Testing:

The testing during this assignment was done periodically throughout the duration of the assignment in a form of print statements and if statements making sure the flow of code is right. The program was first coded to handle small files, this was because of the restriction caused by the buffer, which was reading by one byte at a time. When the code was running right at that point, I just had to adapt it so that it had a longer buffer.

When I completed the bigger buffer size, I used the scripts that were given through the discord. This was where big-sized files were pushed into the program and checked with the head.

After running the scripts, I checked if standard input was working by just typing lines into the terminal. This was where I found a bug with the buffer, it was reading previous enters multiple times, so I used calloc() to clean up the buffer.

After I also tested the code for memory leaks using Valgrind. There were no memory leaks reported.

Question:

How does the code for handling a file differ from that for handling standard input?

There are no major handling differences on how the code handles a file and standard input besides one different distinction. When handling a regular file, open() has to be used to access the file and set the file descriptor. With standard input, on the other hand, this is different the file descriptor is just set to the macro STDIN_FILENO, which is part of unistd.h library. So when calling read(), it does not actually read from a file but from standard input.

What concept is this an example of?

This is an example of the concept of how you can use the same code for other purposes in a way a sense of modularity. This is a very shallow example because it is not that modular but in a way, you could make this modular. The difference is not that big at all. And that could benefit in a way. The fact is that the code does not use an actual file descriptor when using standard input but a number, so the only line that is really different is the read line.