Project 9, Program Design

1. (100 points) The daily sales of a grocery store are saved in the file `sale.txt`. Each line contains the record for a product name, price per pound in dollars, and total number of pounds sold. Write a program that reads file `sales.txt`, store the data in an array of product structures, and sort the products by sale volume. Output the sorted products, including sale volumes in a text file called `sorted_products.txt`. A product structure should have the following attributes:

- name string
- unit price double
- total number of pounds sold double
- sale volume double

You can assume the `sales.txt` as the following format for each product: product name (one single word), unit price, total numbers of pounds sold.

```
apple, 2.5, 421.4
```

1. Name your program product.c.
2. The program should be built around an array of structures, with each structure containing information of a product's name, unit price, total numbers of pounds sold, **sale volume (unit price * total numbers of pounds sold)**. Assume that there are no more than 1000 products in the file.
3. Use fscanf and fprintf to read and write data.
4. Modify the selection_sort function provided to sort an array of product struct. The boxes should be **sorted by sale volume** in ascending order. The function should have the following prototype:

```
void selection_sort(struct product products[], int n);
```

5. Output the sorted products, including sale volumes, in a text file called `sorted_products.txt`, in the following format.

#name unit price ($) units (pound) sold sale volume ($)

celery 2.300000 99.450000 228.735000

blueberry 3.500000 87.300000 305.550000

potato 0.600000 823.650000 494.190000

…

2. Extra credit (20 points) Modify the program (Name your program **product2.c**) so it takes command line argument for the product name and the program displays unit price, total numbers of pounds sold, and sale volume of the product. The program should include the following function:

```
void find_product(struct product products[], int n, char *name);
```

The `find_player` function should find the product by name, print the product's unit price, total numbers of pounds sold, and sale volume with two decimal digits.

```
Example run:

./a.out grape

Output:

Name: grape

Unit price: 2.25

Number of pounds sold: 393.50

Sale volume: 885.38
```

Before you submit:

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

gcc –Wall product.c

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

chmod 600 product.c

3. Submit both product.c and sale.txt (for grading purposes).

4. If you choose to do the extra credit problem, please submit a separate program:

product2.c

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% (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.