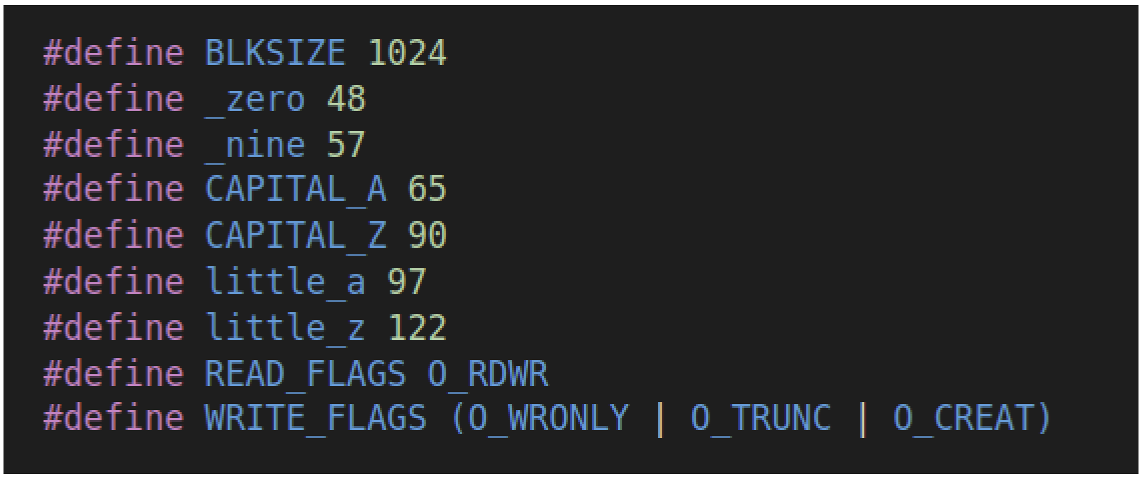
GEBZE TECHNICAL UNIVERSITY

# SYSTEMS PROGRAMMING COURSE HW1 REPORT

Yakup Talha Yolcu
1801042609

## 1-DEFINE’s

Firstly I have these #define ‘s



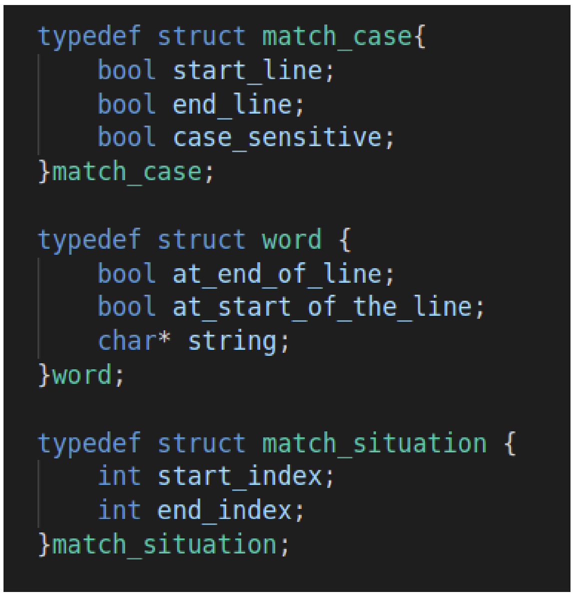
```c
#define BLKSIZE 1024
#define _zero 48
#define _nine 57
#define CAPITAL_A 65
#define CAPITAL_Z 90
#define little_a 97
#define little_z 122
#define READ_FLAGS O_RDWR
#define WRITE_FLAGS (O_WRONLY | O_TRUNC | O_CREAT)
```

BLKSIZE is to allocate memory for file operations
_zero is ASCII value of ‘0’
_nine is ASCII value of ‘1’
CAPITAL_A is value of ‘A’
CAPITAL_Z İs value of ‘Z’
and it is true for little of them

I have RDWR flag when opening a file to read.
I have O_WRONLY flag for writing, O_TRUNC flag to erase contents of file, O_CREATE to make sure of it is created newly.

## 2-STRUCTS’S

In match case struct if it is given that ^ in the match command, start line will be true.
If $ is given, end_line will be true
If /i is given, case_sensitive will be false

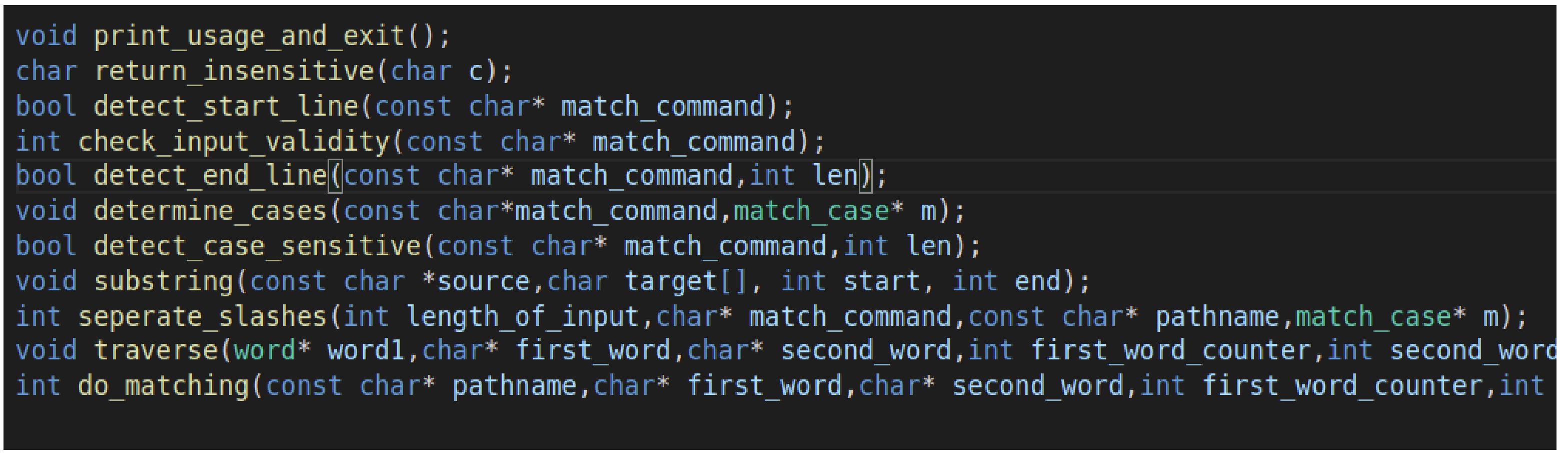


```c
typedef struct match_case{
    bool start_line;
    bool end_line;
    bool case_sensitive;
}match_case;

typedef struct word {
    bool at_end_of_line;
    bool at_start_of_the_line;
    char* string;
}word;

typedef struct match_situation {
    int start_index;
    int end_index;
}match_situation;
```

Word struct represents a single word that read from file. Then by looking \n or empty spaces, we can determine that if this word is at end of line or at start of the line in the file. char*string represents the word.

In match situation struct, for example I have astr1bcd in file, start index is 1 and end index becomes 3 which represent the start index of the change and end index of the change

## 3-FUNCTIONS



```
void print_usage_and_exit();
char return_insensitive(char c);
bool detect_start_line(const char* match_command);
int check_input_validity(const char* match_command);
bool detect_end_line(const char* match_command,int len);
void determine_cases(const char*match_command,match_case* m);
bool detect_case_sensitive(const char* match_command,int len);
void substring(const char *source,char target[], int start, int end);
int seperate_slashes(int length_of_input,char* match_command,const char* pathname,match_case* m);
void traverse(word* word1,char* first_word,char* second_word,int first_word_counter,int second_word
int do_matching(const char* pathname,char* first_word,char* second_word,int first_word_counter,int
```

**print usage and exit** is called whenever input is invalid
**return insensitive** returns the insensitive of the given char
<u>for example</u>: if it is given that ‘0’ it returns 0 because digits     don’t have any insensitive character
<u>another example</u>: if it is given that ‘A’ it returns ‘a’ vice versa is true.

**detect start line** detects the ^ is there or not
**check input validity** checks the input validity like is the number of / are %3==0 , if the first char is /, if the second char is * or not, are the [ and ] numbers is equal or not etc…

**detect end line** detects the $ is there or not

**determine cases** function looks to the input to detect and line and start line. It calls detect end line, detect start line and detect case sensitive functions.

**detect case sensitive** function look for /i

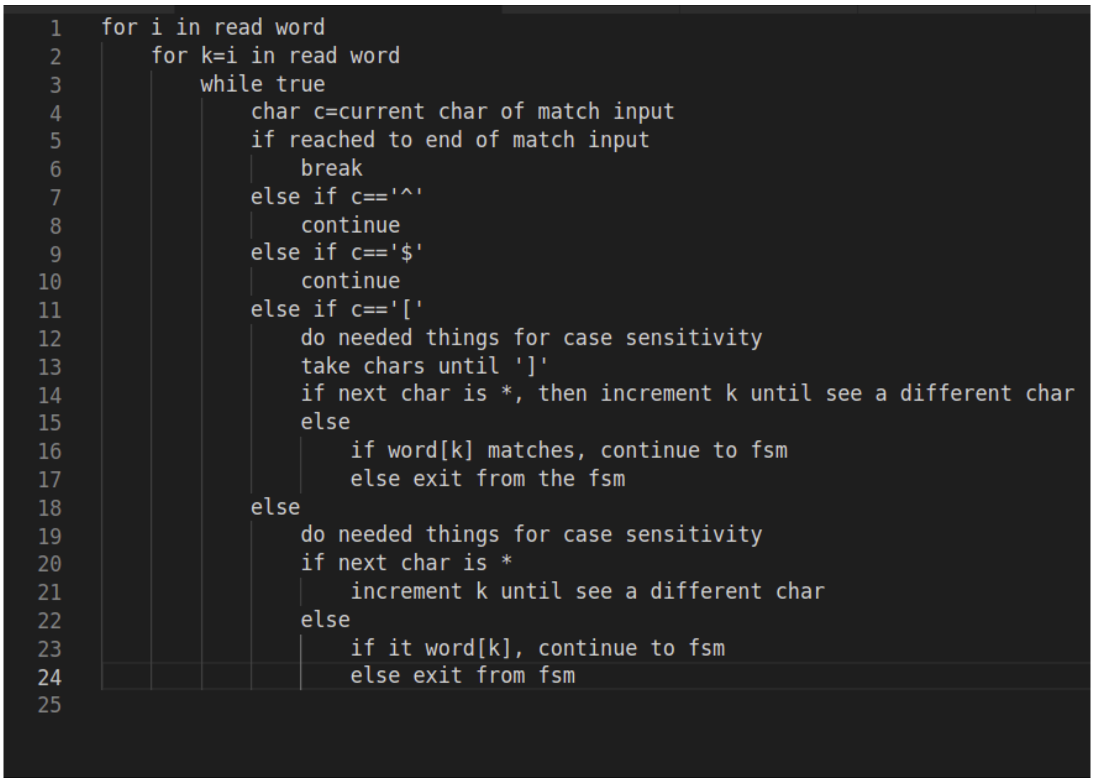
**substring** function returns a substring of the given string by given indices. I decided to write it on my own.

**seperate slashes** function tokenizes the string with the / token

**traverse function** has a fsm to find a match. It will be explained in detail

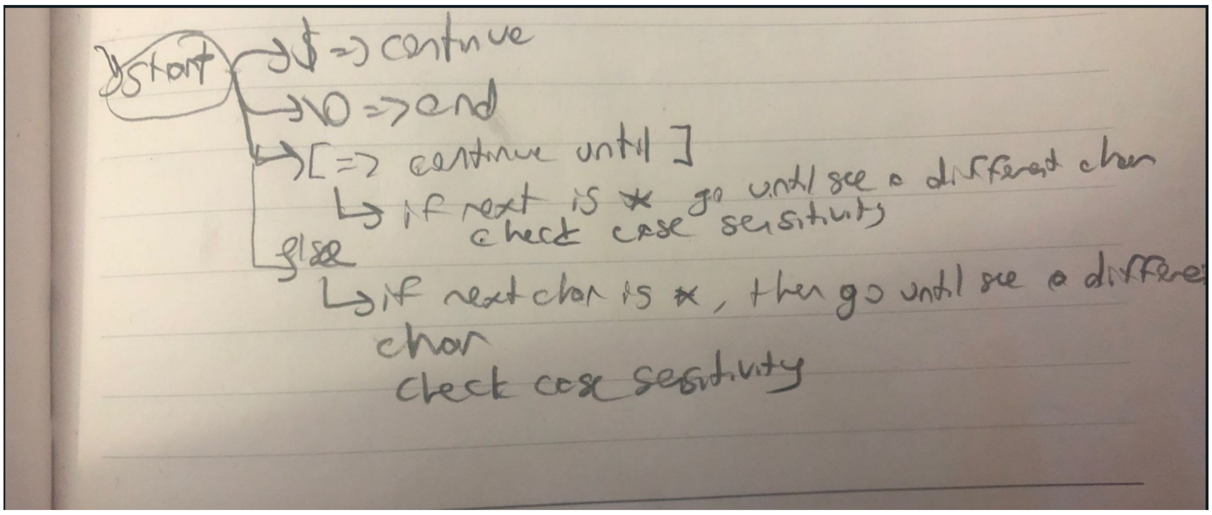
**do matching** function reads the file, sends the words to the traverse function and writes the results to the file

## 3-PSEUDOCODE OF THE FSM



```
for i in read word
    for k=i in read word
        while true
            char c=current char of match input
            if reached to end of match input
                break
            else if c=='^'
                continue
            else if c=='$'
                continue
            else if c=='['
                do needed things for case sensitivity
                take chars until ']'
                if next char is *, then increment k until see a different char
                else
                    if word[k] matches, continue to fsm
                    else exit from the fsm
            else
                do needed things for case sensitivity
                if next char is *
                    increment k until see a different char
                else
                    if it word[k], continue to fsm
                    else exit from fsm
```

## 4-DRAWING OF FSM



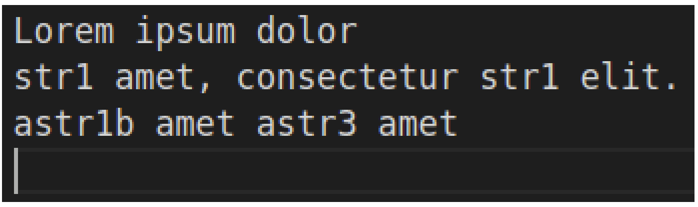
## 5-MY DESIGN DECISIONS

I decided to have two for loop which iterates through the read word and in the inner for loop, I have my FSM to find a match
If I found a match then start index of the change is current index of the outer for loop, end index of the change is current index of the inner for loop

## 6-ACHIEVED REQUIREMENTS

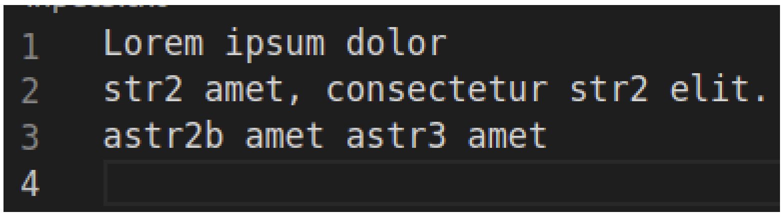
**case a-) ./hw1 "/str1/str2/" input3.txt**

Input file



```
Lorem ipsum dolor
str1 amet, consectetur str1 elit.
astr1b amet astr3 amet

```

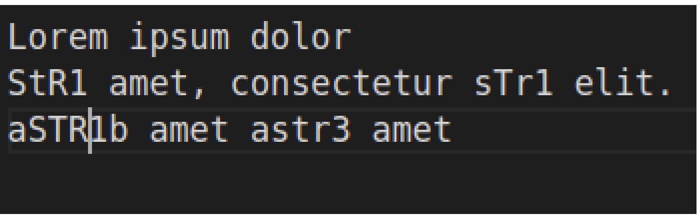
Result



```
Lorem ipsum dolor
str2 amet, consectetur str2 elit.
astr2b amet astr3 amet

```

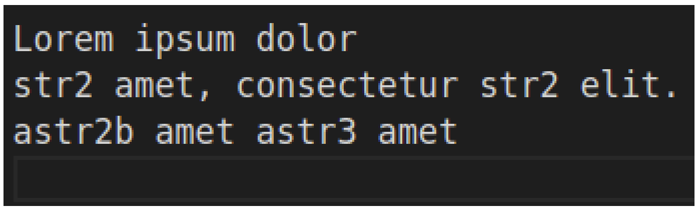
**case b-) ./hw1 "/str1/str2/i" input3.txt**

Input file



```
Lorem ipsum dolor
StR1 amet, consectetur sTr1 elit.
aSTR1b amet astr3 amet
```

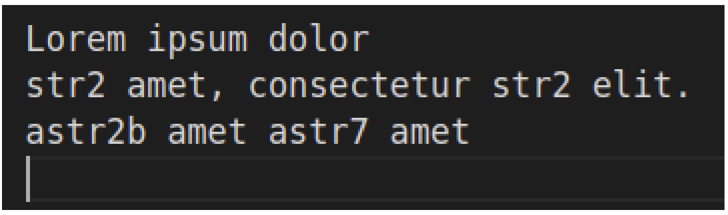
Result



```
Lorem ipsum dolor
str2 amet, consectetur str2 elit.
astr2b amet astr3 amet
```

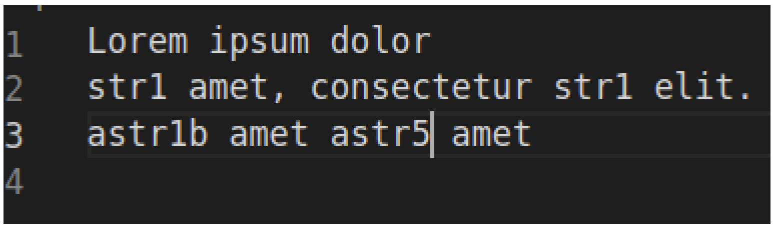
**case c-) ./hw1 "/str1/str2/;/str5/str6/;/str6/str7/" input3.txt**

Input file



```
Lorem ipsum dolor
str2 amet, consectetur str2 elit.
astr2b amet astr7 amet
```

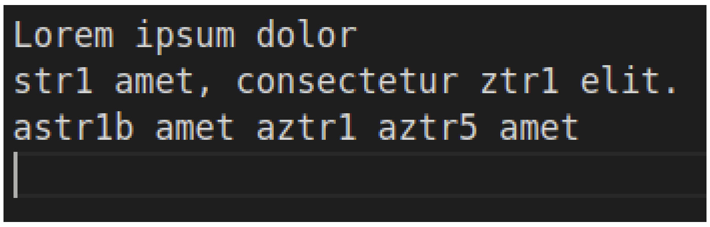
Result



```
Lorem ipsum dolor
str1 amet, consectetur str1 elit.
astr1b amet astr5 amet
```

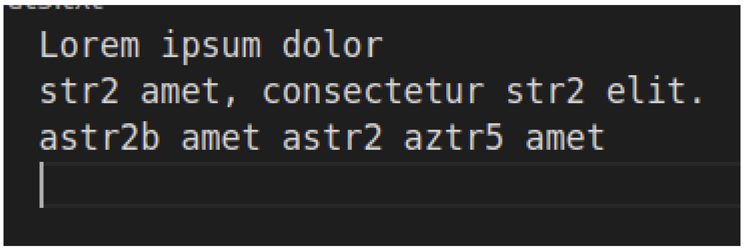
**case d-) ./hw1 "/[zs]tr1/str2/" input3.txt**

Input file



```
Lorem ipsum dolor
str1 amet, consectetur ztr1 elit.
astr1b amet aztr1 aztr5 amet
```

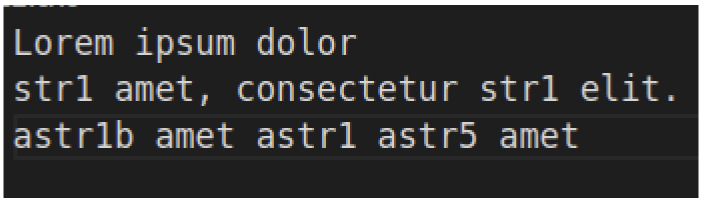
Result



```
Lorem ipsum dolor
str2 amet, consectetur str2 elit.
astr2b amet astr2 aztr5 amet
```

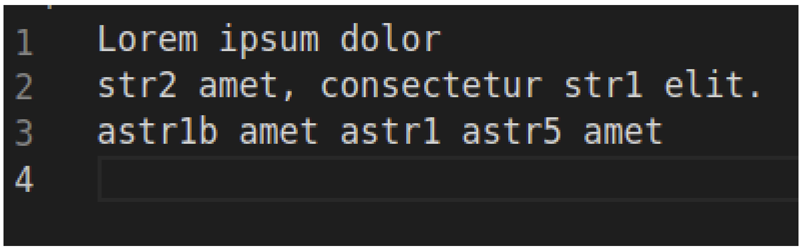
**case e-) ./hw1 "/^str1/str2/" input3.txt**

Input file



```
Lorem ipsum dolor
str1 amet, consectetur str1 elit.
astr1b amet astr1 astr5 amet
```

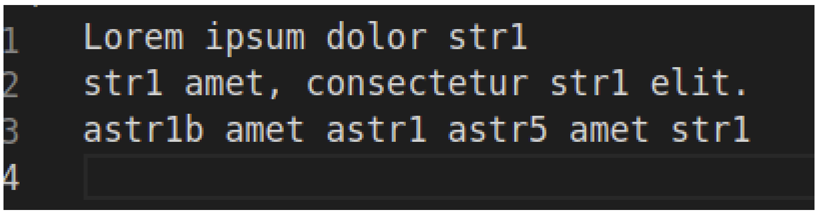
Result



```
Lorem ipsum dolor
str2 amet, consectetur str1 elit.
astr1b amet astr1 astr5 amet
```

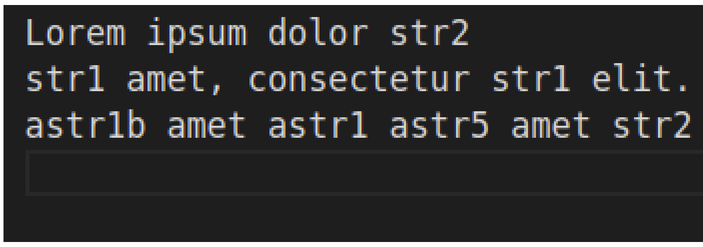
**case f-) ./hw1 "/^str1/str2/" input3.txt**

Input file



```
Lorem ipsum dolor str1
str1 amet, consectetur str1 elit.
astr1b amet astr1 astr5 amet str1
```

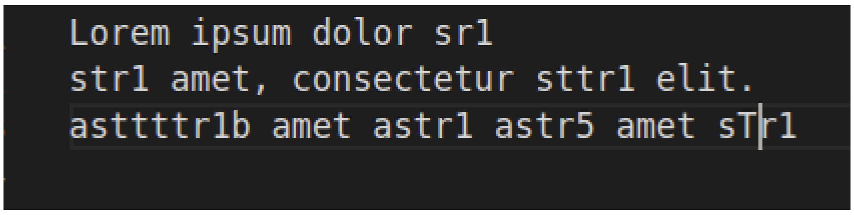
Result



```
Lorem ipsum dolor str2
str1 amet, consectetur str1 elit.
astr1b amet astr1 astr5 amet str2
```

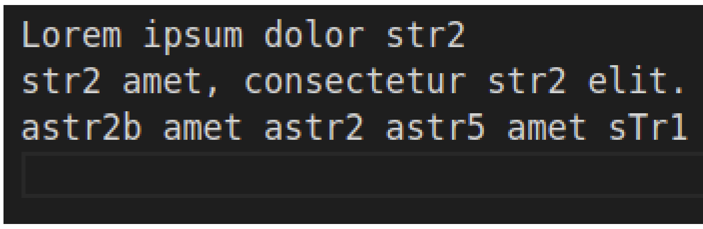
**case g-) ./hw1 "/st*r1/str2/" input3.txt**

Input file



```
Lorem ipsum dolor sr1
str1 amet, consectetur sttr1 elit.
asttttr1b amet astr1 astr5 amet sTr1
```

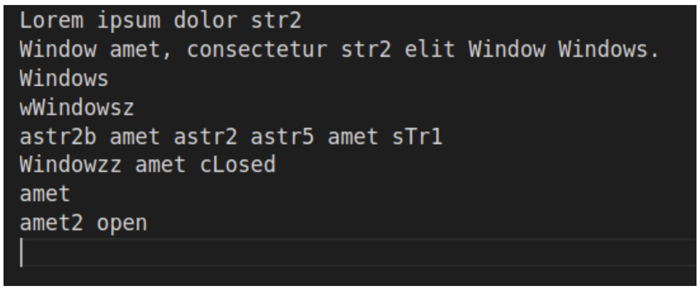
Result



```
Lorem ipsum dolor str2
str2 amet, consectetur str2 elit.
astr2b amet astr2 astr5 amet sTr1
```

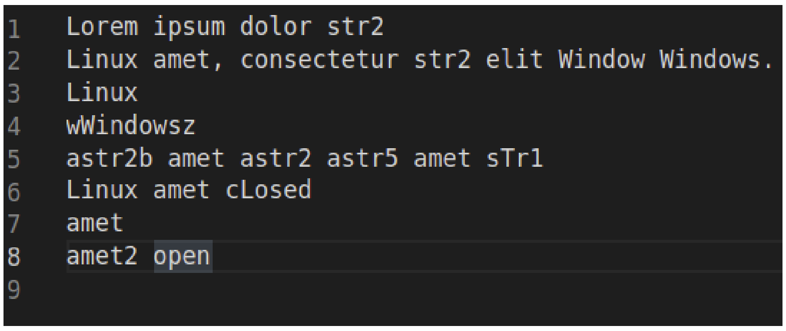
**bonus part-) ./hw1 "/^Window[sz]*/Linux/i;/close[dD]$/open/" input3.txt**

Input file



```
Lorem ipsum dolor str2
Window amet, consectetur str2 elit Window Windows.
Windows
wWindowsz
astr2b amet astr2 astr5 amet sTr1
Windowzz amet cLosed
amet
amet2 open
```

Result



```
Lorem ipsum dolor str2
Linux amet, consectetur str2 elit Window Windows.
Linux
wWindowsz
astr2b amet astr2 astr5 amet sTr1
Linux amet cLosed
amet
amet2 open
```

## 7-FAILED REQUIREMENTS

NONE