Design Document

Author: Miguel Avalos

Student ID: 1704078

Project Description

shoulders prints the first characters up to X size from files or standard input (no spaces or newlines after printing if there are no errors). Number X is the first argument and must be greater than or equal to zero. The arguments after are file names or '-' which indicates standard input. If no file names or '-' are given as an argument, the first X characters from standard input will be printed.

Program logic

shoulders will iterate with for every argument after the size argument. Files or standard input will be read partially onto a buffer. The buffer will then be fully or partially written to standard output depending on the remaining amount of characters left to print to complete X characters. This process is repeated until X amount of characters is printed for each file or standard input indicator '-'. If no filenames or standard input indicators are provided, **shoulders** will perform the procedure above only on standard input.

Data Structures

A character array will be used as a buffer to store data read from standard input or files. This buffer will later be fully or partially printed depending on the context.

Functions

- int headprint(int source, int size)
  This function will print the first characters up to **size (**or the entire source if size is greater than source size) from the provided **source** (a file descriptor). If any error occurs, it will be printed in place of normal output. NOTE: If source is standard input, then headprint will print read error messages for it. Otherwise for files, headprint will rely on filesource function to carry out the message.
    1. Create a small buffer
    2. The following will be done in a loop until **size** characters are printed or the last read from the source does not read any characters. The second condition will be ignored for the first check since no characters have been read yet.
        a. The **source** input will be read partially (or fully if buffer size is greater or equal to **source** size) to a buffer, program checks if there are no errors reading.
        b. Write buffer contents up to the length that the last line was able to read, program checks if there are no errors writing to standard out.
        c. Subtract size by characters that were read from the file.

- int filesource(char *filename, int size)
  This function will print the first characters up to **size (**or the entire file if size is greater) from a file stored in the directory. If any error occurs, it will be printed in place of normal output.
    1. The file will be opened using **filename**; error checking would be if a file is not in the directory, the program lacks access permissions, and others.
    2. Call headprint with file descriptor received from the last line and **size**.
    3. If headprint call returns 2, print read error message with **filename.**
    4. Close file, program checks errors in closing file.

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 Section1.3. (This topic will be covered at the end of week 1 or early week 2, but you can start earlier and answer this question by reading section 1.3.)

  The difference between these two options is that a file must have a filename provided, has to be opened before data is read, and has to be closed after the required actions are done. Standard input doesn't need this. Also, print associated errors for those steps specifically to what source its using.

  Using layering, we can add a module layer to be used for file sources to run code and print errors than a standard input source doesn't need before and after reading and writing source data.