GTU Department of Computer Engineering
CSE 344 - Spring 2022
Homework #1 Report

Hasan Mutlu
1801042673

One small note before all: When using combinations of requirements d.) and g.) multiple times, the program might run for up to 10 seconds. Please be patient.

1. How I Solved This Problem?

These new kind of system call functions were a new concept for me. So I firstly tested read() and write() functions. After figuring out how these functions work, I started developing the program.

At first, I added checking the validity of the input of course. Then, I implemented first three requirements one by one, they were easiest yet the most general requirements. So I attempted implementing them in a way that they would enable combinations. After that, I decided to add requirements e.) and f.), which are related to matching strings only at the line starts or line ends. Then, I implemented the hardest requirements, which are d.) and g.), one by one.

After that, I tested which different combinations are currently working. Combinations of e.) and f.) weren't working and I thought it doesn't make sense to use them combinationally since they are opposite of each other, so I left it as it is. Combinations of d.) and g.) weren't working and it was the most difficult part to make them work together, but I managed to do it.

In the middle of this whole development, I added file locking functionality with fcntl() to protect file from other processes' write operations. But when I tested the program with valgrind, it gave some warnings. I tracked these warnings and saw that they were about struct flock's empty variables. I made a research and set its empty variables to most default values possible. Then the warnings were gone.

2. My Design Decisions

Definitions of some functions:

int inputFormatCheck(const char* argv1):

- ➔ This function checks if the input format is valid.
- ➔ @param argv1 Takes a pointer to argv[1]
- ➔ @return 1 if valid, 0 if invalid

void parse(char* text, char* str1, char* str2):

- ➔ This function parses each replacement operation command. It parses the input to str1 and str2.
- ➔ @param text Pointer to start of the command.
- ➔ @param str1 Pointer to str1
- ➔ @param str2 Pointer to str2

int dCheck():

- ➔ This function checks if there is multiple character matching.
- ➔ @return 1 if there is multi character matching, 0 if there isn't, -1 if closing bracket (']') not found.

int gCheck(char* str1):

- ➔ This function checks if there is multiple repetition of a character
- ➔ @param str1 Pointer to str1
- ➔ @return 1 if there is, 0 if there isn't.

int replace(char* filepath, char* str1):

- ➔ This function replaces str1 with str2
- ➔ @param filepath Pointer to file path
- ➔ @param str1 Pointer to str1
- ➔ @return 1 if there is an error, 0 if there is no issues and replacement is completed.

Design Explanations:

When parsing the input string in parse() function, I parsed an input like "/^[zs]t*r1/str/" into "[zs]t*r1" and "str2". If there is existance of "^" or "$" characters in input, they are replaced with newline characters. Parsing of the remaining string "[zs]t*r1" was done in dCheck() and gCheck() functions.

When replacing strings, I opened the actual file in read only mode and created a new output file in write only mode. I traversed the file starting from every character, by reading string1's length of bytes and seeking back to the next character. I wrote the new contents into the output file. After all file is covered, I deleted the actual file and renamed output file to replace it.

3. Achived and Failed Requirements

Seperately, all requirements work. But some of them have exceptions.

- ➔ Requirement e.) doesn't work for the string at the very beginnig of the file, where there isn't a line above.
- ➔ Requirement f.) doesn't work for the string at the very end of the file, where there isn't a new line below.
- ➔ Requirement g.) doesn't work for characters at the beginning and ending of the string1. For example, inputs like /qw*e/ and /qwe*r/ work but /q*we/ and /qwe*/ doesn't work.

Every combination work except one:

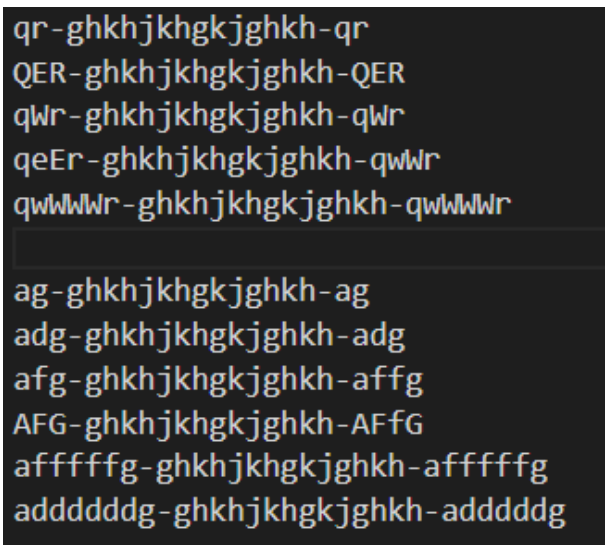
- ➔ Combinations of e.) and f.) doesn't work as they are logically opposite anyway.

One Example Test:

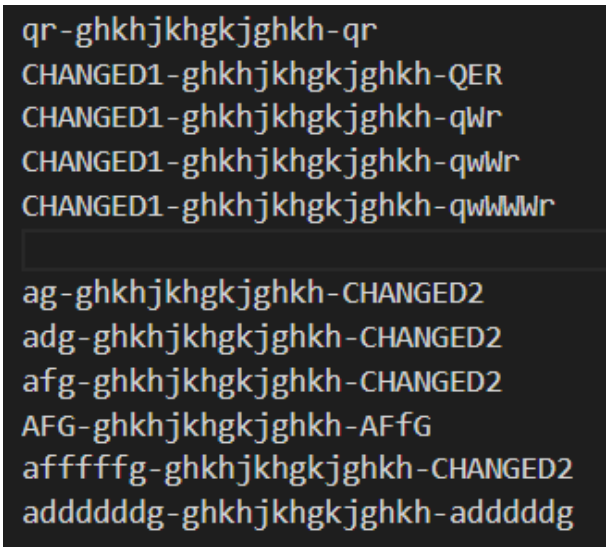
Command :

```
./executable '/^q[we]*r/CHANGED1/i;/a[df]*g$/CHANGED2/' input.txt
```

Before:



After:



Valgrind Result:

```
Replacement is finished successfully.
==3277==
==3277== HEAP SUMMARY:
==3277==     in use at exit: 0 bytes in 0 blocks
==3277==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==3277==
==3277== All heap blocks were freed -- no leaks are possible
==3277==
==3277== For counts of detected and suppressed errors, rerun with: -v
==3277== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```