March 10th, 2022

CSE344 – System Programming - Homework #1 - v2
File I/O

Objective
The goal is to develop a simple text editor accomplishing a common task: string replacement. It must support the following:

a) `./hw1 '/str1/str2/' inputFilePath`
This will replace all occurrences of `str1` with `str2` in the file `inputFilePath` (relative or absolute)

b) `./hw1 '/str1/str2/i' inputFilePath`
This will perform the same as before, but will be case insensitive.

c) `./hw1 '/str1/str2/i;/str3/str4/' inputFilePath`
It must be able to support multiple replacement operations.

d) `./hw1 '/[zs]tr1/str2/' inputFilePath`
It must support multiple character matching; e.g. this will match both `ztr1` and `str1`

e) `./hw1 '/^str1/str2/' inputFilePath`
It must support matching at line starts; e.g. this will only match lines starting with `str1`

f) `./hw1 '/str1$/str2/' inputFilePath`
It must support matching at line ends; e.g. this will only match lines ending with `str1`

g) `./hw1 '/st*r1/str2/' inputFilePath`
It must support zero or more repetitions of characters; e.g. this will match `sr1`, `str1`, `sttr1`, `sttttr1`, etc

And of course it must support arbitrary combinations of the above;
e.g. `./hw1 '/^Window[sz]*/Linux/i;/close[dD]$/open/' inputFilePath`

Assume that the user will try to match/replace only letters and digits, but no symbols or space. Hence you will not need escape characters.

Output
The output will be written to the input file. If you need temporary files, you must use `mkstemp()`.

Is this all?
No. The program might be executed as multiple instances with different replacement commands, on **the same file.** This will result in multiple processes manipulating the same input file with potentially wrong results.

Imagine for instance one process replacing str1 with str2, and another replacing str2 with str3 at the same time. Depending on which process writes/reads what, the end result might vary. You cannot stop the user from invoking your program multiple times, so instead you must protect file access (via the locks seen during our lectures) by making sure than no second instance of the program runs on the same file, before the first one has finished its task with it.

For all file I/O operations, use the low-level system calls presented during our lecture and not C library functions. Make sure you control each system call's return value and print informative

messages via perror/strerror in case of errors. Make sure you print your program's usage in case the commandline arguments are wrong.

Evaluation
This is an all or nothing homework, make it work!

Rules:
1) Compilation error: grade set to 1; if the error is resolved during the demo, then evaluation continues.
2) Compilation warning (with respect to the **-Wall** flag); -1 for every warning until 10. -20 points if there are more than 10 warnings; no chance of correction at demo.
3) No makefile: -30
4) No pdf report submitted (or submitted but insufficient, e.g. 3 lines of text with no design explanation, etc): -20. Other file formats are not admissible.
5) If the required command line arguments are missing/invalid, your program must print usage information and exit. Otherwise: -10
6) The program crashes or doesn't produce expected output with normal input: -100
7) The program doesn't satisfy a requirement: -100
8) Presence of memory leak (regardless of amount – checked with valgrind) -30
9) Late submissions will not be accepted
10) In case of an arbitrary error, exit by printing to stderr a nicely formatted informative message. Otherwise: -10

Is my homework submission valid?
If it works with the case (a), it will be considered valid.

Submission rules:
- Your source files, your makefile and a report; place them all in a directory with your student number as its name, and zip the directory.
- Your report must contain: how you solved this problem, your design decisions, which requirements you achieved and which you have failed.
- The report must be in English.
- You are expected to code the regular expression matching on your own, no libraries!
- Your makefile must only compile the program, not run it!
- Do not submit any binary executable files. The TAs will compile them on their own boxes.
- Your code will be compared against online sources; you are free to be inspired, but you are also expected to write your own code.
- Homeworks are invidual tasks. Proven cases of plagiarism will be punished to the full extent.

Hints:
- Plan/design carefully.
- Check for memory leaks with valgrind before submitting.
- Compile and run your program on more than one POSIX systems to ensure portability, if you can.
- Control whether you satisfy each and every one of the requirements prior to submission.

Good luck.