Project 7, Program Design

You are given a dataset about some all-wheel-drive cars and how much fuel they use. This data file is a subset of a public available CSV file: https://corgis-edu.github.io/corgis/csv/cars/. It allows consumers to directly compare cars.

Write a program to find the top 10 cars with highest city mpg. Write the results in the output file.

The input file is a CSV file with the following fields for each car:

Identification (String), Classification (String), Engine Type (String), Transmission (String), City mpg(Integer), Fuel Type (String), Highway mpg(integer). Fields are separated by comma, with each car on a separate line:

2009 Audi A4 Sedan 3.2, Automatic transmission, Audi 3.2L 6 cylinder 265hp 243ft-lbs, 6 Speed Automatic Select Shift, 17, Gasoline, 26

....

Classification: whether this is a "Manual transmission" or an "Automatic transmission". If it is unknown, it is left blank.

Identification: a unique ID for this particular car, using the year, make, model, and transmission type.

Example input/output:

```
Enter the file name: cars.csv
Output file name:
Top10_city_mpg.csv
```

Technical requirement:

1. Name your program *cars_mpg.c*.
2. The output files should be the named `Top10_city_mpg.csv`.
3. Assume that there are no more than 6000 cars in the file. Assume that each field is no more than 150 characters.
4. Use fscanf and fprintf to read and write data. To read all fields of a car, use the following conversion specifier for fscanf:

```
"%[^,], %[^,], %[^,], %[^,], %d, %[^,], %d\n"
```

5. The program should be built around an array of `car` structures, with each `car` containing information of engine type, transmission, city mpg, fuel type, highway mpg, classification, and identification.

6. Your program should include a function that sorts the cars by city mpg. You can use any sorting algorithms such as selection sort and insertion sort. **Note that with different sorting algorithms, the result might differ when city mpg are the same.**

```
void sort_city_mpg(struct car list[], int n);
```

7. Output file should be in the same format as the input file, with the members separated by comma and each car on a separate line.

2012 Subaru Impreza Hatchback 2.0i, Manual transmission, Subaru 2.0L 4 Cylinder 148 hp 145 ft-lbs, 5 Speed Manual, 25, Gasoline, 34

.....

Before you submit:

1. Compile with –Wall. Be sure it compiles on ***student cluster*** with no errors and no warnings.

gcc –Wall cars.c

2. Test your program with the script.

chmod +x try_cars

./try_cars

Note that with different sorting algorithms, your result might be different from the expected output when city mpgs are the same.

3. Submit both cars.c and cars.csv (for grading purposes).

Grading:

Total points: 100

1. A program that does not compile will result in a zero.
2. Runtime error and compilation warning 5%
3. Commenting and style 15%
4. Functionality 80% (functions were declared and implemented as required)

Programming Style Guidelines

The major purpose of programming style guidelines is to make programs easy to read and understand. Good programming style helps make it possible for a person knowledgeable in the application area to quickly read a program and understand how it works.

1. Your program should begin with a comment that briefly summarizes what it does. This comment should also include your **name**.
2. In most cases, a function should have a brief comment above its definition describing what it does. Other than that, comments should be written only *needed* in order for a reader to understand what is happening.
3. Information to include in the comment for a function: name of the function, purpose of the function, meaning of each parameter, description of return value (if any), description of side effects (if any, such as modifying external variables)
4. Variable names and function names should be sufficiently descriptive that a knowledgeable reader can easily understand what the variable means and what the function does. If this is not possible, comments should be added to make the meaning clear.
5. Use consistent indentation to emphasize block structure.
6. Full line comments inside function bodies should conform to the indentation of the code where they appear.
7. Macro definitions (#define) should be used for defining symbolic names for numeric constants. For example: `#define PI 3.141592`
8. Use names of moderate length for variables. Most names should be between 2 and 12 letters long.
9. Use underscores to make compound names easier to read: `tot_vol` or `total_volumn` is clearer than totalvolumn.