GRADES (contributed by F. R. Salvador)

1. INTRODUCTION

Yamada-Sensei[1] teaches an Advanced Programming class. As preparation for an upcoming Midterm Examination, he gave his students a Practice Quiz and asked them to solve it as a reviewer. The Practice Quiz is worth a total of 30 points if solved correctly. Since it was not a required graded activity, students were given the freedom to choose whether to solve it or not. Some students solved it, while others did not. The actual Midterm Exam which is worth a total of 50 points, if solved correctly, was then given a few days after.

Information about some of the students' Last Name and First Name, Practice Quiz score and actual Midterm exam scores are tabulated in four columns visualized below. Each row represents one student record. The colons represent student data that are not shown in this document due to limited space.

Last Name	First Name	Practice Quiz Score (max 30 points)	Midterm Exam Score (max 50 points)
OKAMURA	MASAHIRO	0	25
ABE	RIKA	27	42
ENDO	SABURO	8	28
NISHIDA	EMI	0	17
:	:	:	:
:	:	:	:
UEDA	MINAKO	14	27

Additional information about the data:

a. All names are in capital letters.

b. If a student did **not** solve the Practice Quiz, the encoded score is **0**. For example, student OKAMURA MASAHIRO did not solve it. On the other hand, ABE RIKA solved it and obtained a score of 27.

2. SKELETON FILE AND OTHER FILES

a. **`GRADES-LASTNAME.c`** - this is the skeleton file that you'll need to edit as base code. Make sure to read and understand the contents and instructions in the skeleton file. Don't forget to rename this file with your own last name. For example, if your last name is SANTOS, then the file should be renamed as GRADES-SANTOS.c.

b. **`main`**.c - this is the file that contains the main() function. Edit Line 18: #include "GRADES-LASTNAME.c" to replace LASTNAME with your own file.

c. **`grades.h`** - this is a header file that contains the structure type declaration for the student structure and function declarations.

d. **`TESTDATA.TXT`** - this text file contains the student data encoded in four columns as explained and visualized above. Open it using any text editor (for example, Notepad) and study its contents. Use it to test your solution; see Section 4. Feel free to edit the file contents to accommodate your own tests.

e. **`EXPECTED.TXT`** - this text file contains expected output of a logically correct solution to the problem requirements.

3. REQUIRED TASKS

Edit **`GRADES-LASTNAME.c`** and encode your solutions to the five tasks described below.

Task #1: Define the **Print_Student_List ()** function which will **printf()** the values of all data in the list of student records starting from the first index to the last index. Each line of output should display the Last Name, First Name, the Practice Quiz Score and the Midterm Exam Score value separated by at least one space. **DO NOT** print any extraneous or unnecessary character or string.

The following is the expected output using input data from **`TESTDATA.TXT`**.

[1] In Nihongo, "sensei" means "teacher".

```
OKAMURA  MASAHIRO  0  25
HARADA  RIKA  27  42
     :
     :  // colon means other data values which are not shown here due to limited space
     :
UEDA  MINAKO  14  27
```

Task #2: Define **Append_Student_List()** function that will append new data in the list of car structures. Append means to add new elements (student data) at the end of the list. Assume that there is memory space still available, and that the new data values are different from those already in the list.

Example: Appending new data, for example, **MAKINO ICHIRO 30 50** into the list, and then printing the updated list will result into:

```
OKAMURA  MASAHIRO  0  25
HARADA  RIKA  27  42
     :
     :  // colon means other data values which are not shown here due to limited space
     :
UEDA  MINAKO  14  27
MAKINO ICHIRO 30  50
```

Task #3: Define a C function that will **sort** the list of structures by LAST NAME and FIRST NAME in alphabetical order. Be careful in formulating your sorting function for students with the same last names (for example SUZUKI SHINTAROU and SUZUKI KEIKO). Note that SUZUKI KEIKO must appear before SUZUKI SHINTAROU in the sorted list – hint: you'll need to use `strcat()`.

HARD REQUIREMENT: You are required to use **SELECTION SORT ALGORITHM** as explained in Chapter 1 of our Course Notes. Do NOT use a different sorting algorithm. Non-compliance will render your solution incorrect which means the score for this task will be 0.

Task #4. Define a C function that will compute the answer to the following question:

Q: *Is* ***<param_last_name>*** ***<param_first_name>*** *in the list of students?*

For this task, it is assumed that the array of student records is already sorted alphabetically by name. There is NO need to call the sorting function inside the function definition.

The function should perform a **BINARY SEARCH** to determine if a student whose name matches the **<param_last_name>** and **<param_first_name>** is in the list of structures or not. For simplicity, assume that all the letters in the parameters are also in upper case. If the search key is found, the function should return the **index** corresponding to where it was found in the array; otherwise, it should return -1.

Example #1. Q: *Is MAKINO ICHIRO in the list of students?*
A: 4 /* a matching student record is in the list, found in index 4 */

Example #2: Q: *Is SUZUKI JOSELITO in the list of students?*
A: -1 /* no matching student record */

Task #5. Define a C function that will compute the answer to the following question (query):

Q: *Who are the students whose Midterm Exam score is LOWER than* ***<param_score>*** *and did* ***NOT*** *solve the Practice Quiz?*

For this task, it is assumed that the array of student records is already sorted alphabetically by name. There is NO need to call the sorting function inside the function definition.

The function should build a NEW list (i.e., array) of student structures that contain data that meet the query conditions. The function should return the number of students (an integer) value that satisfied the query. For example:

Q: *Who are the students whose Midterm Exam score is LOWER than 30 and did NOT solve the Practice Quiz?*

The new list of structures will contain the following data. NOTE: the function should NOT print the data!!!

```
HIMURA  KENSHIN  0  29
NISHIDA  EMI  0  17
OKAMURA  MASAHIRO  0  25
```

The function returns 3 which means that there are 3 student data the met the query condition.

4. HOW TO COMPILE, RUN AND TEST YOUR PROGRAM

Compile your C program, either inside the IDE or in the command line interface. Make sure that there are no syntax/compilation errors. Let's assume that the source file is named as **`GRADES-SANTOS.c`**, and that the executable file is named as **`GRADES-SANTOS.exe`**.

Run the exe file in the command line **with I/O redirection** as shown in the example below.

```
C:\CCPROG2> GRADES-SANTOS < TESTDATA.TXT > OUTPUT-SANTOS.TXT
```

The input redirection will enable data to be read via **`scanf()`** from the text file. Make sure that **`TESTDATA.TXT`** is in the same folder as your C source file. The program should produce the desired output following the format described in Section 3.

The accompanying **`EXPECTED.TXT`** file contains the expected output based on the original values in **`TESTDATA.TXT`** and new data as appended in the **`main()`** function.

5. SUBMIT YOUR FILES VIA CANVAS

Submit/upload two files before the Canvas deadline:

a. **`GRADES-LASTNAME.c`** -- your C source file solution. Don't forget to rename your file with your own last name.
b. **`OUTPUT-LASTNAME.txt`** -- your program's output as described in the I/O redirection example above.

Back-up your solution (files) by sending it as an email attachment to your DLSU email account. Do not delete that email until you have completed CCPROG2.

6. TESTING & SCORING:

- Your program will be black box tested with a different set of test data, and/or different main() function.
- Each correct function definition will be given **10 points** each. Thus, the maximum total score will be 50/50.
- A program that has a syntax/compilation error will be given a score of 0 out 50.
- The score for an incorrect implementation of a required function is 0. For example, if the only correct solution is for Tasks 1 and 2, then the score will be 20/50.

-- The End --