Problem Set 4: Structures, Sorting and Word Count

Please send back via NYU Classes

- A zip archive named as
  `PS04_<your name as FirstLast>.zip`

containing the C code files for both problems.

Total points: 100

Problem 1: Sort an array of structures (80 points)

`sort_struct.c`

Download the file **PS04_Files.zip** from NYU Classes. Using the instructor-provided file **sort_struct.c**, create a program that:

- Has 3 command line arguments, in the following order:
  - Sort-on field name as the literals: first | last | id
  - *File1*
  - *File2*
- Opens *file1* for reading. *File1* is given to you, **name_id.csv**, and each line in the file has three fields separated by a comma and terminated by '\n':

  ```
  firstName,lastName,idNumber
  ```

  where **firstName** and **lastName** are alpha characters and **idNumber** is numeric.
- Reads all lines in *file1* and uses each line to fill the array struct **Student[N]**.
  - use **fgets()** to read a line from the file.
  - use **strtok()** to parse the line into first, last and id.
  - since **fgets()** includes the '\n' in each line of the file, use " \n" (space and newline) as the delimiter string in **strtok()**.
- Using **qsort()**, sorts the Student array according to the sort-on field.
  - Use **switch()** to determine which sort-on you are using.
  - Create a separate **compare()** function for each of first, last and id.
- Opens *file2* for writing and write the sorted student data to *file2*.
  - Write as: `firstName,lastName,idNumber`
  - Use **fprintf(fp, ...)**.

Points are awarded based on (1) command line parsing and usage statement, (2) opening files and reading data into an array of structures, (3) sorting on first name, last name, and id, and (4) using clear code and sensible formatting.

Problem 2: Word count (20 points)

`word_count.c`
Create a program **word_count.c** that counts the number of words in the supplied text file **clear.txt**. Your program must have command-line arguments as follows:

```
./word_count input_file.txt
```

Be sure to have a usage() string or function that prints out usage instructions if, for example, only one arg is present in the command line.

Your program should use functions **fgets()** and **strtok()** to read a line of text and split it into words. Count the words until reading reaches the end of file and **fgets()** returns NULL. Print the total word count as:

```
File <filename> contains <N> words.
```

Hint: since **fgets()** includes the **'\n'** line termination character, use **" \n"** (i.e. space and newline) as the delimiter string in **strtok().**