Assignment 0 Design

I. Introduction

A. Objective

The goal of this assignment is to implement a program, `dog`, that performs the basic functionalities of `cat`.

B. Statement of scope

`dog` accepts any arbitrary (nonnegative) number of filenames and a dash ("-") as inputs and processes them one at a time. The program must be able to handle both text and binary files and does not support any flags. The program will handle dashes as standard input and print to standard output. It will process file arguments in the order they're given.

II. Program Design

1. The program makes note of how many arguments are specified in the command line.

2. If there is only one argument ("./dog" itself), the program will copy standard in to standard out until there are no more input arguments left.

3. If there is more than one argument, the program will loop through and open() all the arguments in the order they were given in.

- **a.** If the argument is a dash, standard in will be copied to standard out until there are no more inputs left.
- **b.** If the argument is an invalid file, print an error message and continue.
- **c.** If the argument is a filename that represents an existing file, read and copy all its contents into standard out.

III. Restrictions

The program will not utilize any of the C library FILE* functions. The system calls: open(2), close(2), read(2), write(2), and warn(2) are used for I/O instead. Buffer size must be fixed and must not be allocated more than 32 KiB of memory.

IV. Testing

A. Classes of Tests

White-box testing will be utilized while testing the functionality of the program. Although the use of C library FILE* functions are restricted for the functionality of the program, they will be used during testing as a convenient way to debug and keep track of values such as loop iterations and filenames. These print statements will be removed before submission to ensure clean and readable code. The program will be incrementally developed, so these tests are performed at every stage. These tests are described as follows:

- **a.** While developing the `cat` functionality, the inputs are printed to stdout to ensure that the args were read and processed correctly.
- **b.** The loop increments and filenames are printed to ensure that the args were correctly processed in order and that each line was completely copied before incrementing to the next line.

Black-box testing will also be utilized through manual testing. The code will be tested using a variety of different combinations of inputs. The test inputs include the following:

a. No additional arguments followed by text and binary ("`./dog [stdin inputs]`")
b. One dash ("`./dog -`")
c. An arbitrary amount of non-existing files
d. An arbitrary amount of existing text filenames (e.g. "`./dog file1 file2`")
e. Arbitrary amounts of binary filenames (e.g. "`./dog file1.bin`")
f. Any combination of binary files and text files (i.e. "`./dog file1.txt - file2.bin file3.txt`")

B. Expected Output

`dog` will copy data from the files specified in standard input, in order, to standard output. If `dog` runs into any errors with a file, an error message will be printed before the program continues to handle the rest of the files. If either no files are specified or "-" (dash) is specified as a filename, the program will copy standard input into standard output until there are no more inputs (e.g. ^D is detected). If an error message is printed at any point, it will be reflected in the return value.

V. Assignment Question

- **How does the code for handling a file differ from that for handling standard input?** The code for handling a file requires the file to first be opened before it can be read and parsed into a buffer. The buffer will then be printed to stdout. Standard input can be directly printed to standard out through the use of user input (arg) command which prints out user input until the end of line statement is reached (^D). This complexity of handling two types of inputs can be solved using modularity. When the program determines what it will be reading from, it will only use the respective module/method.