McGill University
School of Computer Science

COMP-206

Mini Assignment #6

Due: March 22, 2019 on myCourses at 23:30

According to Wikipedia, a comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. A CSV file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

A company has text data that is not CSV; however, they need to convert their data to CSV format.

The original text file looks like the following extract of the first 3 lines:

```
Jake Maria Desmond Ali Maria
25 34 19 42 29
Montreal Wawa Head-Smashed-In-Buffalo-Jump Bangor Saint-Louis-du-Ha-Ha
```

The needed CSV file should look like this:

```
Jake, 25, Montreal
Maria, 34, Wawa
Desmond, 19, Head-Smashed-In-Buffalo-Jump
Ali, 42, Bangor
Maria, 29, Saint-Louis-du-Ha-Ha
```

You need to use modular programming techniques to enhance code readability and team work. Remember that modular programming is [citing Wikipedia again] a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Your mission is to implement a C program that will implement the needed logic to read a text file and convert it to CSV. The program should also be able to read the generated csv file and print its content to the screen.
Let's create a separate module to convert the input file to a csv file, then another module that reads a csv file and print its content to the screen. We should also create a third module that is the main one having the main function. This last module will also implement multiple utility functions to search the csv file, adding a new record to it and also deletes a record from it.
You can follow these steps:

1. Create a C module called **convert_to_csv.c** that implements the following function: **void load_and_convert(const char* filename)**
   Split the code in 2 files:
   - **convert_to_csv.c** that contains the implementation
   - **convert_to_csv.h** that contains the declaration

   The function takes one argument that is the name of the input file to be converted.
   It creates a new csv file called output.csv. Note that **load_and_convert** function is the public interface that will be called by other modules. You may create other private helper functions to help you break down your code into multiple logical smaller functions. These functions however should not be callable by other modules.

2. Create a C module called **read_csv.c** that implements the following function: **void read_csv(const char* csv_filename)**
   The function will read a csv file and print its content to the screen line by line
   Split the code in 2 files:
   - **read_csv.c** that contains the implementation
   - **read_csv.h** that contains the declaration

   Note that **read_csv** function is the public interface that will be called by other modules. You may create other private helper functions to help you break down your code into multiple logical smaller functions. These functions however should not be callable by other modules.

3. Write a third module **main.c** that will contain your main function as described below as well as the following functions:

   1. Write a function that will check whether a name exists in the csv file. If it does exist, print its record to the screen. If the name is repeated multiple times in the csv file, then print to the screen all the records for that name:
      void find_name(const char* csv_filename, const char* name);
      If the name was not found, then print "Name not found" to the screen.

   2. Write a function that will add a new record to the end of the csv file. The signature of the function will be:
      void add_record(const char* csv_filename, const char* name, const int age, const char* city);

   3. Write a function that will delete a record from the csv file. The signature of the function will be:
      void delete_record(const char* csv_filename, const char* name);
      delete only the first occurence of name from the csv file.

The main() function that will be used to test your functions is provided as follow, please use it as the main function of your code:

```
int main()
{
  /* Question 1 */
  load_and_convert("input.txt");

  /* Question 2 */
  read_csv("output.csv");

  /* Question 3.1 */
  find_name("output.csv", "Maria");

  find_name("output.csv", "Jason"); //Jason doesn't exist

  /* Question 3.2 */
  add_record("output.csv", "Jason", 36, "Skookumchuk");
  read_csv("output.csv"); // to print to the screen

  /* Question 3.3 */
  delete_record("output.csv", "Maria");
  read_csv("output.csv"); // to print to the screen

  return 0;
}
```

WHAT TO HAND IN

Everything must be submitted to My Courses before the due date. Remember that you can hand in your assignment up to two days late but there will be a penalty of 5% each day. After that, your assignment will not be accepted. Please hand in the following:

- A single zip file having:
  - **all your c and h files**
  - **a sample input file having few lines (copy/paste from the sample that was being provided. Feel free to add more lines to it if needed)**
  - **a shell script tester.sh that compiles all your c files and runs the executable**

HOW IT WILL BE GRADED

The assignment is worth a total of 20 points.

- 20 points are distributed as follow:
  - 1 point for main function having the proper “include files”
  - 4 points for load_and_convert function
    - 2 points for reading properly the input file
    - 2 points for writing properly the output csv file
  - 3 points for read_csv function
    - 2 points for reading properly the csv file line by line
    - 1 point for printing the lines one after the other
  - 4 points for find_name function
    - 2 points for finding all the occurences
    - 1 point for printing the records to the screen
    - 1 point for detecting if the record does not exist
  - 4 points for add_record function
  - 4 points for delete_record function

Please note that the shell script will not be graded, however a penalty of -3 points will be applied to the overall grade if the shell script does not run properly.

GRADING RULES

The following rules are followed by the TA when grading assignments:

- A program must run in order to get a grade (even if it does not run well). If it does not run (does not compile) it will receive a zero. (Make sure to run your programs from Trottier – they sometimes do not run the same from home when logging in using putty.)
- The TA will grade using the mimi.cs.mcgill.ca server.
- All questions are graded proportionally (assuming the program runs at all). This means that if 40% of the question is correct, you will receive 40% of the grade.
- The TA will run the script and observe the execution of your program.
- The TA will check whether the csv file is created and the content is printed out to the screen properly.

- The TA will check whether the find_name function is behaving properly under all possible use cases.
- The TA will check whether the delete_record function is behaving properly under all possible use cases.
- The TA will check whether the add_record function is behaving properly.
- The TA is free to modify the input file and check whether your code is able to deal with any input file that respects the same pattern as the sample provided.