

**GTU**  
**DEPARTMENT OF**  
**COMPUTER ENGINEERING**

**CSE 344 – Spring 2022**

**HOMEWORK 1**  
**REPORT**

**SÜLEYMAN GÖLBOL**  
**1801042656**

# 1. REQUIREMENTS

## *NONFUNCTIONAL REQUIREMENTS*

1. Portability → The application should be portable. All computers that has Linux Distro and GCC compiler can run the program. Also it can be run on Windows when Windows Subsystem for Linux 2 activated.
2. Maintainability → In case of an error occurrence, the system uses perror or stderr in order to give feedback on terminal.
3. Performance → The system should initially be able to process as many entries as possible. Each request has to be processed with different terminals. The system's performance should be fast enough to show user the feedback.

## *FUNCTIONAL REQUIREMENTS*

In order to compile the program, user have to use “make” command that uses gcc. If make or gcc is not installed user can install it via “*sudo apt-get install build-essential*” command. Make command runs “gcc main.c sg\_replacer.c -Wall -o hw1” command.

In order to run the program, user have to write 2 extra command line arguments. First is the string occurrence instruction, second one is input file path.

```
./hw1 '/str1/str2/' inputFilePath
```

This will replace all occurrences of str1 with str2 in the file inputFilePath

```
./hw1 '/str1/str2/i' inputFilePath
```

This will perform the same as before, but will be case insensitive.

```
./hw1 '/str1/str2/i;/str3/str4/' inputFilePath
```

It must be able to support multiple replacement operations.

```
./hw1 '/[zs]tr1/str2/' inputFilePath
```

It must support multiple character matching; e.g. this will match both ztr1 and str1

```
./hw1 '/^str1/str2/' inputFilePath
```

It must support matching at line starts; e.g. this will only match lines starting with str1

```
./hw1 '/str1$/str2/' inputFilePath
```

It must support matching at line ends; e.g. this will only match lines ending with str1

```
./hw1 '/st*r1/str2/' inputFilePath
```

It must support zero or more repetitions of characters; e.g. this will match sr1, str1, sttr1 etc.

```
./hw1 '/^Window[sz]*/Linux/i;/close[dD]$/open/' inputFilePath
```

It must support arbitrary combinations of the above.

## 2. PROBLEM SOLUTION APPROACH

Firstly, I defined 5 replace mode. And I made them in a special way so if we AND different replace modes in binary bitwise mode, it will give us 0 result.

Also when I OR them in bitwise mode, because of all of them on different digits, different OR's will give us different results all the time.

```
typedef enum ReplaceMode{  
    SENSITIVE = 16, /* Binary: 10000 */  
    INSENSITIVE = 8, /* Binary: 01000 */  
    LINE_START = 4, /* Binary: 00100 */  
    LINE_END = 2, /* Binary: 00010 */  
    REPETITION = 1, /* Binary: 00001 */  
} ReplaceMode; /* Replace mode is in binary
```

So, if I want to Bitwise OR SENSITIVE, LINE\_START and LINE\_END; The value of mode is special. It will never be same with other way OR's. I used this while I am comparing results.

Another problem that I had was arbitrary mode. So I created different functions for different modes. For example if first argument contains both [ ] operations and \* operation, my specific function calls replace() method for every character inside square brackets.

Also, there is a possibility that other character inside bracket has more repetition on file. So, I created a parameter called isLast.

For example; if argument is /Window[sz]\*/Linux/ and the file contains Windowzz;

My program will make Windowzz -> Linuxzz. To prevent this, only in the last character of square bracket's mode, it converts 0 repetition occurrences. So it makes Windowzz -> Linux.

.

### 3. TEST CASES AND RESULTS

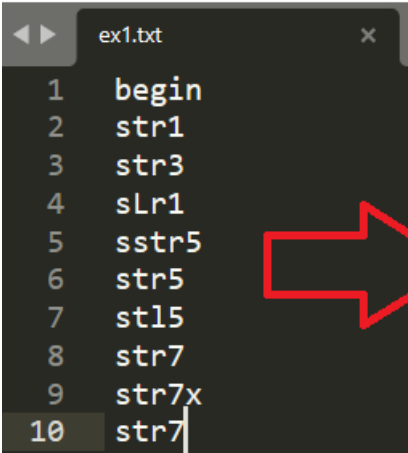
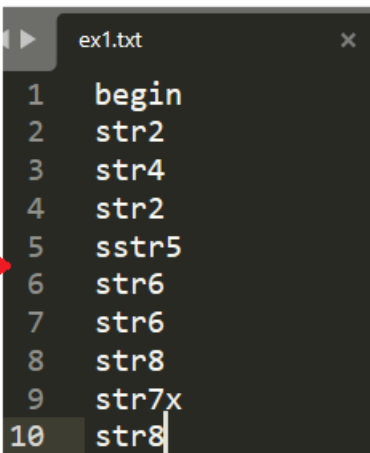
3.a) `./hw1 '/s[tl]r1/str2/i;/sTR3/str4/i;/^st[lr]5/str6/;/str7$/str8/' files/ex1.txt`

The command above has 4 operations. The first one and the second one are insensitive and the third and fourth one is sensitive due to doesn't contain "i" character.

For the first one, it replaces all stl1 and str1's [insensitive] with str2.

Second one replaces sTR3's with str4 [insensitive].

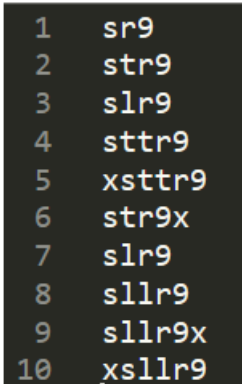
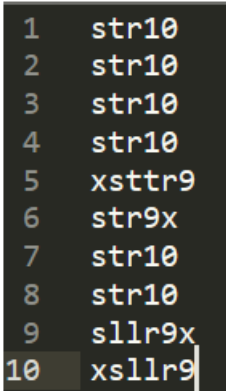
Third one replaces str5's and stl5's with str6 only if it's on the beginning of line. Fourth one replaces str7's with str8 only if it's at the end of the line.

INPUT	OUTPUT
 <pre>1 begin 2 str1 3 str3 4 sLr1 5 sstr5 6 str5 7 stl5 8 str7 9 str7x 10 str7</pre>	 <pre>1 begin 2 str2 3 str4 4 str2 5 sstr5 6 str6 7 str6 8 str8 9 str7x 10 str8</pre>

3.b) `./hw1 '/^s[lt]*r9$/str10/' files/ex2.txt`

The command above has 1 operation. The purpose is replacing str9's and slr9's with str10 [if it's both starts on that line and ends on that line.]

Also it supports repetition of l and t characters even it has 0 repetition.

INPUT	OUTPUT
 <pre>1 sr9 2 str9 3 slr9 4 sttr9 5 xsttr9 6 str9x 7 slr9 8 sllr9 9 sllr9x 10 xsllr9</pre>	 <pre>1 str10 2 str10 3 str10 4 str10 5 xsttr9 6 str9x 7 str10 8 str10 9 sllr9x 10 xsllr9</pre>

3.c) `./hw1 '/^Window[sz]*/Linux/i;/close[dD]$/open/' files/ex3.txt`

The command above has 2 operations.

For the first operation the purpose is to replace Window, Windows, Windowss, Windowz, Windowzz (and with more repetitions) with Linux in insensitive way. For the second one the purpose is replace closed and closed that is at the end of the line.

INPUT	OUTPUT
1 xWindows	1 xWindows
2 xWindowz	2 xWindowz
3 xxWindowss	3 xxWindowss
4 Windows	4 Linux
5 Windowss	5 Linux
6 WINDOWSS	6 Linux
7 WINDOWSSxx	7 Linuxxx
8 Windowz	8 Linux
9 Windowzz	9 Linux
10 WINDOWZZ	10 Linux
11 WINDOWZZxx	11 Linuxxx
12 closedx	12 closedx
13 closeDx	13 closeDx
14 closed	14 open
15 closeD	15 open

3.d) `./hw1 Invalid Command Line Argument`

In the case of invalid command line argument program exits with perror and prints instructions.

```
sg1b1@Sg1b1PC:/mnt/c/Apparatus/GTU/Year3/Semester2/CSE344/hw1$ ./hw1 Invalid Command Line Argument
ERROR FOUND ON ARGUMENTS; PLEASE ENTER A VALID INPUT! INSTRUCTIONS:
./hw1 '/str1/str2/' inputFilePath -> For just to replace
./hw1 '/str1/str2/i' inputFilePath -> Insensitive replace
./hw1 '/str1/str2/i;/str3/str4/' inputFilePath -> Multiple replacement operations
./hw1 '/[zs]tr1/str2/' inputFilePath -> Multiple character matching like this will match both ztr1 and str1
./hw1 '/^str1/str2/' inputFilePath -> Support matching at line starts like this will only match lines starting with str1
./hw1 '/str1$/str2/' inputFilePath -> Support matching at line ends like this will only match lines ending with str1
./hw1 '/st*r1/str2/' inputFilePath -> Support 0 or more repetitions of characters like this will match sr1, str1, sttr1
./hw1 '/^Window[sz]*/Linux/i;/close[dD]$/open/' inputFilePath -> it supports arbitrary combinations of the above
: Success
Goodbye!
```

3.e) Running at the Same Time (Multiple Processes Manipulating)

When I put the different commands simultaneously; in the first one purpose is replace close's with "open"; in the second terminal purpose is replace open with "acik". The results is below. It works.

## VALGRIND MEMORY RESULTS

The output from valgrind about heap and leaks is like below:

```
Successfully wrote to the file
==12784==
==12784== HEAP SUMMARY:
==12784==    in use at exit: 0 bytes in 0 blocks
==12784==   total heap usage: 49 allocs, 49 frees, 1,659 bytes allocated
==12784==
==12784== All heap blocks were freed -- no leaks are possible
==12784==
==12784== For lists of detected and suppressed errors, rerun with: -s
```

Valgrind gives warning (shows on error summary) if we use string operations for an empty string, and I have this problem. For example `char *str = ""` and I used `strlen(str)`. But it didn't create any problem while I'm running.

## 4. SYSTEM CALLS

In order to make sure the results of system calls has no problem I check the return values of calls. And if it contains error I use `perror` function for the error and exit with a return call other than 0. Because 0 means success.

```
// OPENING FILE ON READ ONLY MODE
if( (fdRead = open(filePath, O_RDONLY, S_IWGRP)) == -1 ){
    perror("Error while opening the file to read.\n");
    exit(2);
}
```

```
// READING IS COMPLETED. CLOSING THE FILE
if( close(fdRead) == -1 ){
    perror("Error while closing the file.");
    exit(4);
}
```

While reading file, I'm putting read call inside of a while loop that doesn't have body. So it read has error it reads again until no error comes.

```
while( (readedBytes = read(fdRead, buffer, statOffFile.st_size)) == -1 && errno == EINTR ){ /* Intentionanlly Empty Loop to deal interruptions by signal */}
if(readedBytes <= 0){
    perror("File is empty. Goodbye\n");
    exit(3);
}
```

In the `read()` function; the size of the bytes that will be read and written into the buffer is in `statOffFile.st_size` variable. The size of file is found with `stat()` function in `<sys/stat.h>` header.

```
struct stat statOffFile; //Adress of statOffFile will be sent to stat() function in order to get size information of file.
if(stat(filePath, &statOffFile) < 0){
    perror("Error while opening the file.\n");
    exit(EXIT_FAILURE);
}

char *buffer = (char*)calloc( statOffFile.st_size , sizeof(char)); // Buffer that data will be stored
```

While opening file for the write mode I also use O\_TRUNC so this way I discard previous content of the file while writing. Also I use S\_IWGRP gives write access for group.

```
if( (fdWrite = open(filePath, O_WRONLY | O_TRUNC, S_IWGRP)) == -1 ){ // O_TRUNC helps us to truncate [writing like from scratch]
    perror("Error while opening the file to write.\n");
    exit(5);
}
```

## 5. LOCKING

For locking files, I used fcntl() function. In the image below, I used memset to initialize the locking by copying sizeof(lock) bytes of 0 char to lock. Then I set lock's l\_type variable to F\_WRLCK for a writing lock. Then because of system call safety I checked if fcntl() function is equal to -1 for the error possibility. Fcntl function puts write lock on the file.

```
// LOCKING
memset(&lock, 0, sizeof(lock)); // Init the lock system
lock.l_type = F_WRLCK; // F_WRLCK: field of the structure for a write lock.
if ( fcntl(fileDesc, F_SETLKW, &lock) == -1) { // Putting write lock on the file.
    perror("Error while locking with fcntl(F_SETLKW)");
    exit(EXIT_FAILURE);
}
```

At the end after writing to the buffer, I unlocked by setting lock.l\_type to F\_UNLCK. Then I sent the address of lock to fcntl function with F\_SETLKW parameter and again I checked the system call's return value for safety.

F\_SETLKW parameter helps us for if a signal is caught while waiting, then the call is interrupted and (after signal handler returned) to return immediately.

```
// UNLOCKING
lock.l_type = F_UNLCK;
if ( fcntl(fileDesc, F_SETLKW, &lock) == -1) {
    perror("Error while unlocking with fcntl(F_SETLKW)");
    exit(EXIT_FAILURE);
}
```