Jay Offerdahl
EECS 678 Lab 3

1. **Program Design:** Briefly describe the design of the program. How many processes and how many pipes are created with your solution? How does data flow from process to process?

My program has four child processes and three pipes in between them. The processes break down as follows: find, xargs grep, sort, head. This accomplishes the task in the most modular and feasible solution.

Data flows from process to process obviously through the pipelines. In the first child process, I redirected STDOUT to the output side of the first pipeline. I then redirected STDIN in the second child process to the input side of the first pipeline. Within the second child process, I redirected STDOUT to the output side of the second pipeline. Next, in the third child process, I redirected STDIN to the input side of the second pipeline, and STDOUT to the output side of the third pipeline. Lastly, I redirected STDIN to the input side of the third pipeline, finishing the chain of pipelines.

2. **Testing:** How did you test and debug your solution?

I initially tested the pipeline by writing a string in the first child, reading that string in the second child, writing it again in the second child and again until I read it for the final time in the fourth child. This allowed me to figure out and grasp the concept of pipelining, in order to progress throughout the rest of the lab.

Next, after adding the execution functions, I ran into several errors because I entered the execution parameters wrong, so I had to consult online documentation to ensure my implementation was correct.

Lastly, I was running into an infinite loop (seemingly) after what I thought was a complete program, so I walked through every line of code multiple times until I realized that I never closed the input/output sides of the pipelines in the parent process, which was causing the error.

3. **How do pipes contribute to the UNIX philosophy?**

Pipelining contributes to the UNIX philosophy because it quite simply allows programs (processes) to be as modular as they can be, so they can accomplish one task and do it well, instead of trying to accomplish several tasks in a poor manner.

Pipelining makes it easy to handle text streams, which is exactly what we did here. So, according to the UNIX philosophy, ensures a universal interface.