Thomas Gibbons

October 14, 2016

ECE 3220

LAB 5

Objective

The objective of this lab was to extend what we learned in lab 4. We used functions from the previous lab to perform various tasks like offsetting, scaling, or normalizing data along with extending our knowledge of file handling. We learned to handle command-line arguments with the argc and argv** variables. We handled multi-dimensional arrays like argv** where the first dimension is which string and the next is which character in the array.

Discussion

In this lab there was a lot more argument handling then anything, so I will mostly focus on how I handled all of the error checking of the arguments.

For the first check I see if the number of arguments is one in case they only put "./lab5". Which then shows how to call the help command.

Next, I set up a bunch of variables that were set to zero and will be bumped to signify different tasks. For example I bump one variable if "-s" is a command. This lets me know later after going through the arguments that I should do scaling of some sort.

When I went through all of the arguments, I simply had a count to bump through different string arguments. I would compare it with the different valid arguments in my program and if it was equal I would do various error checking and if still valid bump a variable.

For "-n", "-o", and "-s", the arguments required that it was followed by a number. So to check I compared the first character and made sure its ascii value was between that of '1' and '9'. I did not check the whole number, but it at least made sure it was numerical. For "-0" and "-s", I converted the number from string to double and stored the value. For "-n", I converted the number from string to int and stored the value. I used an error check from last lab where the number after "-n" must not only be a number but be between 0 and 99. When a valid number is found after these arguments the count of arguments is also bumped since I don't need to check if a value will be an argument.

For "-r", the argument also requires a string after. If there is a string after, I go through each character to know its length for later memory allocation. Also bump argument count to avoid going through it. This argument is the only place I have found any sort of fault in my program. I'm not really sure how to deal with it either. An example of such error is "./Lab5 –n 5 –r –s". In this case I would see the “-r” and then make the name of the file “-s”. If the user wanted statistics, it is not going to happen since I bump over it. If I got rid of the bump though, when there is an actual filename it would show up as an invalid argument which it also is not. And it might just be that someone wants to name their file “-s”. Anyways that it is a minor flaw but not really sure how to check whether the user meant to name it that or not.

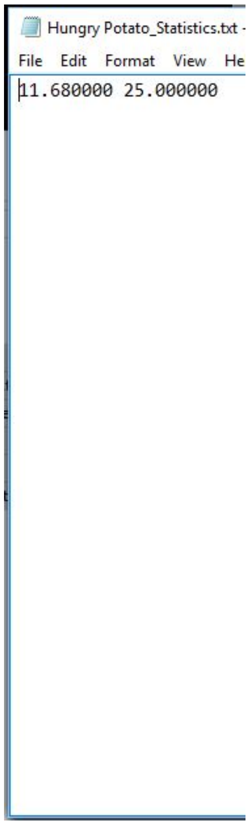
Now all of the arguments have been processed, next I check the most crucial flag. If a file number was not included, then I display error and terminate.

Next I check to make sure there is at least one other argument to take action with or I display error and terminate.

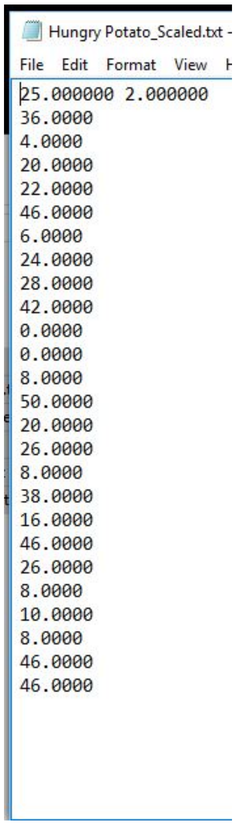
The rest of the program is similar to the last lab except only doing all of the tasks that have been flagged by the argument processing. So if the offset flag is set, offset will happen with the given offset value stored. If there was a rename flag, all of the arguments will save it under the rename or if there was no rename then it proceeds as done in the previous lab.

Finally I freed all the allocated memory including the input file names, output file names, and data arrays. The link to my code on my GitHub account is http://github.com/tgibbons95.

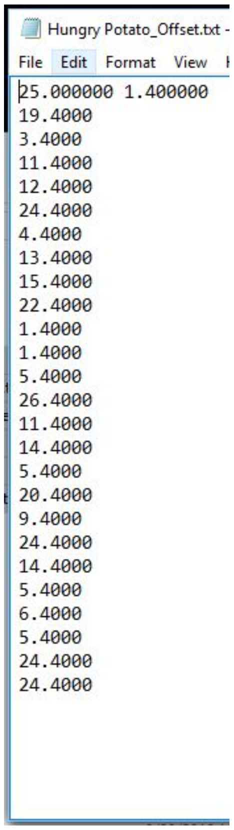
Below are screenshots of my updated files when I do every argument in my program and have the new name set to “Hungry Potato”.



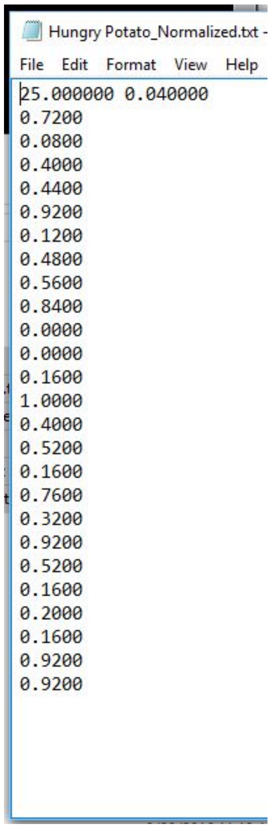
```
11.680000 25.000000
```



```
25.000000 2.000000
36.0000
4.0000
20.0000
22.0000
46.0000
6.0000
24.0000
28.0000
42.0000
0.0000
0.0000
8.0000
50.0000
20.0000
26.0000
8.0000
38.0000
16.0000
46.0000
26.0000
8.0000
10.0000
8.0000
46.0000
46.0000
```



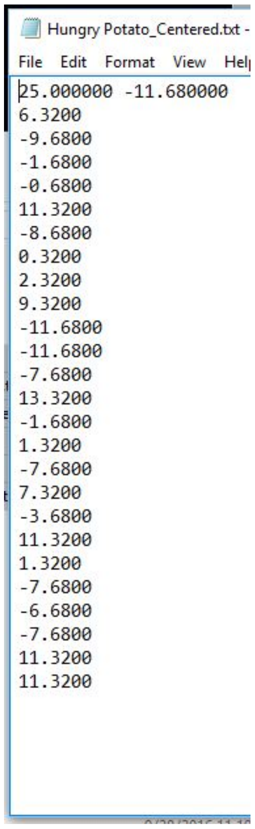
```
25.000000 1.400000
19.4000
3.4000
11.4000
12.4000
24.4000
4.4000
13.4000
15.4000
22.4000
1.4000
1.4000
5.4000
26.4000
11.4000
14.4000
5.4000
20.4000
9.4000
24.4000
14.4000
5.4000
6.4000
5.4000
24.4000
24.4000
```



Hungry Potato_Normalized.txt -

File Edit Format View Help

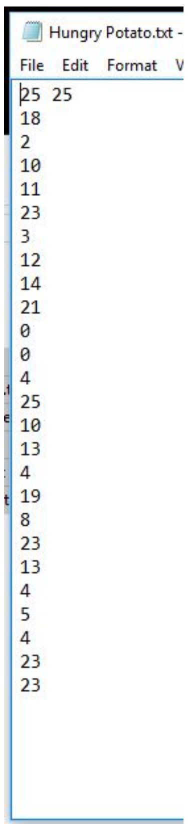
```
25.000000 0.040000
0.7200
0.0800
0.4000
0.4400
0.9200
0.1200
0.4800
0.5600
0.8400
0.0000
0.0000
0.1600
1.0000
0.4000
0.5200
0.1600
0.7600
0.3200
0.9200
0.5200
0.1600
0.2000
0.1600
0.9200
0.9200
```



Hungry Potato_Centered.txt -

File Edit Format View Help

```
25.000000 -11.680000
6.3200
-9.6800
-1.6800
-0.6800
11.3200
-8.6800
0.3200
2.3200
9.3200
-11.6800
-11.6800
-7.6800
13.3200
-1.6800
1.3200
-7.6800
7.3200
-3.6800
11.3200
1.3200
-7.6800
-6.6800
-7.6800
11.3200
11.3200
```



Hungry Potato.txt -

File Edit Format V

```
25 25
18
2
10
11
23
3
12
14
21
0
0
4
25
10
13
4
19
8
23
13
4
5
4
23
23
```