COMP 2510
Assignment 1

1 Introduction

For this assignment, we are going to write a program that allows the user to enter & display student records as well as saving them to a file. The main purpose is to practise using the I/O functions in the standard C library.

2 Input/Output

There are 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. This file should be opened in binary mode. However, data is saved as text to the file (using `fprintf`). If the file does not exist already, it is created; if it does exist, its content is deleted. New records entered by the user are always appended to the file. The file should only be opened once at the start of the program & closed at the end.

In order to simplify this assignment, we'll use a very basic user interface. Essentially, the program repeatedly prompts & obtains commands (which must be integers) from the user & executes them. The user quits the program by pressing the "end-of-file" key when prompted for a command; some cleanup may need to be performed before the program exits. (You may choose what you want to display for a prompt but it must be displayed to standard error.)

User input is always read a line at a time &, depending on what we are trying to read, only the first word or the first & second words (if we are reading a name) of the line are used. Extra words are silently ignored.

The following are brief explanations of the valid commands:

n (a positive integer) display record number n. (Note: record 1 is the first record in the file.)

0 append a record. (This will prompt the user to input data for a record. See section 4.)

-n (a negative integer) display all records starting from record number n.

If the user enters an invalid command (this basically means that the user does not enter a number), it is silently ignored (i.e., with no error message) & the user prompted for another command.

4 Appending Records

Appending a record will require getting user input for a student. The data for a student (a student record) consists of an ID, a name & a score. The program prompts for each of these in turn &, as mentioned above, user input is always read a line at a time.

- A valid ID must be a word consisting of a lowercase `a` followed by 8 digits.
- A name consists of a first name & a last name. When prompted for a name, the user is expected to enter a first name followed by a last name (on the same line). A valid first name must have at least 2 characters & fewer than 20 characters; each character must be either an alphabet or a hyphen & the name cannot start or end with a hyphen. The same requirements apply to a valid last name. For example, `Homer` & `Homer-J` are valid first names whereas `D'Arcy`, `B`, `-homer` & `bart-` are not.
- A valid score must be an integer between 0 & 100 inclusive (e.g., 65).

The program needs to obtain the ID, the name & the score from the user & write them to the data file.

To get these data for a student, the program first repeatedly prompts the user for an ID until either

1. a valid ID is obtained from the user, or

WINTER '17

2. the user presses the end-of-file key (at the start of a line), or

3. the user enters a blank line, i.e., a line consisting entirely of "whitespaces" & has no words.

Cases 2 & 3 signify that the user wants to abort the append record operation. When either happens, the program goes back to prompting the user for a command as described in the previous section.

After obtaining a valid ID, the program does the same thing for a name & then the same thing for a score. Again, for either item, the user can abort the append record operation by either

- pressing the end-of-file key (at the beginning of a line), or
- entering a blank line (i.e., a line consisting entirely of "whitespaces" & has no words)

when prompted. Otherwise, the program keeps on prompting until it gets a valid data item: a valid first & last name for the name, a valid score for the score.

Note that if the user enters an invalid ID or an invalid name or an invalid score, no error message is displayed; the user is simply prompted to re-enter that item.

If a valid ID, name & score are obtained, they are appended to the file. After writing the record, the program goes back to prompting the user for commands as described in the previous section.

When writing to the data file, the ID is followed by the first name, then the last names & then the score. The ID is written in a width of 10, the first name & last name are each in a width of 20 & the score is in a width of 4. All are left-adjusted & padded by spaces. Furthermore, both the first & the last name are written in all lowercase.

In the following, the first line shows a record in the file & is intended to show the file format (note: ␣ denotes the space character). The second line is not part of the file; it is only there to serve as a "ruler" to make it easier to see the location of each item in the record:

```
a12345678␣apu␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣nahasapeemapetilon␣␣97␣␣
12345678901234567890123456789012345678901234567890123456789012345678901234
```

The record is immediately followed by the next record (if any). There is no newline or other special separator between records. Note that each record takes up the same number of characters (54) so that we can "seek" within the file.

5 Displaying Records

This always results in output: the number of records displayed (which could be 0) is always printed (to standard output) after the command is executed (i.e., after the matching records, if any, are displayed).

Records are always displayed to standard output. If the user enters a positive integer n as a command, then the n-th record (if there is one) is displayed. (See example below.) If the user enters a negative integer n, then all records starting from the $|n|$-th record (if any) are displayed. (Note: $|n|$ is the absolute value of n.) After displaying the records, the number of records displayed is printed. If there is no record to display (e.g., the file has too few records), 0 is printed.

Assume that the file consists of the record shown in the previous section followed by 2 other records (i.e., the file contains 3 records). The following is part of a possible sample session:

```
> 1    # display record 1
a12345678 : nahasapeemapetilon, apu : 97
1
> -1   # display all records
a12345678 : nahasapeemapetilon, apu : 97
a23456789 : simpson, homer : 25
a33333333 : flanders, ned : 88
3
> hello  # invalid command (note: no output)
> -4     # no records displayed (file has only 3 records)
0
> -2xyz  # for simplicity, just use -2; display all records starting from record 2
a23456789 : simpson, homer : 25
a33333333 : flanders, ned : 88
2
>
```

WINTER '17

In the above, > is the prompt printed by the program. As mentioned, extra input words are ignored. (They serve as comments in the sample above.)

Note the output format for a record: ID, space, colon, space, last name, comma, space, first name, space, colon, space, score, followed by a newline. Names are displayed in all lowercase (just as they are saved in the file).

6 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 error. Records should always be displayed to standard output. This is to separate the two types of output to facilitate testing. Error messages are also printed to standard error, but note that they are not needed when the user enters invalid data.

You may assume that input lines have fewer than `LINESIZE` characters so that you can use an array of `LINESIZE` characters to store a line. Here `LINESIZE` is a macro with a value of 512.

It is important to 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 corresponding output file exactly.

7 Submission & Grading

Be sure to read this section carefully. There are strict requirements on submission.

This assignment is due at 11pm on Friday, February 17, 2017. Submit a zip file to `ShareIn` in the directory:

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

where `<X>` is your set. Your zip file must be named `<lastname><firstname>_<id>.zip`, where `<lastname>` is your last name, `<firstname>` your first name & `<id>` your student ID. (See below if you need to make multiple submissions.) For example, `SimpsonHomer_a12345678.zip`. Do not use spaces in this file name.

Your zip file must unzip directly to your source file(s) *without creating any directories.* We'll compile your file(s) using

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

after unzipping.

Do not put executables in your zip file. Also, 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 ID, e.g., `SimpsonHomer_a12345678_v2.zip`. If more than one version is submitted, we'll only mark the version with the highest version number. (When submitting multiple versions, there must be exactly one zip file with the highest version number. If it is unclear to us which version to mark, you may fail to get credit for the assignment.)

In this course, we limit ourselves to ANSI C (C89) & its standard C library: you must implement any function that you use & that is not in the standard ANSI C library. In practice, this probably means that you should not use any function that we have not mentioned in class unless you implement them yourself. Furthermore, your program must compile without errors or warnings under `gcc` with the switches mentioned above. If it does not meet these requirements, it is likely that you will receive a score of 0 for the assignment. Otherwise, the grade breakdown is *approximately* as follows:

Design & code clarity (includes indentation)	10%
Command input & validation	10%
Displaying records (must read from file)	30%
Appending records (includes data validation)	40%
General error-handling	10%

Note: In order to test whether your program is appending records correctly, it is necessary that your program be able to display records.

WATERLOO