Lab 5
Command-line Arguments, Strings and Multi-dimensional Arrays

Topics

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

Prelab Assignment

- Read Chapter 5, up to and including section 5.10. (Kernighan & Ritchie)
- Investigate the functions: atoi(), strtod(). You should bring a piece of paper with a brief description of the functions.

Assignment

This lab is a continuation of lab 4. You should use the repository that you created then.

You should modify your lab 4 programs so that all options and parameters are input through the command line when the program is run. That is, the program won't ask the user to enter the options/parameters during execution, but it will act according to the parameters passed when calling the program. This can be done using arguments for the main function:

```
int main(int argc, char **argv)
```

The options/parameters that need to be handled are:

-n:	File number (value needed)
-o:	offset value (value needed)
-s:	scale factor (value needed)
-S:	Get statistics
-C:	Center the signal
-N:	Normalize signal
-r:	Rename files (name needed)
-h:	Help (display how the program should be called, including the different input options)

Consider the following examples to illustrate how to use the options above:

- If you call the program like this: ./My_Lab5_program –n 3 –o 2.5
  Your program should read the file "Raw_data_03.txt", and it should offset the signal by 2.5, saving the corresponding file "Offset_data_03.txt".
- If you call the program like this: ./My_Lab5_program –s 1.7 –S –N –n 10
  Your program should read the file "Raw_data_10.txt"; it should scale the signal by 1.7 (and save the corresponding file "Scaled_data_10.txt"); it should calculate the statistics of the original signal (and save them to the corresponding file); and it should normalize the original signal (again, saving the corresponding file).
  Notice that the input parameters could be in any order. Also, that some options require a value, and some do not.
- If you call the program like this: ./My_Lab5_program –n 2 –r NewName
  The program will create a copy of the file "Raw_data_02.txt" named "NewName.txt"
- If you call the program like this: ./My_Lab5_program –n 2 –r NewName –C –s 3
  The program will create a copy of the raw file as in the example above; the centered file that needs to be created will be named "NewName_Centered.txt", and the scaled file will be named "NewName_Scaled.txt".
- If you call the program like this: ./My_Lab5_program –h
  Your program should display a message indicating how the program should be called, what the different options are, and some example calls. Then, the program exits.

Your program should be able to handle incorrect or incomplete input. Consider the following examples.

- ./My_Lab5_program
- ./My_Lab5_program –s 1.7 –S –N –n
- ./My_Lab5_program –n 7 -o
- ./My_Lab5_program –C

In all cases above, the program does not have enough information. It should print a message indicating so, and then a message similar to (or the same as) the one displayed for the Help option. Then, the program should exit.

If there is enough information, even if some inputs don't make sense or are incomplete, the program should be able to run, but it should display warning messages. For example:

- ./My_Lab5_program 7 –o –S –N –n 5 –a – C -r
  The 7 is meaningless, there is no value for the offset, the –a option is not implemented, there is a space between the 'dash' and the C, and there is no new file name. The program should recognize that all those cases are not valid, and it should display a warning message (e.g.: "Option xx is not valid"). However, there are some valid inputs, so the program should go ahead and execute those. In this particular example, the program should create the files "Statistics_data_05.txt" and "Normalized_data_05.txt".

<u>Push your code to your github repository.</u>

Experiments:

Run various cases, using different combinations of parameters (correct and incorrect).

Deliverables (Pre-lab due at the beginning of the lab 5 session, the rest is due at the beginning of the following week's lab session):

1. Pre-lab assignment.

2. Lab Report:

   The lab report should include:

   1) The objective of the lab (written in your own words).
   2) Results: show a few examples of the renamed files. Include screenshots of the program being called with the different parameters, as well as the help and error/warning messages.
   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.

3. Demonstration:

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

Grading:

Pre-lab:	5	
Demo:	30	
Report:	50	
Code:	15	(proper indentation / comments)