Project 7, Program Design

Consider some data gathered in a file for general pain medicine and allergy medicine for a local drugstore in a small town for a six month period:

Name	*InStock*	*Jan*	*Feb*	*Mar*	*Apr*	*May*	*Jun*
Acetaminophen	310	83	183	141	98	193	92
Mortrin	275	46	82	103	108	95	121

…

The numbers represents the number of units in stock at the drugstore, and units sold for the six months starting January. The file has the following format:

Acetaminophen 310 83 183 141 98 193 92

…

Write a program that can read in the data from a file and sort the medicine by total units sold for the six months. The sorted data should be written the same file name as the input file name with added extension of `.srt`. For example, if the original file name is `pain_allergy.txt`, the output file name is then `pain_allergy.txt.srt`.

1. Define a structure `medicine` to store the name (string), unitsInStock (integer), and six month's unitsSold as an array of int, and total units sold (int). Assume the name is no more than 100 characters. Assume the name is one word name.
2. Build an array of medicine structures. Assume that there are no more than 100 medicines.
3. Modify the selection_sort function to sort an array of medicines. The medicines should be sorted by total units sold. The function should have the following prototype:

```
void selection_sort(struct medicine meds[], int n);
```

4. The output file should include the total units sold as the following.

#	Name	InStock	Jan	Feb	Mar	Apr	May	Jun	Total
1	Aleve	177	23	53	99	75	35	86	371
2	Zyrtec	675	111	43	88	109	45	32	428
3	Claritin	349	112	123	98	93	43	65	534
4	Mortrin	275	46	82	103	108	95	121	555
5	Benadryl	477	201	123	98	65	43	50	580
6	Ibuprofen	873	31	134	98	103	210	45	621
7	Aspirin	633	87	100	121	99	176	97	680
8	Acetaminophen	310	83	183	141	98	193	92	790
9	Allegra	783	182	239	122	83	92	76	794

Before you submit:

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

gcc –Wall sort_medicine.c

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

chmod 600 sort_medicine.c

3. Submit sort_medicine.c and pain_allergy.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.