CS246 lab Notes #4 Files, Pipes, Redirection

- File functions
  - fopen
    - The possible file open types:
      - r - Reading only. Start at beginning of file
      - r+ - Reading and writing. Note: all writing is overwriting, not inserting.
      - w - Writing only. In other words, deletes contents of file (sets to 0 length). Cannot read previously written contents of file.
      - w+ - Reading and writing with truncation. In other words, deletes the whole file, but you can read previously written contents of file if necessary.
      - a - Appending. Cannot read contents of file.
      - a+ - Reading and Appending. Can read, but all writes are done at the end of the file regardless of calls to fseek or similar.
    - Note: The above is machine dependant, you should check before working on a new platform.
    - Also note: adding a "b" to the end of the file open type opens the file for binary editing.
      - Sometimes the operating system does things for convenience of the user when writing to text files, opening a file for binary editing suppresses all of these things.
  - feof
    - feof(<stream>);
    - Returns true if you are at the end of a file
    - Example: while(!feof(fp)) { /* code */ }
  - fflush
    - fflush(<stream>);
    - Saves all changes to the stream right then instead of waiting for one of the default pipes.
    - Usually unnecessary, as fclose calls it.
    - Sometimes things get out of sync; also want to use if the program is crashing and the changes are not being updated.
- Streams revisited
  - The file pointer argument to the above functions is considered a stream
  - Also, we've seen three "standard" streams before
    - stdin
      - Standard Input
      - External input to a function
      - What scanf, getc, gets take by default
    - stdout
      - Standard output
      - Normal output from a function

        - What printf , putc, puts use by default
    - stderr
        - Standard error
        - A different output from a function
        - Separate from stdout
        - No C function uses this by default, need to use <func>(stderr, args) to use it:
          fprintf(stderr, “out of memory”);
- Piping
    - UNIX has the capability to put a program's output somewhere other than the terminal, such as feed it into another program.
        - Basically, UNIX forms a link between the stdout of one program and the stdin of another program.
    - Example: suppose you have a file hw1.c in your directory. Type cat hw1.c | more
        - Remember that cat displays a file to stdout without pause
        - cat with no arguments will take input from stdin and display it to stdout
        - cat with multiple arguments (filenames) will concatenate all files to stdout
    - This puts the output of cat hw1.c into the functionality of the more program.
    - Brief tangent:
        - New command, finger
        - Displays all the users currently logged onto the system.
        - There are far too many users to read all on one screen, so try finger | more
    - The advantages of piping are more pronounced when you use some of the more specialized UNIX utilities.
    - Note: you may pipe as many times as you want. A chain of programs piping to each other is called a pipeline.
        - Try: cat | cat | cat | cat | cat
        - What happens?
    - pipe both stderr and stdout into the next program's stdin.
        - csh/tcsh: |&
        - bash: must talk about redirection first
- Redirection
    - Think about the beginning and end of a pipeline. The stdin is the user input to the terminal, and the stdout is what is printed to the terminal.
    - But what if you want the input from another source, like a file?
    - Remember that stdin, stdout and stderr are just special files setup by the system with specific names. Redirection allows you to use a file in place of any of the three stream locations.
    - Brief tangent:
        - wc
        - Counts characters, lines, words in a file

        - By default displays all; wc is equivalent to wc –clw
        - -c – Characters
        - -l – Lines
        - -w – Words
    - stdin from file
        - Use < after the command to use the file following it as the input file.
        - Example: try the following three operations.
            - wc hw1.c
            - wc < hw1.c
            - cat hw1.c | wc
    - stdout to file
        - Use > after the command to use the file following it as the output file.
        - Example:
            - ls –l > ls.txt
        - That's how we got the long file from the first recitation
    - stderr to file (csh/tcsh can not do this)
        - In general, stdout is considered the “first” output stream and stderr the “second”, and thus they are represented by 1 and 2 respectively
        - Example:
            - bash: cat blah 2> error.txt
    - stdout to stderr (csh/tcsh can not do this)
        - stdout of a program to is written combined to its stderr
        - Example
            - bash: cat blah 1>&2
            - bash: cat blah 2> error.txt 1>&2
            - can you pipe this?
    - stderr to stdout (csh/tcsh can not do this)
        - stderr of a program to is written combined to its stdout
        - Example
            - bash: cat blah 2>&1
            - bash: cat blah 2>&1 | wc –l
            - bash: cat blah >error.txt 2>&1
    - both stdout and stderr to file
        - Example
            - bash and csh/tcsh: cat blah >& error.txt
    - bash: mv blah gunk 1>&2 2> error.txt
    - How to suppress messages:
        - Unix has a file called /dev/null
        - It is always empty, even if you redirect to it
        - Thus to suppress error messages in bash, just do <command> <args> 2> /dev/null
        - Or to suppress all output (both bash and csh/tcsh): <command> <args> >& /dev/null

- Excercise:
  - Write a C program that prints something to stdout and stderr.
  - Redirect program out:
    - stdout pipe to wc -l
    - stdout only to file
    - stderr only to file
    - stderr to stdout then pipe to wc -l
    - stderr to stdout then to file
    - stdout to stderr then to file
    - all output to file