Jordan Mai
1645861

Design Document

1. **Project Description**
    a. Our program called *shoulders* is a program that copies bytes and copy to stdout. The program will take in non empty files to be read in one at a time and copy the given amount of bytes and at most 100 bytes of content into stdout. If no files are given, the program will take in user input at stdin. The program will end once all contents are read in from the command line argument.

2. **Program Logic**
    a. PSEUDOCODE
        i. Read in first command line argument argv[1] to consider number of bytes
        ii. For int i=2 to EOF:
            1. Open the file at argv[i] in the command argument
            2. Read in at max argv[1] bytes from the file
            3. If argument is '-' or empty then read from stdin
            4. Write out the file to stdout
            5. Close file
        iii. If there are any errors in opening or during reading files, end the program and print out an error message

3. **Data Structure**
    a. An array works by allocating space for some bytes of content to be put into. A buffer array is used to store bytes read from a file. This array of characters will then be used to print to stdout.
    b. A file contains amounts of bytes of whatever content. Files will be used for where the program opens and reads bytes from to be copied to stdout.

4. **Functions:**
    a. int main(int argv, char*argv[]):
        i. This function is the main function for the program. It will take in arguments from stdin and read the files or inputs one by one. The program expects inputs of an integer(which determines the number of bytes for the program, and optional files to be read from or '-'. The program will read in each argv file and copy the maximum of bytes specified to stdout. If no file argument is inputted, the program will read from stdin in which the user types in things. This program will use a for loop to iterate through every command argument and open and read from each file to stdout. The output of copied bytes will be to stdout.

5. **Question: How does the code for handling a file differ from that for handling standard input? Describe how this complexity of handling two types of inputs can**

be solved by one of the four ways of managing complexity described in Section 1.3.

a. The code for handling a file will need to have an open command to open the file to read from. Then it needs to close the file at the end. The code for handling standard input will not need the open and close commands because it can just read from whatever the user types in the stdin command line. For a file, the code will need to check for errors in opening and ensuring file works, while for standard input, no error check is needed as it is already processed.
b. Handling two different inputs can possibly lead to large complexity and unnecessary time. This complexity of handing two types of inputs can be solved using the layering to minimize the interaction between two modules. Because handling files is different from handling standard input, we can use the concept of layering to build off of previous modules or code to improve complexity. By using a layering method, we can create a code design that is able to read in both files and stdin simultaneously, but just have the correct error checks for each different input. The code will both accept file input and stdin as part of the layering complexity, and won't need separate code methods.