CSE 344
SYSTEM PROGRAMMING
HW1

MUSTAFA GURLER
171044034

REPORT

At first I tried to solve the problems one by one, but there was confusion. I had to use a nested structure to solve the problems.

First I had to check the semicolons to separate the received argument, then I divided it into 3 parts according to the / character and sent it to the read_from_fd function. I opened the file name I put into the function with the open method and assigned it to fd. I decided to keep the fd in a temporary array by reading it with the read function. My aim was to go through the array and see that when each character is matched, the rest of it matches in the function and compare it according to the length of the first word received. This idea could be valid for the first, second and third options. Just checking that the 3rd letter is i where I split it into 3 parts by case insensitivity / character. If this is true, I was checking with the or operation that it is -32 or +32 according to the ASCII characters. Since numbers are not case sensitive anyway, I did not apply these operations to them.

```
int consist_number(unsigned char* buffer, int it){

    for(int i=0 ; i<10 ; i++){
        if(buffer[it] == 48+i){
            return 1;
        }
    }
    return 0;
}
```

My logic in the square brackets was a little different. If a word contains square brackets, it must contain more than one word thanks to the letters in the square brackets. I took the words there one by one and put them into my function. If he could match one anyway, he wouldn't match the others. I can easily do these operations with square brackets that I know the starting points of. If the return value is greater than zero, I advance my buffer as much as the word I searched for where I entered. This way, I don't look again at the places I changed.

```
if (consist_of_paranthesis_find(parsed_string[0]) == 1){
    start_number_paranthesis = consist_number_of_paranthesis_start(parsed_string[0])+1;
    end_number_paranthesis = consist_number_of_paranthesis_end(parsed_string[0])-1;
    char temp_array[100];

    int temp_array_it = 0;

    int value = end_number_paranthesis - start_number_paranthesis + 1;

    for(int j=0 ; j<value ; j++){
        s=0;
        temp_array_it = 0;

        for(int k=0 ; k<start_number_paranthesis-1 ; k++){
            temp_array[temp_array_it] = parsed_string[0][k];
            temp_array_it++;
        }

        temp_array[temp_array_it] = parsed_string[0][start_number_paranthesis+j];
        temp_array_it++;

        for(int k=end_number_paranthesis+2 ; k<strlen(parsed_string[0]) ; k++){

            temp_array[temp_array_it] = parsed_string[0][k];
            temp_array_it++;
        }

        part_a_result = part_a_start(buffer, i, temp_array, 0, parsed_string[2], enter);
        if(part_a_result > 0){
            paranthes_length = value+1;
            break;
        }
    }
    if(part_a_result > 0){

        if(part_a_result == 1){
            if(enter == 1 || enter == 2){
                i = i - 1;
            }
```

When reading a star, I match it normally from the star to the value before it. After that, I count how many values there are in my buffer, then I advance those values as the iterator of my buffer. In this way, I can advance directly, regardless of how much the value is.

```
int asterix_flag = -1;
int value = -1;
int asterix_number = -1;
if(consist_of_asterix_for_string(parsed_string) > 0){
    asterix_flag = 1;
    value = consist_of_asterix_for_string(parsed_string);

}

if(value-1 == string_it && buffer[it] == parsed_string[string_it]){
    asterix_number = find_of_asterix_number(buffer, it, parsed_string, value);

    string_it = value+1;
    it = it + asterix_number;
}
```

The dollar sign and the upper sign are essentially the same logic. I was able to do the up sign but didn't have time to do the dollar sign. I looked at the probability that if the word contains the superscript and the word in front of the buffer is 10 or 13 based on ASCII characters. After that, it was easy. By checking the flag I assigned, I increase the iterator of the word in the first letter of the word. The next one is already proceeding in the same way as the first option.

```
        enter=0;
        if(find_of_began_line_string(parsed_string[0]) >= 0){

            if(buffer[i-1] == 10 || buffer[i-1] == 13 || i == 0){
                enter = 1;
            }
        }

        if(find_of_end_line_string(parsed_string[0]) >= 0 || i == bytes_read){
            enter = 2;
        }
        int s=0;
```

Generally, I tried each option one by one and together in order to work together. There were some logic errors and I had to correct them. But I managed to do all the others except the dollar sign.

The lock operation seemed simple at first, but my only problem was that I had to write the lock part twice, since I performed the open and close operations twice to delete the file and write it again. Examples run on different terminals may cause errors. However, an example running in a single terminal does not cause an error.

```
lock.l_type = F_UNLCK;

fcntl(fd, F_SETLKW, &lock);

if(close(fd) == -1){
    perror("error");
}

fd = open(fileName, O_RDWR | O_CREAT | O_TRUNC, S_IRWXU );



if(fd == -1){
    perror("error");
}

memset(&lock, 0, sizeof(lock));
lock.l_type = F_WRLCK;

fcntl(fd, F_SETLKW, &lock);

write(fd, array, count);

lock.l_type = F_UNLCK;

fcntl(fd, F_SETLKW, &lock);

if(close(fd) == -1){
    perror("error");
}
```

```
FILE DESCRIPTORS: 3 open at exit.
Open file descriptor 2: /dev/pts/0
   <inherited from parent>

Open file descriptor 1: /dev/pts/0
   <inherited from parent>

Open file descriptor 0: /dev/pts/0
   <inherited from parent>


HEAP SUMMARY:
    in use at exit: 0 bytes in 0 blocks
  total heap usage: 6 allocs, 6 frees, 25 bytes allocated

All heap blocks were freed -- no leaks are possible
```

After that, the error checking part came. I cleared the warnings with the -Wall flag. Some unused parts I made while writing the code. I checked for memory leaks with Valgrind. I watched valgrind by freeing 6 malloc functions that I created in total. Finally, after I wrote No memory leak information, I also took care of this part.