Project 7, Program Design

You are given a summarized dataset about state demographics in the united states. This data file is a subset of a public available CSV file: https://corgis-edu.github.io/corgis/csv/state_demographics/.

Write a program to sort the states by percentage of population whose ages are equal or greater than 65 years old. Write the results in the output file.

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

state (String), 2020 population (int), 2010 population (int), percentage of population whose ages are under 5 years old (double), percentage of population whose ages are under 18 years old (double), percentage of population whose ages are equal or greater than 65 years old (double),

with each state on a separate line:

Connecticut, 3605944, 3574097, 5.1, 20.4, 17.7

….

Example input/output:

```
Enter the file name: state_demographics.csv
Output file name:
sorted_state_demographics.csv
```

Technical requirement:

1. Name your program *`project7_demographics.c`*.

2. The output file should be the same name as the input file name with added prefix of `sorted_`. For example, if the original file name is `states.csv`, the output file name is then `sorted_states.csv`. The output file name should not be hard coded.

3. Assume that there are no more than 100 states in the file. Assume that each state is no more than 150 characters.

4. Do NOT hard code the number of states in your program. Your program needs to keep track of the number of states in the file as it reads from it.

5. 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, %lf, %lf, %lf\n"
```

6. The program should be built around an array of `state` structures, with each `state` containing state name, 2020 population, 2010 population, percentage of population whose ages are under 5 years old, percentage of population whose ages are under 18 years old, percentage of population whose ages are equal or greater than 65 years old.

7. Your program should include a function that sorts the states by percentage of population whose ages are equal or greater than 65 years old. You can use any sorting algorithms such as selection sort and insertion sort. **Note that with different sorting algorithms, the result might differ when percentage of population whose ages are equal or greater than 65 years old are the same.**

```
void sort_states(struct state list[], int n);
```

8. Output file should be in similar format as the input file, with the members separated by comma and each state on a separate line, and 1 decimal digit for the doubles.

   Maine,1362359,1328361,4.7,18.5, 21.2

   …..

Before you submit:

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

gcc –Wall project7_demographics.c

2. Test your program with the script.

chmod +x try_project7_demographics

./try_project7_demographics

3. Download *project7_demographics.c* and submit on Canvas>Assignments.

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.