COMP 2511

Assignment 1

1 Introduction

For this assignment, we are going to write a program that allows the user to enter scores for students, save them to a file & retrieve them later. The main purpose is to practise using the I/O functions in the standard C library.

Note that in this course, we limit ourselves to ANSI C. Except when explicitly stated to the contrary, you must implement every function you use that is not in the ANSI standard C library.

2 Input/Output

There are a number of useful rules concerning input/output:

- Always use `fgets` and `sscanf` for interactive input (i.e., input from the keyboard); functions such as `getchar` should only be used for non-interactive input (e.g., reading from a file using I/O redirection) or when testing/debugging
- Always check for the "end-of-file" condition & handle it appropriately; for `fgets`, this means checking for the null pointer as its return value. (Note: `fgets` actually returns the null pointer on end-of-file or file error.) For `getchar`, this means comparing its return value with `EOF`.
- Always make sure you have successfully "read" the items you need; in particular, this means checking the return value of `sscanf`

In general, it's a good idea to check the return value of every function you call.

3 The Main Menu

On startup, the program displays a menu whereby the user can choose to

1. Enter data
2. Display data
3. Quit

(Note: The choices must be numbered exactly as shown above.)

Quitting the program is obvious. However, some cleanup may need to be performed before exiting the program. In addition, the user can use the "end-of-file key" to quit the program at the main menu.

Data entered by the user is saved to a file named `data.txt` (in the current directory). This file should be created when the program starts. If the file already exists, its content is truncated (deleted). This file should be opened only once at the beginning of the program — it is not a good idea to keep on opening & closing the same file.

To facilitate testing, the program ignores extra trailing words in user input. For example, if the user enters the following at the main menu

```
3 hello world
```

the program uses 3 as the choice.

4 Entering Data

The user is prompted for the name of the student & then his/her score.

The name can be entered in 2 forms:

- lastname,firstname (i.e. a comma separates the last name & the first name)
- firstname lastname

Note that the name basically consists of 2 words (the first name & the last name). In the first case, we also allow spaces & tabs before the last name, around the comma & after the first name. In the second case, spaces & tabs are allowed before the first name, between the first name & last name, & after the last name. (See below for additional requirements for the user to exit data entry & go back to the main menu.)

A further requirement is that both the first name & last name must consist only of letters (either uppercase or lowercase) & hyphens. The program should reject any input that is not valid & immediately prompt for it again. Note that as mentioned, extra trailing words are ignored.

We'll use ␣ to denote the space character when necessary. The following are examples of valid input for the name:

- `simpson,␣homer␣`
- `␣␣Simpson␣,homer`
- `␣homer␣␣␣Simpson`
- `Maggie␣Simpson-Flanders`
- `Homer␣J␣Simpson`

Note that in the last example, the first name is `Homer` & the last name is just `J`; `Simpson` is an extranous word & is ignored.

An example of invalid input for the name:

- `R2␣D2`

The score is a integer between 0 & 100 inclusive. If the user enters an invalid score, he should be immediately re-prompted for it. As before, extra trailing words are ignored.

After finishing entering the name & score of one student, the user is prompted to enter the data for another student. This repeats until the user indicates that he is done with entering data by either

- entering the "end-of-file key" when prompted for the name or the score; or
- entering "`!!`" (without the double quotes) as the first word when prompted for the name (this is to facilitate testing via I/O redirection)

When this happens, any partially-entered data should be discarded (this means that if the user has entered the name but then enters the "end-of-file" key for the score, the entered name should be discarded) & the program should return to the main menu. Note that from the main menu, the user can choose to enter more data, display them or quit the program.

Every time the program succeeds in reading the data for a student from the user, it should write that data to the data file as text. You are not allowed to use an "array of student data". The exact file storage format is not specified. But your program must be able to read back the data from the file.

5 Displaying Data

When the user chooses to display data, all the data in the data file are displayed. *To facilitate testing, they must be displayed to standard error.*

The following example shows the output format when information about students is displayed:

```
Simpson, Homer:   5
Simpson, Bart:  25
Flanders, Ned: 100
```

(i.e. last name, first name: score). Note that both the first name & the last name have their first letter in uppercase & the rest in lowercase (even though they may not be so when the user entered them).

After printing all the student information, the minimum, maximum & average of all the scores are printed (again to standard error) & the program returns to the main menu. The following shows the output format:

```
Min:   5, Max: 100, Ave:  66.60
```

The average is printed to 2 decimal places.

Sample input/output files will be provided. You must ensure that your program works correctly at least for those files. This means your output must match the expected output exactly. (Note: We may run more tests when we are marking your assignment.)

6 Additional Requirements

You must format your code properly. You can lose up to 20 marks if your program is not indented correctly.

Appropriate error-handling should be performed. For example, consider what should be done if the specified data file cannot be opened.

The program should be fairly user-friendly. As an example, if the user enters a valid name but an invalid score, he should only need to re-enter the score & not both the name & the score.

Use separate functions & do not use "global" variables.

7 Submission & Grading

This assignment is due at noon on Monday, June 25, 2012. Submit a zip file to `In` in the directory:

```
\COMP\2511\a1\
```

Your zip file should be named `<name_id>.zip`, where `<name_id>` is your name & student ID separated by an underscore (for example, `SimpsonHomer_a12345678.zip`) Do not use spaces to separate your last & first names. Your zip file must unzip directly to your source file(s) without creating any directory. (For this assignment, only one source file is needed. However, you may use multiple source files if you so choose.)

Do not submit rar files. They will not be marked.

If you need to submit more than one version, name the zip file of each later version with a version number after ID, e.g., `SimpsonHomer_a12345678_v2.zip`. If more than one version is submitted, we'll only mark the files in the version with the highest version number. (There must be exactly one zip file with the highest version number; otherwise, your submission will not be marked & you will get no credit for the assignment.)

If your program does not compile under GCC with the standard switches (`-ansi -W -Wall -pedantic`), you may receive zero for the assignment.

Otherwise, the grade breakdown is *approximately* as follows:

Design & code clarity	10%
Error-handling	10%
Main menu & quitting	10%
Data input & validation	35%
Writing data to file (includes opening file)	10%
Displaying data (must read data from file)	25%

Note: As mentioned, marks will be deducted for improperly-indented code.