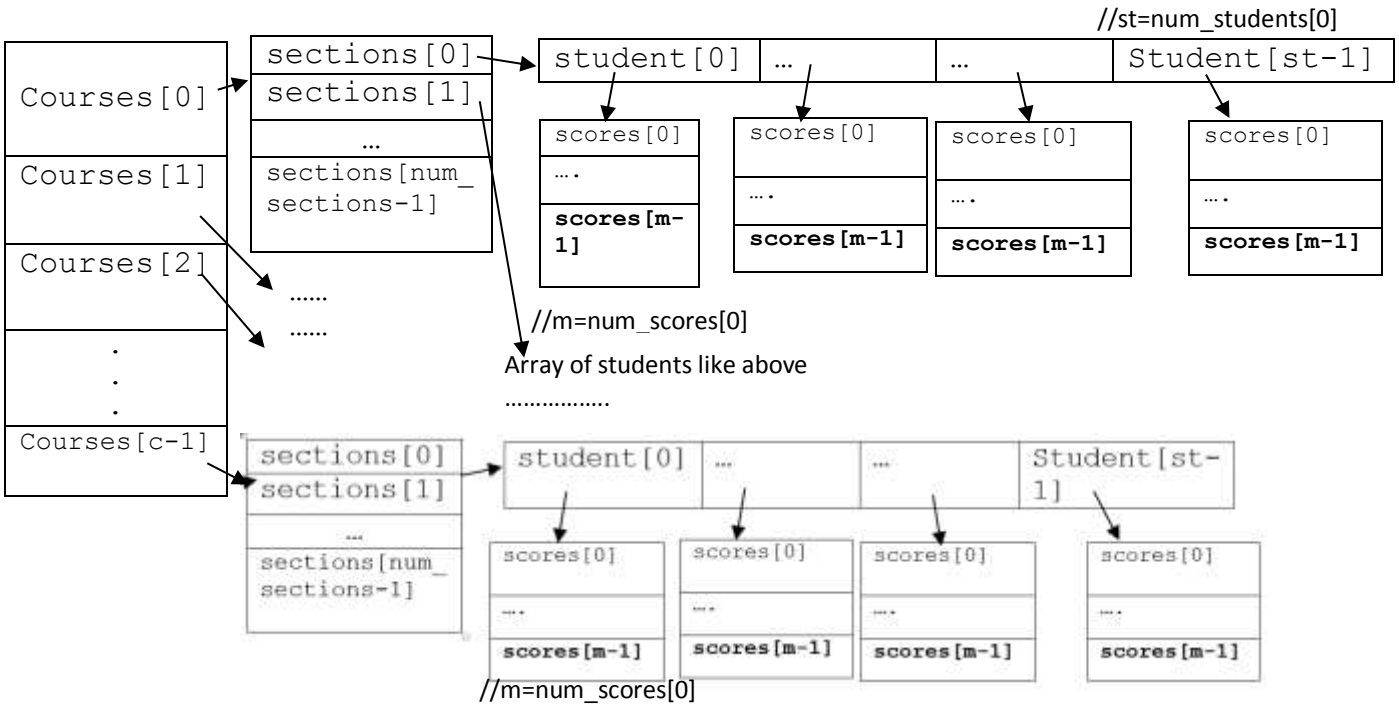
Implementation Details - Criteria for Full Credit

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(int *num_courses)`: This function takes the 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(int num_students[], int num_scores[], int num_sections)`: This function takes the references of two arrays, one for number of students, and one for number of scores for a course, as well as the number of sections, which is the length of both arrays. The function reads all the data for all the sections of a course, fills up the num_students and num_scores array of the course and returns a 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 the size of the

array. 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. It frees up all the memory allocated within it. You can create more function as needed to ease the process.

Implementation Detail - Optional (Extra Credit)

In class, a tool called leak_detector was shown, which, can be used to identify memory leaks in your code. The necessary code and directions to use it are on the course webpage. If you choose to use it, please do the following:

1. Put #include "leak_detector_c.h" after your usual #includes.

2. Add the line atexit(report_mem_leak); in the beginning of your main function.

3. Keep both the files leak_detector_c.c and leak_detector_c.h in the same directory as your source file.

4. When compiling, do the following steps at the command prompt:

> gcc -c leak_detector_c.c
> gcc -c coursesummary.c
> gcc -o memtest leak_detector_c.o coursesummary.o
> memtest

After running the executable memtest, there will be a file created called leak_info.txt.
Open this file to see what, if any memory leaks are in your program.

Grading Details

While I don't typically make public my detailed grading rubric, here are some portions of it to keep in mind:

1. The code will be compiled and tested in CodeBlocks using the mingw compiler from the following download (codeblocks-20.03mingw-setup.exe). The maximum grade a program can receive without compiling is 50% and will typically be lower.

2. There will be a deduction for programs that are 100% correct that do use different structures than outlined in this assignment or do not implement the functions described.

3. There will be deductions for programs that do not allocate memory dynamically as designated in the instructions.

4. There will be deductions for programs that do not properly free the dynamically allocated memory.

5. Full credit can only be earned if the following style issues are properly address:

a) Header Comment
b) Good Use of White Space
c) Meaningful Variable Names
d) Reasonable Internal Comments
e) Comment for each Function indicating what it takes in and what it does/returns.

6. A significant portion of the grade (30% - 50%) will be based on the correctness of the secret test cases (posted after the assignment is due).

7. Deductions may be made for stylistic issues not affecting correctness (a long sequence of unnecessary if statements, for example) that are not outlined here.

Deliverables

You must submit a two files over WebCourses:

1) A source file, *coursesummary.c*. Please use stdin, stdout. There will be an automatic 10% deduction if you don't do so.

2) Your Process Document, which should be named LASTNAME_PROCESS.doc (or .docx or .pdf or .txt)

Notes About Assistance

If you need help on the assignment, please visit the Zoom office hours of the TA or Instructor, or visit the SARC hours listed in Webcourses. Do NOT copy code from others or discuss low level details of the assignment with others outside of the course staff or SARC. You may get minor help debugging from others but not help solving the problem. (For example, if your parents can spot an accidental = vs. == issue, I see no problem with them telling you. However, if a friend gives you a function that calculates a section's average grade, that is not allowed.)