Lab 4
Functions, Arrays and File Handling

Topics

1. Use functions to perform different tasks
2. Work with arrays
3. File Handling - Use file pointers (FILE*) to read in and write out files.

Prelab Assignment

- Read Chapters 4 and 5 (up to and including section 5.5) (Kernighan & Ritchie)

Assignment

Initialize a git repository on your local machine and then start to work on your assignment under that repository.

Part 1: Read data from a file, perform simple operations, save data into another file.

In this part you need to write a program that can access files with some numerical data. The data correspond to a signal that has been sampled. The format of the files is as follows: the first row of the file will have two integers, separated by a space. The first value represents the length of the signal (i.e. the number of sample points). The second value represents the maximum possible value that could have been measured (which may or may not appear in the actual data). Starting at row two, all the way to the end of the file, you have single integer values that correspond to the data samples. Consider the following example:

```
5  15
2
11
0
8
13
```

The example above indicates that the file contains 5 data samples, and the values could be 15 or less (in this case the maximum is not present).

The files will be named: "Raw_data_01.txt", "Raw_data_02.txt", …, "Raw_data_10.txt", "Raw_data_11.txt", …, etc. You can find some example files on Blackboard.

1. Your program should ask the user the number of the file to open (1, 2, etc.). Create a function that opens the corresponding file and stores the data samples into an integer array.
   - How can you open the correct file? That is, how can you use the number entered by the user to "create" the correct file name? **Hint:** consider the function *sprintf()*;
2. Ask the user if he/she wants to offset or scale the original signal. You should ask for the offset value or scaling factor (type **double**).
   - The offset represents a vertical translation of a signal. That is, a constant value will be added to each data sample (think of function transformations in math).
   - Scaling the signal is multiplying it by the scaling factor.

   Hint: What type should the transformed array be?
3. Finally, you should save the transformed signal into a new file. The file should be named "Offset_data_nn.txt" or "Scaled_data_nn.txt", where nn is the same number of the Raw file (two digits format: 01, 02, …, 10, 11, etc.). The file should have the following format: The first row will have the length of the signal (number of samples) and the offset value or scaling factor. The next rows will have the transformed values. All double values should be saved with 4 decimal points. Consider the following examples (left: offset; right: scale):

5 -2.5	5 0.5000
-0.5000	1.0000
8.5000	5.5000
-2.5000	0.0000
5.5000	4.0000
10.5000	6.5000

Part 2: Some more calculations and specific transformations

In this part you will add some functionality to your program. You will calculate some statistics of the input data and you will perform some common transformations used when analyzing signals.

1. Write a function that calculates the mean value (average) of a signal. That is, you should take the data samples from the original (raw) file and calculate their average.

   Hint: Use type **double** for the average.

2. Write a function that finds the maximum value of the signal.
3. Save the above values into a file named "Statistics_data_nn.txt". The format should be as follows: average max_value
   Example: 23.6000 13
4. Centering a signal: a common transformation performed to signals is to center them at zero. This means that the signal should have a zero-mean. Use the functions developed in parts 1 and 2 to center an input (raw) signal. Save the transformed signal into a file named "Centered_data_nn.txt". The format of this file should be the same as the format of the "Offset_data_nn.txt" files.
   Hint: what should the offset be in this case?
5. Normalizing a signal: Another common transformation is to normalize signals. One case of normalization is to scale the original signal so that all the values are between 0 and 1 (or sometimes between -1 and 1). Use the functions developed in part 1 to normalize an input (raw) signal. Save the transformed signal into a file named "Normalized_data_nn.txt". The format of this file should be the same as the format of the "Scaled_data_nn.txt" files.
   Hint: use the maximum possible value that you can obtain from the first row of the raw file. What should the scale factor be in this case?

Push your code to your github repository.

Experiments:

Run various cases of scaling factors and offsets (part 1). Obtain the "Statistics", "Centered" and "Normalized" files for all the raw files posted on Blackboard. Make sure that your results are correct.

Deliverables (due at the beginning of the following week's lab session)

1. Lab Report:

   The lab report should include:

   1) The objective of the lab (written in your own words).
   2) Results: show at least one example output for each one of the "Raw_data" files. Include screenshots and sample output files.
   3) Discussion: Discuss your results. Describe any problems that you may have encountered while coding the lab. How did you fix them? Talk about what you learned, interesting/unexpected things that you observed, problems/limitations that your current approach may have, alternative solutions/approaches, etc. Include any other relevant comments related to the lab.
   4) Screenshot of your github repository showing your commit.
   5) Append your source code with proper comments to the end of you report.

2. Demonstration:

   The TA will ask you to run some examples, to demonstrate that your programs are working properly.

Grading:

Demo:	20
Report:	60
Code:	20 (proper indentation / comments)