Problem 1 -- Fixing your cat

Revisit your concatenation program from Unit 1. Make sure to correct any mistakes with error handling/reporting and partial write handling. Then modify it to accept only the following syntax:

```
catgrepmore pattern infile1 [...infile2...]
```

For **each** `infile` specified, open that file, but instead of copying to a specific file or to standard output, write the contents of the file into a pipeline of `grep pattern` piped into `more` (you could also use `pg` or `less`). This will require forking and execing two child processes. It is recommended that your original program be the parent of both the grep and the more children (as opposed to using grandchildren).

The intention is to display only those lines of the file that match the pattern, and to page through them one screen at a time. When the user exits the pager program (by pressing the q key), move on to the next `infile`, if any. When the user sends a keyboard interrupt SIGINT (typically by pressing Control-C), do not abruptly terminate the program. Instead, first write to stderr the total number of files and bytes processed, and then exit.

Is this a fairly silly exercise? Perhaps. One could accomplish the same thing with existing UNIX utilities in a number of ways.

Be on the lookout: Setting up the pipes, and in particular avoiding dangling file descriptors, is important to proper program operation. If your program terminates unexpectedly or gets stuck in an endless loop, this may be a symptom of sloppy file descriptor work.

In order to properly test all aspects of your program, you must use sample files that are at least 16K long, so that the pipe buffer can actually fill up. You must also test for proper behavior when the user quits the pager program, or sends SIGINT. Using the ps command, make sure that when your program exits, under any circumstances, that the more and grep commands have also exited. If they get stuck, it is probably because of dangling file descriptors.

The same restrictions apply as in PS#1: Do not use the stdio library for opening, closing, reading, writing, forking, execing or setting up the pipe. Use the system calls directly. You may use library functions for argument parsing, error checking and reporting, printing out the exiting message, etc.

Helpful note: In order for `more` to receive keystrokes without interfering with reading from stdin, it opens a special device `/dev/tty` which is your session terminal. We haven't covered this yet. Furthermore, to receive the keystrokes without waiting for a newline character, more will put the `tty` device into a special mode. It may happen during development and testing of this program that more exits without restoring the `tty` to the normal mode, and your terminal session will appear to be "crashed". Characters will not echo and nothing will seem to work. To get out of this mode, press Control-J several times, then type the command `stty sane` and then press Control-J (NOT the Enter key!) again.