CSE344 – System Programming - Homework 1 Report

1-) Solution of Problem

I've create a simple struct which holds the data and size of data to holds words inside the input file. I read the input file charachter by charachter and combine them to create words. While I was reading words from file, I checked them each by each according to developed algorithm. If the expression and words are matched, I replaced the word with the given argument and placed It into the buffer. To inhibit the other process to acess file, I locked them. Algorithm whick I mentioned before, developed according to finite state machine to represents regular expression for this homework. Due to some reason that I can't figure that out, when I try to run my code with valgrind to detects memory leaks, my code won't work properly.

2-) Functions

int check_chars(char char1, char char2, int mode);

Checks two charachters whethey they are equal each other or not according to mode which represents the CASE_SENSITIVE or CASE_INSENSITIVE.

int check_input(char *string);

Checks the given arguments are valid or not.

char *check_words(char *word, char *expression, char *string, int mode);

Checks the given expression and words which come from file is equal or not. If they are equal it returns the new string.

int check_alternative(char *string, char *expression, int mode);

Controls the expression and word by using the developed algorithm.

char *remove_symbols(char *string);

Remove $ and ^ signs after the necessary values are get.

int check_line(char *arg);

Check the argument has ^ or $ symbol.

int find_case(char *arg);

Check the argument to determine whether it is case sensitive or not.

int find_operation(char *arg);

Checks the arguments to determine whether single operation or multiple operation.

void parse_arguments(int argc, char *argv[]);

Parses the arguments in proper way.

void write_file(char *file_path, Buffer text);

Writes buffer to given file path.

void read_file(char *fpath, char *str1, char *str2, int mode);

Reads files and modify the words into the buffer.

3-) Test Cases

3.1) Test case for a and b parts of homework.

Input: '/windows/linux/i'

```
1 windows - windowS - windowwws - windowz
```

Shape 1: Before the program executed.

Output:

```
1 linux - linux - windowwws - windowz
```

Shape 2: After the program executed.

3.2) Test case for c part of homework.

Input: '/windows/linux/i;/linux/android'

```
1 windows - linux - Windows - windowz
```

Shape 3: Before the program executed.

Output:

```
1 android - android - android - windowz
```

Shape 4: After the program executed.

3.3) Test case for d part of homework.

Input: '/[zs]tr1/str2/'

```
1 str1 - ztr1 - dtr1
```

Shape 5: Before the program executed.

Output:

```
1 str2 - str2 - dtr1
```

Shape 6: After the program executed.

3.4) Test case for e part of homework.

Input: '/^str1/str2/'

```
1 str1 str1
2  str1
```

Shape 7: Before the program executed.

Output:

```
1 str2 str1
2  str2
```

Shape 8: After the program executed.

3.5) Test case for f part of homeowork.

Input: /str1$/str2/‘

```
1 str1 str1 str1
2  str1 str1
3 str1
```

Shape 9: Before the program executed.

Output:

```
1 str1 str1 str2
2  str1 str2
3 str2
```

Shape 10: After the program executed.

3.6) Test case for G part

Input: '/st*r1/str2/'

```
1 sr1 str1 sttr1
2 sttttr1
```

Shape 11: Before the program executed.

Output:

```
1 str2 str2 str2
2 str2
```

Shape 12: After the program executed.