CSE 333 25au Exercise 6

out: Monday, October 6, 2025
due: Wednesday, October 8, 2025 by **10:00 am**, **No late exercises accepted**.

Goals: Learn how to use various parts of the standard C file I/O library, particularly ones that will be needed on later projects. Also reenforce understanding of exactly what is in a file that is stored on the disk.

Description: Write a C program that accepts a filename as a single command-line argument. The program should read the file, copying the contents of the file to `stdout` in reverse order, character by character (or, more precisely, byte by byte).

For example, consider a file name `foo.txt` in the current directory that contains the following 7 bytes (the 7th byte is the newline character at the end):

```
hello!
```

Running the program should result in the following:

```
bash$ ./ex6 foo.txt

!ollehbash$
```

Pay careful attention to where the newline and other characters are in this output.

You will need to use several standard library calls in order to do this. In particular, you should use:

- `fopen` , with mode "rb", to open the file
- `fclose` to close the file
- `fseek` and `ftell` to figure out how large the file is. (We'd suggest you `fseek(f, 0, SEEK_END);` and then `size = ftell(f);` to get the file size, in bytes.)
- `fseek` and `fread` to read the file's contents, in reverse order, byte-by-byte.
- `printf` , with the `%c` format code, to print out the characters (bytes) of the file one at a time.

Your code should read the bytes one-at-a-time and print them immediately. Do not read the entire file into memory and print it backwards or create a reversed copy of the data and print that. A goal of this exercise is to gain experience with the various file I/O functions described above.

Remember to consult the C/C++ Reference Material (http://www.cplusplus.com/reference/) and Linux manual pages (https://man7.org/linux/man-pages/index.html) to learn how to use each of these functions. (Check out the `stdio.h` header)

Note: Some of these library functions are quite old and have parameters or results whose type is `long`. Style guides and programs like `cpplint` generally discourage using `long` and recommend using types like `int64_t` instead. But for these library functions it's appropriate to use variables of type `long` to match function parameter and result types, since that guarantees an exact match regardless of the exact size of `long` on the underlying machine. Feel free to ignore any `cpplint` warnings about using `long` in these situations.

Hint: you can use linux commands like `hexdump -C foo.txt` or `hexdump -c foo.txt` to look at the bytes in a file `foo.txt` in hex and in other formats to see exactly what's there, including invisible characters. Many text editors also have modes to do this.

Your code must:

- compile without errors or warnings on CSE Linux machines (lab workstations, attu, or CSE home VM)
- have no crashes, memory leaks, or memory errors on CSE linux machines (Suggestion: `valgrind`)
- be contained in a single file called `ex6.c` that compiles with the command `gcc -Wall -g -std=c17 -o ex6 ex6.c` -- do not submit a Makefile.
- be pretty: the formatting, modularization, variable and function names, and so on must make us smile rather than cry. (Suggestion: `cpplint --clint`)
- be robust: you should think about handling bogus input from the user (if any), and you should handle hard-to-handle cases (if there are any) gracefully.
- have a comment at the top of the source file with your name, and CSE or UW email address.

You should submit your exercise using the Gradescope dropbox linked on the course resources web page.