CSCI 330 ASSIGNMENT 3

IMPLEMENT `cat` COMMAND (50PTS)

PURPOSE

The purpose of this assignment is to provide practice using the system calls for working with files on a UNIX system. You will be writing a basic implementation of the `cat` command using C++.

DESCRIPTION

As you should recall, the `cat` command takes a list of files as command line arguments. It then opens each file in turn, writing each file's entire contents to standard output in the order they were supplied.

You will be responsible for writing a C++ program that implements this behavior. If you are unclear on what your output should look like, compare your program's output to the output from the actual `cat` program. As seen in its manpage (`man cat`), the actual command does have some additional options that can be used, but other than the single dash, "-", for standard input, you don't need to implement those.

REQUIREMENTS

1. The user can pass as many filenames as they wish as command line arguments. Your program must be able to handle all of the files, no matter how many are passed.
2. No matter how long each file is, your program must be able to read and output all of the data it contains. This may be tested with a file that is larger than all of the RAM available to you, so you need to find a way to print its contents anyway.
3. All of the data must be displayed, even if the files contents have non-text data. Notice that this means `cout <<` will not be sufficient on its own.
4. The UNIX system calls that we spoke about in class must be used to implement the reading portion. They may be used for the writing portion as well, but this is not required for *this* assignment. This means that the C++ file stream classes are **not** permitted to be used for the file input. (Use `open`, `read`, `close`.)
5. If "-" (a single dash) on its own is specified as one of the files to read and output, read the data from *standard input* instead of opening a file. Do not open a file called "-".
6. Make sure to clean up after you are done with each file that you have opened, calling `close` on its file descriptor. Remember that no file was opened in the case of "-" from above.
7. Make sure to properly document your code. This includes a documentation box at the top of your program and before each function you write. You must also include comments on the portions of the code that do important work in order to explain what is going on, and why. If documentation is missing. Your grade on the assignment will suffer.
8. Your code should compile properly with just `g++` and no special flags on `turing` and/or `hopper`.

WHAT TO TURN IN?

All you need to turn in for this assignment is the C++ source code file. Submit it via Blackboard. It is allowed to have any of the file extensions appropriate to a C++ source code file (`.C`, `.cc`, `.cpp`, etc.), but I recommend "`.cc`".