GEBZE TECHNICAL UNIVERSITY

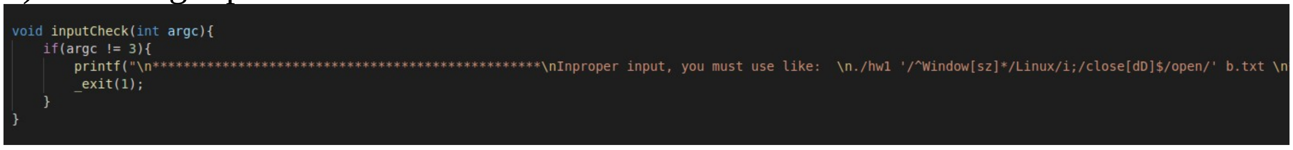
SYSTEM PROGRAMMING COURSE
HW-1

Derviş Ali DUMAN
1801042626
3th Grade

Design:

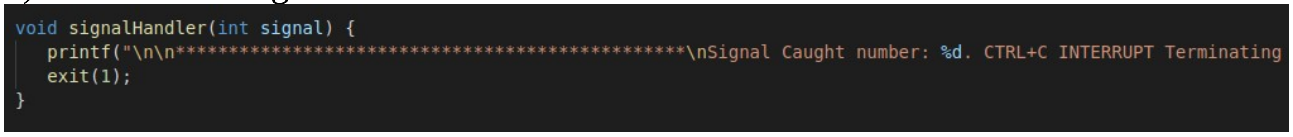
First thing I created a Finite State Machine for Regex in a paper then I just need to Implement it in C language. Of course there few steps to achive this goal and here they are:

1) Checking inputs from terminal and file:



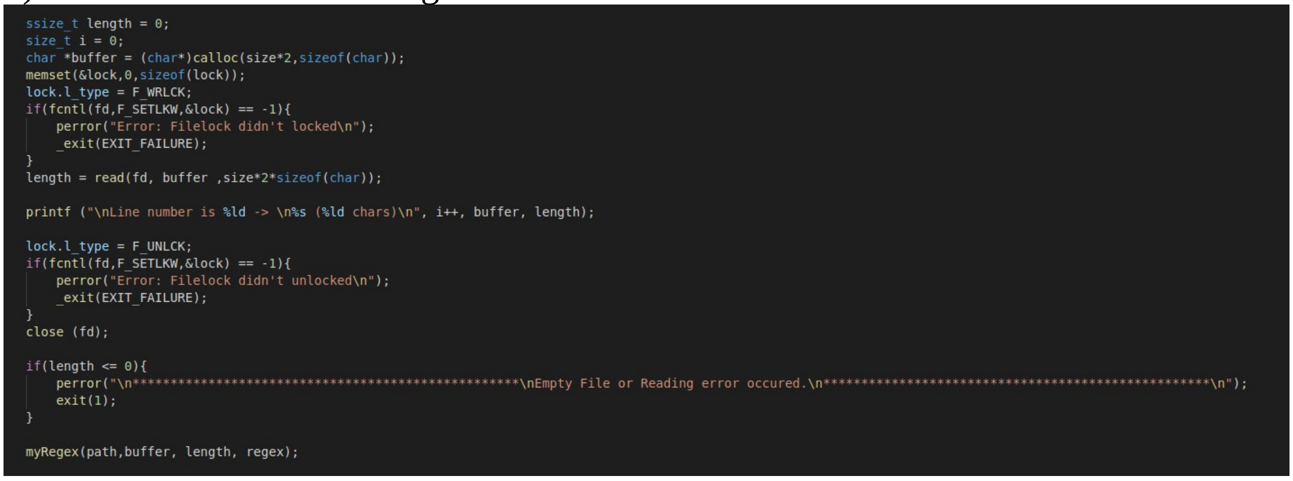
```c
void inputCheck(int argc){
    if(argc != 3){
        printf("\n**************************************************\nInproper input, you must use like:  \n./hw1 '/^Window[sz]*/Linux/i;/close[dD]$/open/' b.txt \n
        _exit(1);
    }
}
```

2) Check ctrl+c signal:



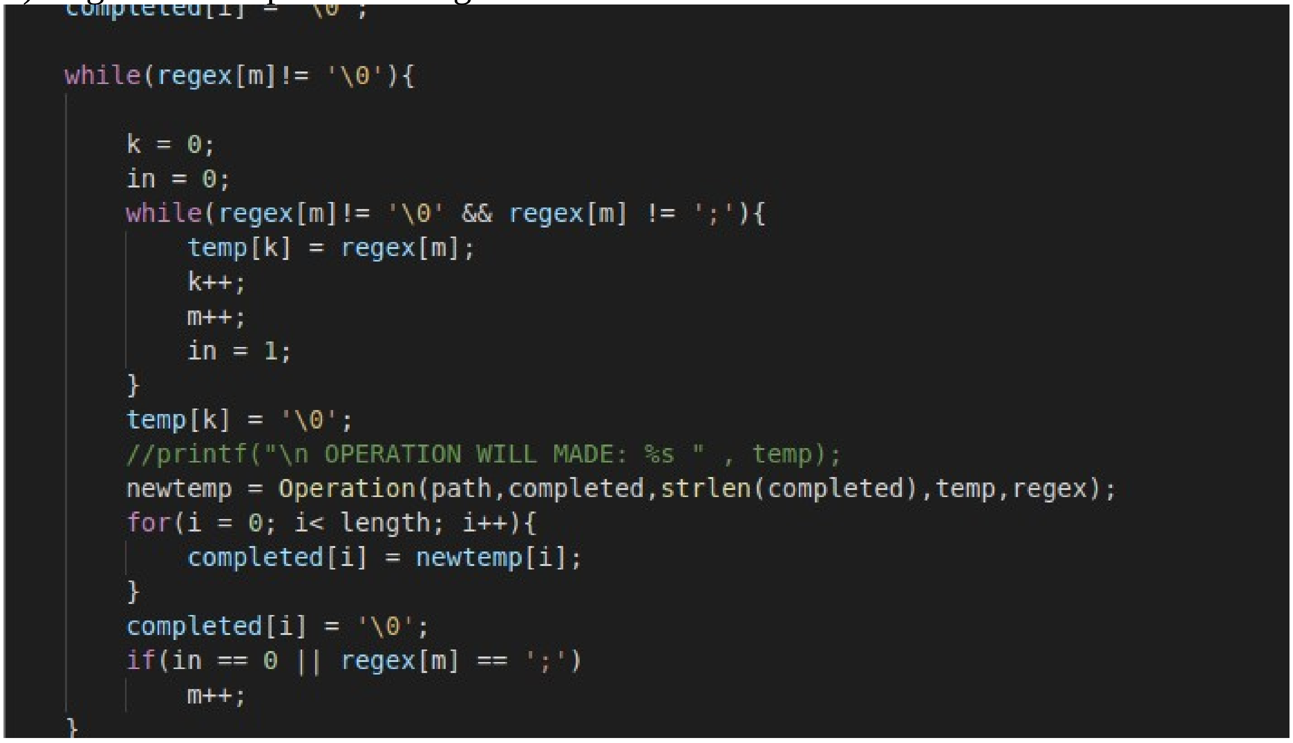
```c
void signalHandler(int signal) {
    printf("\n\n**************************************************\nSignal Caught number: %d. CTRL+C INTERRUPT Terminating
    exit(1);
}
```

3)Read the file while locking the file:



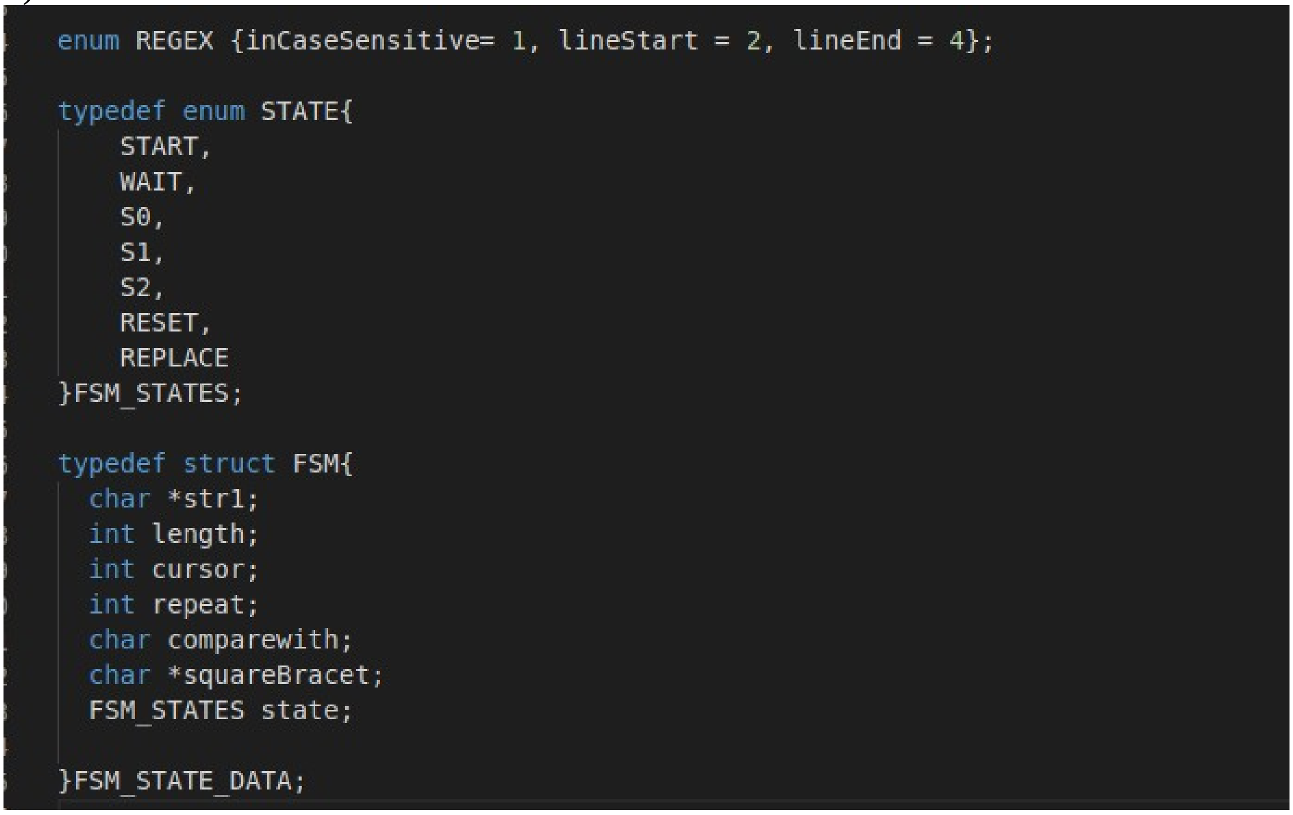
```c
ssize_t length = 0;
size_t i = 0;
char *buffer = (char*)calloc(size*2,sizeof(char));
memset(&lock,0,sizeof(lock));
lock.l_type = F_WRLCK;
if(fcntl(fd,F_SETLKW,&lock) == -1){
    perror("Error: Filelock didn't locked\n");
    _exit(EXIT_FAILURE);
}
length = read(fd, buffer ,size*2*sizeof(char));

printf ("\nLine number is %ld -> \n%s (%ld chars)\n", i++, buffer, length);

lock.l_type = F_UNLCK;
if(fcntl(fd,F_SETLKW,&lock) == -1){
    perror("Error: Filelock didn't unlocked\n");
    _exit(EXIT_FAILURE);
}
close (fd);

if(length <= 0){
    perror("\n**************************************************\nEmpty File or Reading error occured.\n**************************************************\n");
    exit(1);
}

myRegex(path,buffer, length, regex);
```

4) Regex funtion parses the regexes and creates enum values.



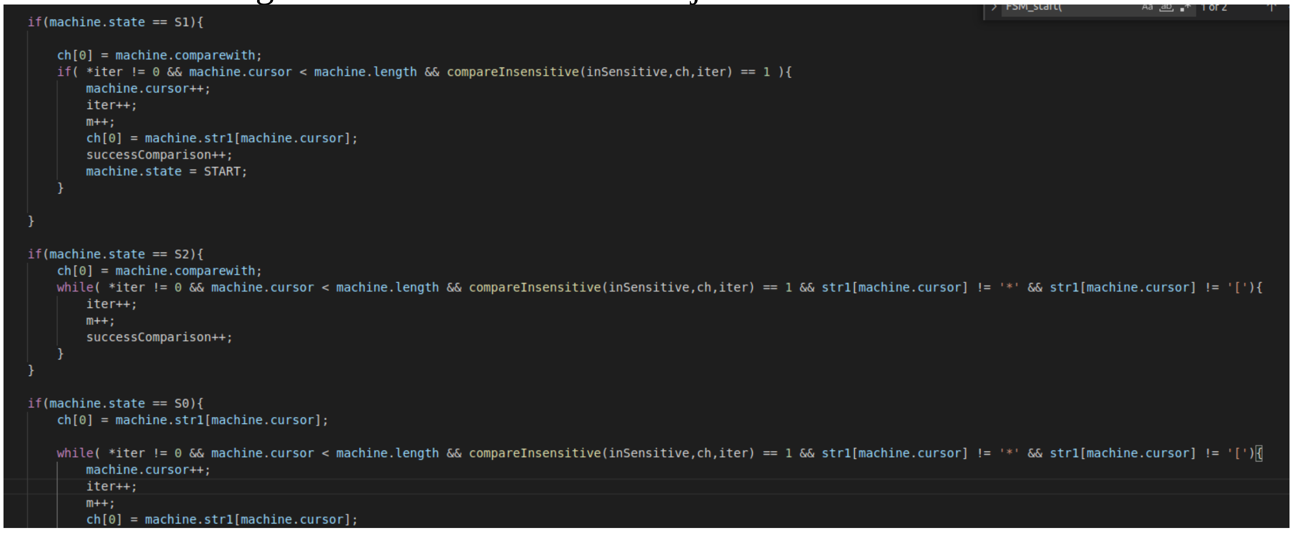
```c
    completed[i] = '\0';

    while(regex[m]!= '\0'){

        k = 0;
        in = 0;
        while(regex[m]!= '\0' && regex[m] != ';'){
            temp[k] = regex[m];
            k++;
            m++;
            in = 1;
        }
        temp[k] = '\0';
        //printf("\n OPERATION WILL MADE: %s " , temp);
        newtemp = Operation(path,completed,strlen(completed),temp,regex);
        for(i = 0; i< length; i++){
            completed[i] = newtemp[i];
        }
        completed[i] = '\0';
        if(in == 0 || regex[m] == ';')
            m++;
    }
```

5) Created FSM with structs:



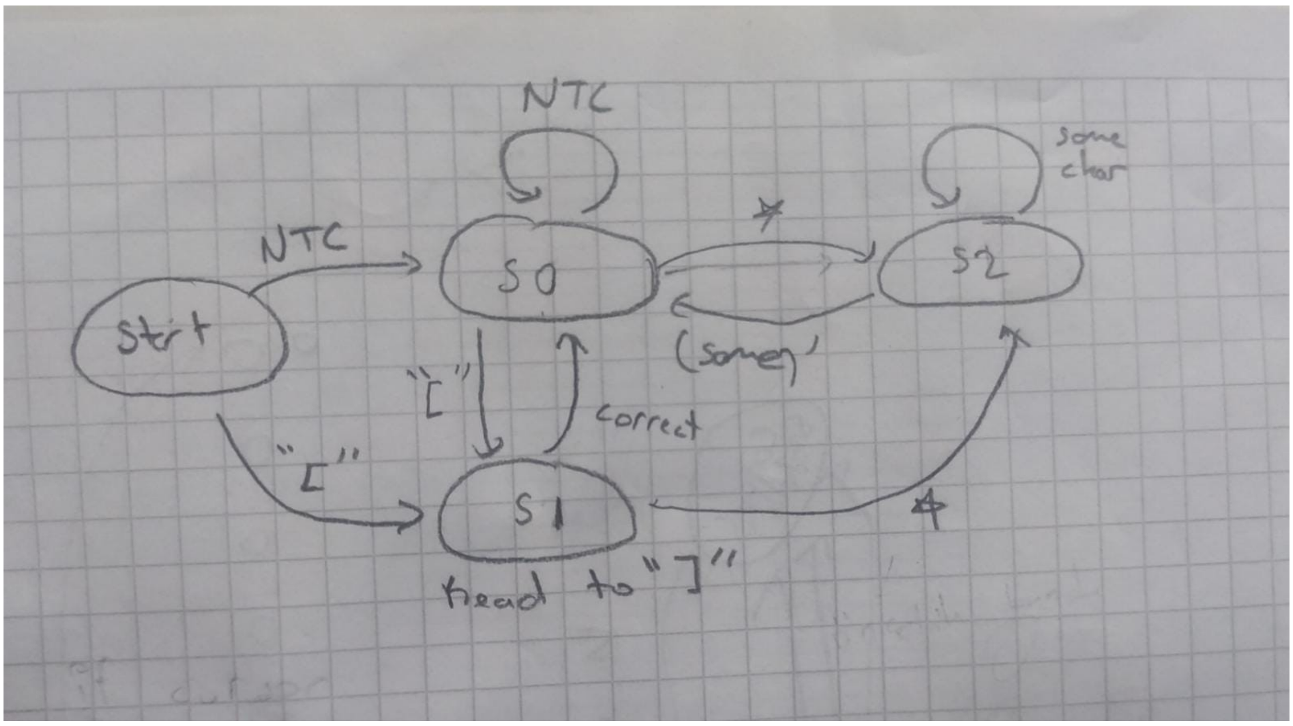
```c
enum REGEX {inCaseSensitive= 1, lineStart = 2, lineEnd = 4};

typedef enum STATE{
    START,
    WAIT,
    S0,
    S1,
    S2,
    RESET,
    REPLACE
}FSM_STATES;

typedef struct FSM{
  char *str1;
  int length;
  int cursor;
  int repeat;
  char comparewith;
  char *squareBracet;
  FSM_STATES state;

}FSM_STATE_DATA;
```

You can see as rigth below in FSM function I just used states:



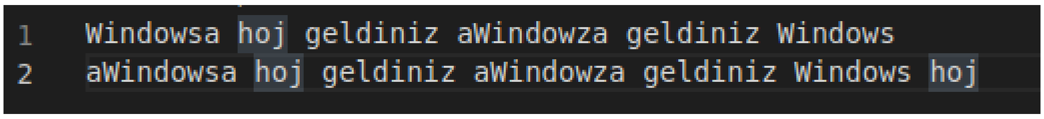
```c
if(machine.state == S1){

    ch[0] = machine.comparewith;
    if( *iter != 0 && machine.cursor < machine.length && compareInsensitive(inSensitive,ch,iter) == 1 ){
        machine.cursor++;
        iter++;
        m++;
        ch[0] = machine.str1[machine.cursor];
        successComparison++;
        machine.state = START;
    }

}

if(machine.state == S2){
    ch[0] = machine.comparewith;
    while( *iter != 0 && machine.cursor < machine.length && compareInsensitive(inSensitive,ch,iter) == 1 && str1[machine.cursor] != '*' && str1[machine.cursor] != '['){
        iter++;
        m++;
        successComparison++;
    }
}

if(machine.state == S0){
    ch[0] = machine.str1[machine.cursor];

    while( *iter != 0 && machine.cursor < machine.length && compareInsensitive(inSensitive,ch,iter) == 1 && str1[machine.cursor] != '*' && str1[machine.cursor] != '['){
        machine.cursor++;
        iter++;
        m++;
        ch[0] = machine.str1[machine.cursor];
```

Here Is my FSM:



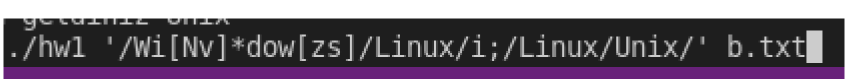
TEST SECTION

TEST 1: Operation is trying multiple combinations
File 1:



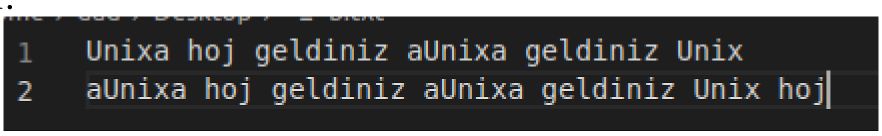
```
Windowsa hoj geldiniz aWindowza geldiniz Windows
aWindowsa hoj geldiniz aWindowza geldiniz Windows hoj
```

Input 1:



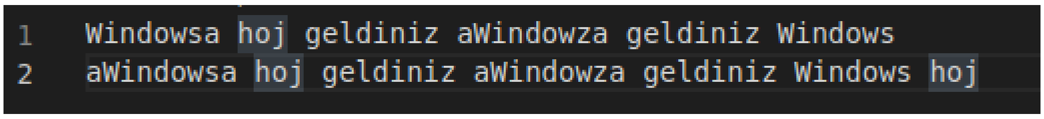
```
./hw1 '/Wi[Nv]*dow[zs]/Linux/i;/Linux/Unix/' b.txt
```

Output 1:



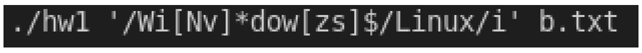
```
Unixa hoj geldiniz aUnixa geldiniz Unix
aUnixa hoj geldiniz aUnixa geldiniz Unix hoj
```

TEST 2: Operation is trying multiple combinations with $ character so we can see just end of the macthing regex changed
File 2:



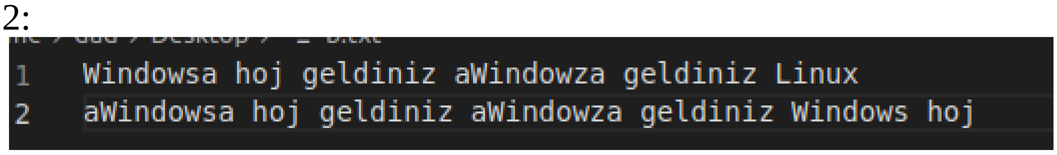
```
Windowsa hoj geldiniz aWindowza geldiniz Windows
aWindowsa hoj geldiniz aWindowza geldiniz Windows hoj
```

Input 2:



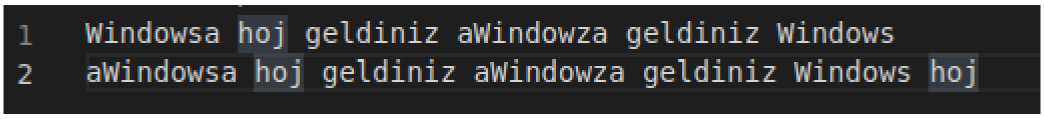
```
./hw1 '/Wi[Nv]*dow[zs]$/Linux/i' b.txt
```

Output 2:



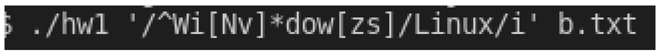
```
Windowsa hoj geldiniz aWindowza geldiniz Linux
aWindowsa hoj geldiniz aWindowza geldiniz Windows hoj
```

TEST 3: Operation is trying multiple combinations with ^ character so we can see just lines first of the macthing regex changed
File 3:



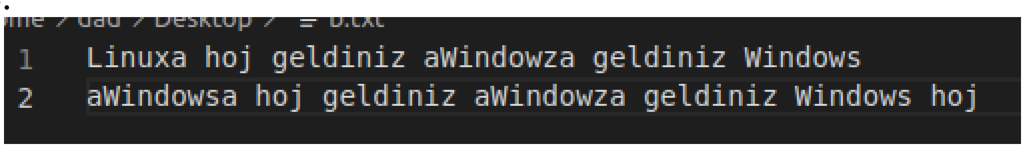
```
Windowsa hoj geldiniz aWindowza geldiniz Windows
aWindowsa hoj geldiniz aWindowza geldiniz Windows hoj
```

Input 3:



```
$ ./hw1 '/^Wi[Nv]*dow[zs]/Linux/i' b.txt
```

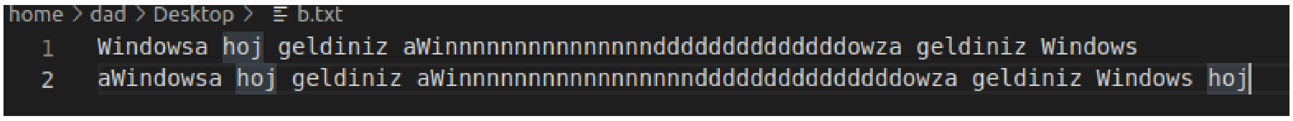
Output 3:



```
Linuxa hoj geldiniz aWindowza geldiniz Windows
aWindowsa hoj geldiniz aWindowza geldiniz Windows hoj
```

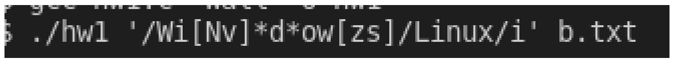
TEST 4:   Operation is trying multiple occurance when * comes even if [] comes before

File 4:



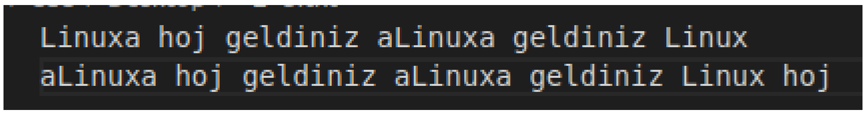
```
home > dad > Desktop > b.txt
Windowsa hoj geldiniz aWinnnnnnnnnnnnnnnddddddddddddddowza geldiniz Windows
aWindowsa hoj geldiniz aWinnnnnnnnnnnnnnnnnddddddddddddddddowza geldiniz Windows hoj
```

Input 4:



```
$ ./hw1 '/Wi[Nv]*d*ow[zs]/Linux/i' b.txt
```

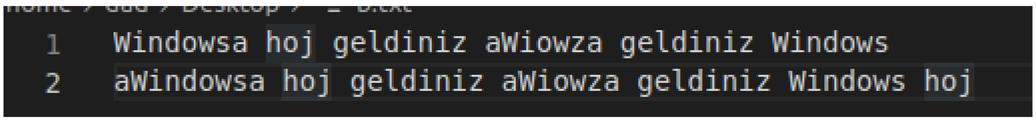
Output 4:



```
Linuxa hoj geldiniz aLinuxa geldiniz Linux
aLinuxa hoj geldiniz aLinuxa geldiniz Linux hoj
```

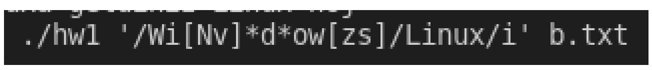
TEST 5:   Operation is trying no occurance when * comes even if [] comes before

File 5:



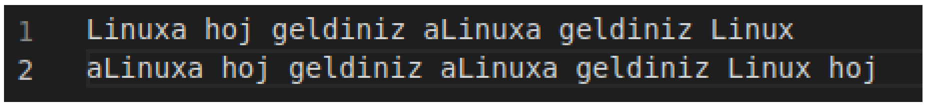
```
Windowsa hoj geldiniz aWiowza geldiniz Windows
aWindowsa hoj geldiniz aWiowza geldiniz Windows hoj
```

Input 5:



```
./hw1 '/Wi[Nv]*d*ow[zs]/Linux/i' b.txt
```

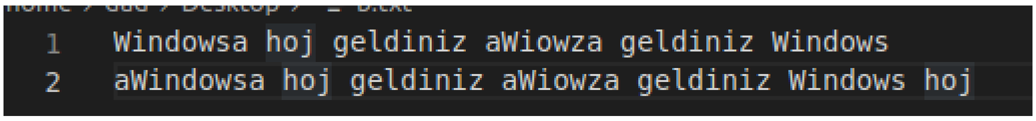
Output 5:



```
Linuxa hoj geldiniz aLinuxa geldiniz Linux
aLinuxa hoj geldiniz aLinuxa geldiniz Linux hoj
```

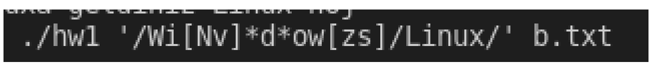
TEST 6:   Operation is trying Case sensitivity even if inside this []

File 6:



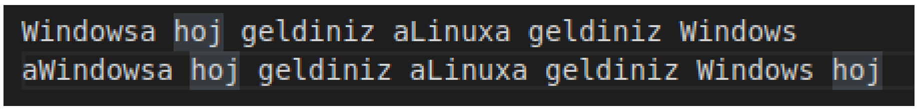
```
Windowsa hoj geldiniz aWiowza geldiniz Windows
aWindowsa hoj geldiniz aWiowza geldiniz Windows hoj
```

Input 6:



```
./hw1 '/Wi[Nv]*d*ow[zs]/Linux/' b.txt
```

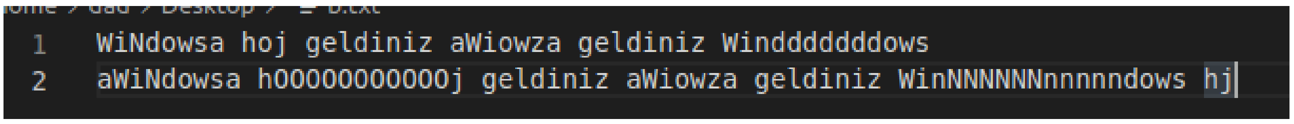
Output 6:



```
Windowsa hoj geldiniz aLinuxa geldiniz Windows
aWindowsa hoj geldiniz aLinuxa geldiniz Windows hoj
```

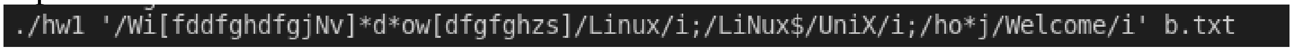
TEST 7: [ ] with multiple words

File 7:



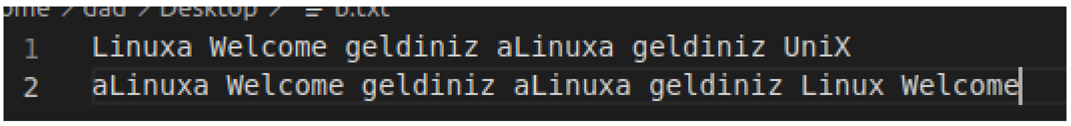
```
WiNdowsa hoj geldiniz aWiowza geldiniz Windddddddows
aWiNdowsa hOOOOOOOOOOOj geldiniz aWiowza geldiniz WinNNNNNNnnnnndows hj
```

Input 7:



```
./hw1 '/Wi[fddfghdfgjNv]*d*ow[dfgfghzs]/Linux/i;/LiNux$/UniX/i;/ho*j/Welcome/i' b.txt
```

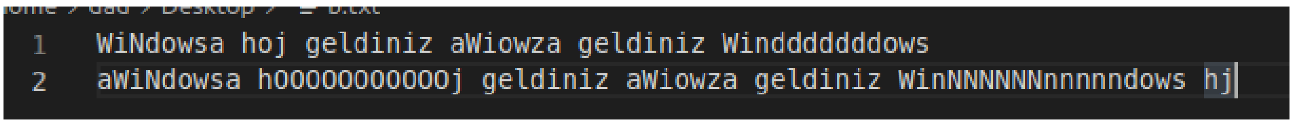
Output 7:



```
Linuxa Welcome geldiniz aLinuxa geldiniz UniX
aLinuxa Welcome geldiniz aLinuxa geldiniz Linux Welcome
```

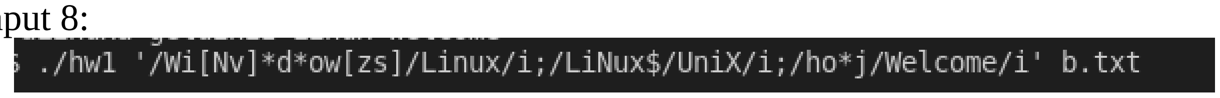
TEST 8: EVERY CASE ADDED

File 8:



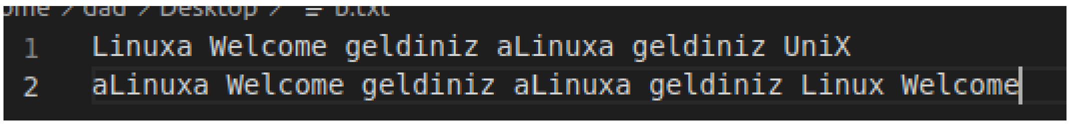
```
WiNdowsa hoj geldiniz aWiowza geldiniz Windddddddows
aWiNdowsa hOOOOOOOOOOOj geldiniz aWiowza geldiniz WinNNNNNNnnnnndows hj
```

Input 8:



```
$ ./hw1 '/Wi[Nv]*d*ow[zs]/Linux/i;/LiNux$/UniX/i;/ho*j/Welcome/i' b.txt
```

Output 8:



```
Linuxa Welcome geldiniz aLinuxa geldiniz UniX
aLinuxa Welcome geldiniz aLinuxa geldiniz Linux Welcome
```