CST8234 – C Programming W18 (Lab 4)

Programming Exercise

The purpose of this lab is to

- Demonstrate that you can create and manipulate structures, using pointers
- Demonstrate that you can sort an array of structures, using appropriate comparison functions

Statement of Problem

Write a program (e.g., 'lab4.c') that generates a sorted list of rental properties.

Each property will be implemented as a structure that has members that record the street name, the street number, the number of bedrooms, and the monthly rent.

Using a typedef to simplify naming your struct is a good idea, but not mandatory.

You will initialize a 12-element array of these structs. You will provide your own (fictitious) street names and numbers, but will randomly generate the number of bedrooms (1-4) and the rent (800-1200, in one hundred dollar increments). The random numbers must vary each time you run the program. Three of your rental properties must use *exactly* these addresses ("24 Sussex Drive", "1 Sussex Drive", and "5 Sussex Drive"), and two of them must exist on another street. All the other properties can exist on their own streets.

You'll need a function to print a summary of each rental property, which will be used to print out lists. This function should use fixed-width columns (as per earlier labs) to make the list is easy to read (i.e., by having the rents for each property all aligned in a single column, ditto for the number of bedrooms).

You will then print out the list three times, with the following sorting criteria:

1. Sorted by number of bedrooms, increasing. If there are two or more rental properties with the same number of bedrooms, then those properties will be ordered by the monthly rent, increasing.
2. Sorted by rent, increasing. If there are two or more rental properties with the same monthly rent, then those properties will be ordered by the number of bedrooms, increasing.
3. Sorted by street address, increasing. The sorting is by street name first, then street number (e.g., "5 Sussex Drive" and "24 Sussex Drive" come before "1 Wellington Avenue")

You will add an appropriate header before each list, as demonstrated in the following sample output

```
$ ./lab4.exe

Sorted by Rooms
Address                          # Rooms     Rent
-------------------------        -------  -------
 77 Frog Court                         1      500
 33 Fish Lane                          1      600
 18 Robin Road                         1      700
  5 Sussex                             1      800
 24 Sussex                             2      400
  1 Sussex                             3     1200
 10 Turtle Boulevard                   4      600
  5 Snake Street                       4      800

Sorted by Rent
Address                          # Rooms     Rent
-------------------------        -------  -------
 24 Sussex                             2      400
 77 Frog Court                         1      500
 33 Fish Lane                          1      600
 10 Turtle Boulevard                   4      600
 18 Robin Road                         1      700
  5 Sussex                             1      800
  5 Snake Street                       4      800
  1 Sussex                             3     1200

Sorted by Address
Address                          # Rooms     Rent
-------------------------        -------  -------
 33 Fish Lane                          1      600
 77 Frog Court                         1      500
 18 Robin Road                         1      700
  5 Snake Street                       4      800
  1 Sussex                             3     1200
  5 Sussex                             1      800
 24 Sussex                             2      400
 10 Turtle Boulevard                   4      600
```

Randomization

To enable the identical sequence of random numbers with each invocation, you'll put the following line of code at the top of your main() function

```
    srand( time(NULL) );
```

You'll need to include the standard library "time.h" in your program.

Sorting

You'll find that a simple way to sort an array of N elements is to implement the following pseudo-code

```
for ( i = 0 .. N-2 )
       for ( j = 0 .. N-i-1 )
              if compare( element j, element j+1 )
                     Swap element j and element j+1
```

What does "compare(element j, element j+1)" mean? That basically means, "should element j be AFTER element j+1", based on a numeric assessment. I.e., we boil each comparison down to a difference between two numbers… and if that value is less than zero, it is assumed that element j should come before element j+1; if the value is greater than zero, it is assumed that element j should come after element j+1 (hence the swap); and if the value is exactly zero, then it doesn't matter which comes first.

This means you're going to need a comparison function to generate that numeric difference, for each sorting criterion. E.g., your compareByRent() function might use the following pseudo-code to generate the positive/zero/negative result required by the bubble sort.

```
int diff = property_A.rent - property_B.rent;
if ( diff == 0 )
       diff = property_A.numRooms - property_B.numRooms
return diff
```

You'll have to pay attention to the sort by street address function to make sure that addresses on the same street are sorted by numerical order, not lexigraphic (a.k.a., dictionary) order. I.e., "24 Sussex Drive" comes after "5 Sussex Drive", not before it.

You'll find you can use the built-in 'strcmp' function (declared in string.h) to compare strings this way.

Special Restrictions

In an effort to force you to pass information via parameters, rather than relying on global variables, you are not permitted to define any global variables.

In an effort to force you to divide your code into logical blocks of code, rather than putting everything in main(), you may not have any functions (including main()) that exceed 25 lines of code (where a line of code is considered to be a definition or statement, including those separated by semicolons. Attempts to jam multiple statements onto a single line will be penalized, as the goal is to come up with multiple logically-sized functions .

Structs must be passed into functions (e.g., your comparison functions, or your swap routine) by ***reference*** (i.e., as pointers) rather than by ***value*** (i.e., as copies of the structs). This means that you'll end up using "->" notation, not the ".". Failure to use pointers will result in deducted marks.

If you want to use multiple source files, you may, but you'll have to edit the 'makefile' to specify a different set of source files.

Submission

When you are complete submit your programs to Blackboard.

But… before you do, please check that you've satisfied the following submission requirements

1. Did you make sure that you have a header comment in your source files?
2. Did you confirm that your zipped submission is a ".zip" file (not a '.rar' or '.tar.gz' or '.7z' file)?
3. Did you zip up a folder that includes your *user name*, and the lab/assignment indicator? I.e., if you open up your own zipped submission, you should see a *folder* called (for example) "smit9112_L4". If you just see *file(s)*, you've done it wrong, and you'll need to go to the *parent* folder, and try zipping your lab folder.
4. Did you remove all the unnecessary files from the folder contained in your zipped submission? I.e., you open up your own zipped submission, and click on the folder called (for example) "smit9112_L4", you should see *just your makefile and your source files (e.g., lab4.c)*.
5. Is your makefile actually called 'makefile' or 'Makefile' (without any filename extensions?)

If you don't satisfy submission requirements #2 or #3 you will get ZERO on this lab. If you realize afterwards that you've made a mistake, don't panic! you are allowed to correct your mistake and re-submit. But you ONLY get TWO submissions per lab/assignment, so try to make sure you double-checked everything *before* doing your first submission… and only use the second submission in case of emergency.