Project 7, Program Design

A school teacher has a list of leftover school supplies from previous semesters. The teacher would like to maintain an ordered list of the supplies. Write a program supply_list.c that reads in the data (a list of supplies in random order) from a file, and order the items by name, and write the ordered list to the output file.

Assume the input file has the format of color (one word string), name, followed by quantity (int) with each type of supply on a separate line:

```
blue 16 crayon

purple 20 #2 pencil

red 6 pencil box

orange 23  glue sticks
```

…

1. The user will be asked to enter the input file name. The list of sorted supply list should be written to the same file name as the input file name with "sorted_" added at the beginning. For example, if the original file name is `Mrs_Olson_supply.txt`, the output file name is then `sorted_Mrs_Olson_supply.txt`.
2. Read the data using fscanf function. Assume color is one word. To read the name (last item in a line), use "%[^\n]\n" as the conversion specifier for fscanf function.
3. Define a structure `supply` to store the color (string), quantity(integer), and name (string). Assume the color and name are no more than 100 characters.
4. Build an array of `supply` structures. Assume that there are no more than 200 supply in the list.
5. Modify the selection_sort function provided to sort an array of supplies. The supply list should be **sorted by name using strcmp** in ascending order. For example, crayon should be before glue sticks in the list.The function should have the following prototype:

   ```
   void selection_sort(struct supply list[], int n);
   ```

6. The output file should include all supply in the same format as the input file but in sorted order by name.

Before you submit:

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

gcc –Wall supply_list.c

2. Be sure your Unix source file is read & write protected. Change Unix file permission on Unix:

chmod 600 supply_list.c

3. Test your program.

chmod +x try_supply

./try_supply

4. Submit supply_list.c and supply.txt (for grading purposes).

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%

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.