Project 9, Program Design

Write a program that reads file `boxes.txt` and sort the boxes by volume. Output the sorted boxes, including volumes in a text file called `sorted_boxes.txt`. A box has the following attributes:

- Length inches double
- Width inches double
- Height inches double
- Weight pounds double

You can assume the file as the following format for each box.

```
12.0, 2.5, 5.6, 23.4
```

1. The program should be built around an array of structures, with each structure containing information of a box's length, width, height, **volume**, and weight. Assume that there are no more than 100 boxes.
2. Use fscanf and fprintf to read and write data.
3. Modify the selection_sort function provided to sort an array of box struct. The boxes should be **sorted by volume**. The function should have the following prototype:

```
void selection_sort(struct box myBoxes[], int n);
```

4. Output the sorted boxes, including volumes, in a text file, in the following format.

```
#     Length     Width      Height     Volume          Weight
0     1.000000   1.000000   1.000000   1.000000   100.000000
1     3.500000   3.500000   2.500000   30.625000  15.800000
2     48.000000  36.000000  1.000000   1728.000000     2.900000
3     12.000000  12.000000  12.000000  1728.000000     15.000000
4     18.200000  18.200000  14.200000  4703.608000     25.900000
5     22.700000  22.800000  10.600000  5486.136000     4.500000
```

Suggestions:

1. Set up box struct.
2. Use fscanf function to read the input file (note that fscanf returns number of entries filled).
3. Initially output unsorted array to screen.
4. Modify the selection_sort function for processing boxes.
5. Initially output sorted array to the screen.
6. When output is correct, write to the output file.

Extra Credit (20 points)

The file boxes_with_contents.txt also contains the content of each box. Write a program that also store the content in the array of structures besides the length, width, height, volume, and weight. Assume the file as the following format for each box.

```
12.0, 12.0, 12.0, 15.0, Fresh Apples
```

Assume that box's content name is no more than 100 characters long.

Your program should sort the boxes by volume and output the sorted boxes in the following format:

| # | Length | Width | Height | Volume | Weight | Contents |
|---|---|---|---|---|---|---|
| 0 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 100.000000 | Kryptonite Pellets |
| 1 | 3.500000 | 3.500000 | 2.500000 | 30.625000 | 15.800000 | Lead fishing weights |
| 2 | 48.000000 | 36.000000 | 1.000000 | 1728.000000 | 2.900000 | Movie Posters |
| 3 | 12.000000 | 12.000000 | 12.000000 | 1728.000000 | 15.000000 | Fresh Apples |
| 4 | 18.200000 | 18.200000 | 14.200000 | 4703.608000 | 25.900000 | Fine German Wine |
| 5 | 22.700000 | 22.800000 | 10.600000 | 5486.136000 | 4.500000 | Styrofoam Peanuts |

**Before you submit:**

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

*gcc –Wall sort_boxes.c*

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

*chmod 600 sort_boxes.c*

3. Submit both sort_boxes.c and boxes.txt (for grading purposes).

4. If you choose to do the extra credit problem, please submit a separate program: sort_boxes_with_contents.c and boxes_with_contents.txt.

Total points: 100 + 20(extra credit)

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.