COMP 2510

Assignment 1

1 Introduction

For this assignment, we are going to write a program that allows the user to enter, display & modify student records & save them to a file. 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 (C89). Except when explicitly stated to the contrary, you need to implement any function that you use & that is not in the standard ANSI C library.

2 Input/Output

A couple of useful rules concerning input/output:

- Use `fgets` and `sscanf` for interactive user input.
- Always check for success/failure immediately after a "read" operation. For `fgets`, this means checking against the null pointer as its return value; for `sscanf`, this means checking whether it returns the correct number of assignments.

3 The Program

The program takes the name of a data file as its only argument on the command-line. For portability reasons, this file should be open in binary mode. However, data is saved as text to the file (using `fprintf`). If the file does not exist already, it is created; otherwise, its content is deleted. *Note that you are allowed to open the file only once.*

In order to simplify this assignment, the user interface is a bit awkward. Basically, the program repeatedly prompts & obtains a command (an integer) from the user & executes that command. The user can quit the program by entering the command for quitting (which is -2) or by pressing the "end-of-file" key when prompted for a command.

Note that in all user input, only the first word (if there is one) in the line the user enters is used. Extra words are simply ignored. However, the first word may need to be converted to the specific data type we need.

The following are the valid commands with brief explanations:

-2 quit

-1 append a record

0 display all records

n (a positive integer) modify record number n (if it exists)

If the user enters an invalid command, it is simply ignored (with no error message) & the user prompted again for a command. Note however that the user can also quit the program using the end-of-file key when prompted for a command.

Quitting the program is obvious. However, some cleanup may need to be performed before exiting the program.

Both appending & modifying a record will require inputting data for a student. The data for a student (a student record) consists of an ID & a score. A valid ID must begin with the character '`a`' followed by exactly 8 digits (e.g., a12345678); a valid score must be an integer between 0 & 100 inclusive (e.g., 65). The program needs to obtain these two pieces of information from the user & write them to the data file.

Only IDs & scores are saved to the data file. However, the records in the file are regarded as numbered, starting from 1. This means that the first record in the file (the first pair of ID & score) has record number 1.

When a record is displayed, it is displayed with its record number. For example, when the user chooses to display all records (by entering 0 for the command), the displayed records are numbered starting from 1. The following sample output shows the output format:

```
1 a12345678 35
2 a23456789 100
3 a11112222 60
```

Note that the record number, the ID & the score are separated by a single space character. *It is important that records be displayed to standard error.* If there is no data to display (e.g., the user has not entered any record), the display command is simply ignored — no error message should be printed.

The numbering of records also applies when the user wants to modify data. When the user enters a positive integer for a command, that integer is regarded as the record number of the record to modify. If that record does not exist, the command is simply ignored (with no error message) & the user re-prompted for another command. If the record exists, it is displayed, again to standard error & with a record number (which should be the same as the number the user entered), before the user is prompted to enter new data for the student.

4 Entering Data for a Student

This is necessary when the user chooses to append or modify a record. To get the data for a student, the program repeatedly prompts the user for an ID until either

1. a valid ID is obtained, or
2. the user presses the end-of-file key, or
3. the user enters -1 (i.e., minus one)

Cases 2 & 3 signify that the user wants to abort the operation. When either happens, the program goes back to prompting the user for a command.

After obtaining a valid ID, the program then repeatedly prompts the user for a score until either

1. a valid score is obtained, or
2. the user presses the end-of-file key, or
3. the user enters -1 (i.e., minus one)

Again, cases 2 & 3 signify that the user wants to abort the operation. When either happens, the program goes back to prompting the user for a command.

If both a valid ID & a valid score are obtained, they are written to the file (in the appropriate location). When modifying a record, the new information overwrites the existing record that we want modified; if appending a record, the new record is written at the end. After writing the record, the program goes back to prompting the user for a command.

The exact file format is not specified. However, only IDs & scores are written to the file & they must be saved as text (using `fprintf`). Furthermore, each pair of ID & score must occupy the same number of characters to enable us to "seek" between records within the file.

For the purpose of obtaining user input, you may assume that each input line has fewer than `LINESIZE` characters where `LINESIZE` is a macro with the value of 1024.

5 Additional Requirements

- Use separate functions & do not use "global" variables.
- You are not allowed to store the information for more than one student in memory.
- *All prompts should be printed to standard output. Records should always be displayed to standard error.* This is to separate the two types of output to facilitate testing. Note that error messages are not needed when the user enters invalid input. Hence error messages, which are also printed to standard error, should be rare. (One case when you need to print an error message is the failure to open the data file.)
- When appending or modifying a record, you must use a "seek" operation (e.g., `fseek`) to go to the correct location; you are not allowed to "loop" through the file, looking for the correct location.
- Format your code properly. You'll lose marks if your code is not indented properly.
- *Sample input/output files will be provided. You must ensure that, for a given input file, the output of your program matches the provided output exactly.*

6 Submission & Grading

Read this section carefully. There are strict requirements on submission. If your submission does not meet these requirements, you may fail to get credit for this assignment.

This assignment is due at 11pm, Friday, February 12, 2016. Submit a zip file to "ShareIn" in the directory:

```
\COMP\2510\a1\set<X>\
```

where `<X>` is your set. 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 (without creating any directories). We'll basically compile your file using

```
gcc -ansi -W -Wall -pedantic *.c
```

after unzipping.

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

If you need to submit more than one version, name the zip file of each later version with a version number after your name, e.g., `SimpsonHomer_a12345678_v2.zip`. If more than one version is submitted, we'll only mark the version with the highest version number. (In this case, there must be exactly one zip file with the highest version number; if there are two or more of them, you may fail to get credit for the assignment.)

Your program must compile without warnings or errors under `gcc` with the following options:

```
-ansi -W -Wall -pedantic
```

Furthermore, you must implement any function that you use & that is not in the standard ANSI C library. (In practice, this usually means that you should not use functions that we have not mentioned in class.) If your program does not meet these requirements, you may receive a score of 0 for the assignment. Otherwise, the grade breakdown is *approximately* as follows:

Design & code clarity (including indentation)	15%
Command & record input (including validation)	30%
Displaying records (must read from file)	15%
Appending records	15%
Modifying records	25%