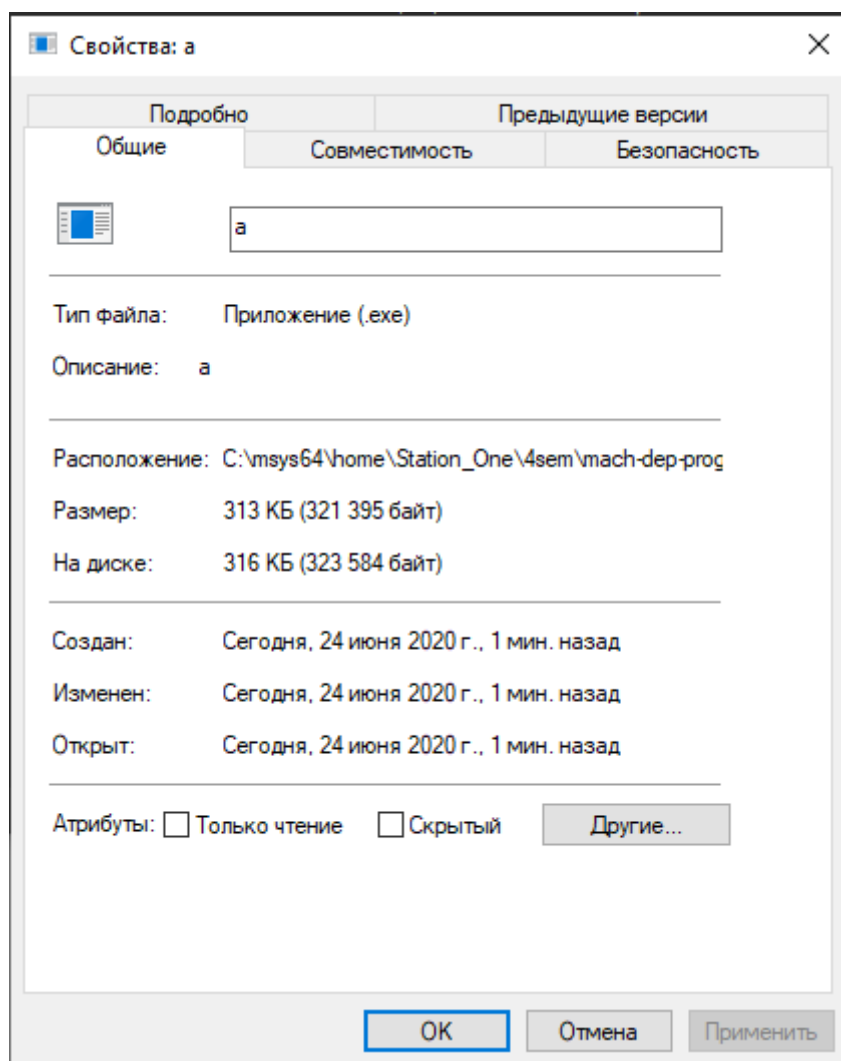


Для начала рассмотрим сам файл, который необходимо отреверсить:

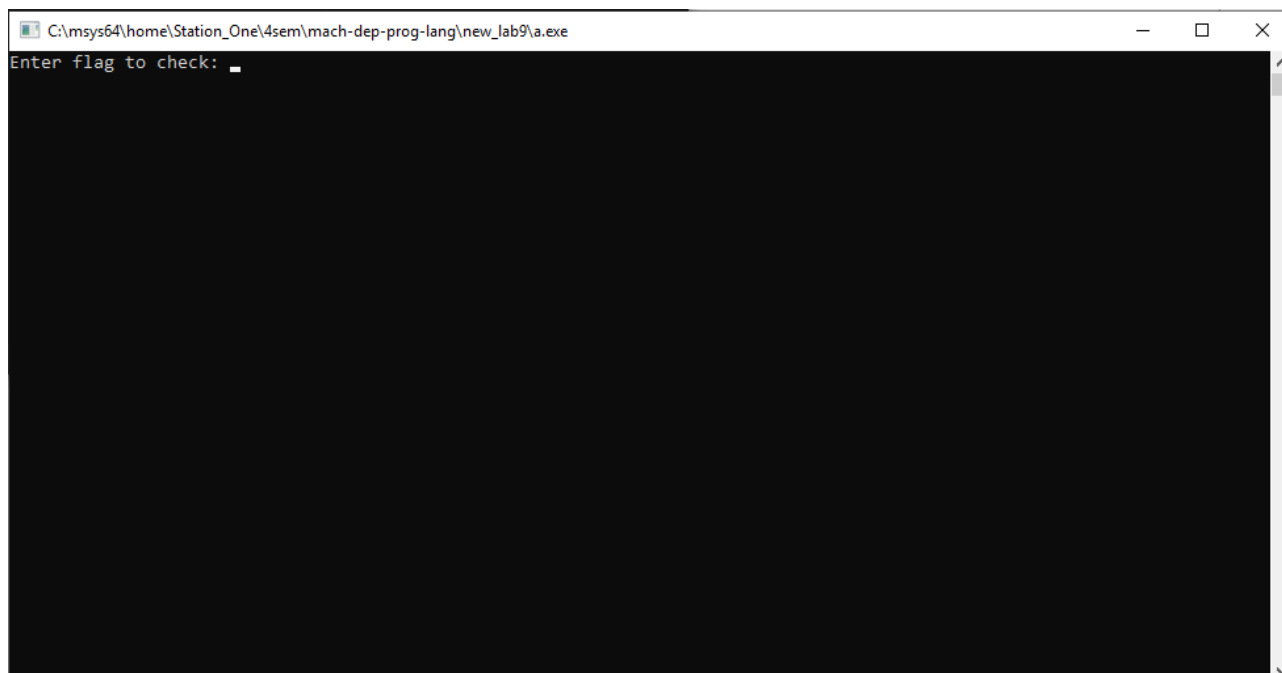
a.exe



Ничего необычного.

Попробуем запустить и понять, что от нас требуется:

Строка приглашения говорит о наличии флага, который нужно ввести для проверки.



При вводе информации нам сообщают неприятную новость:

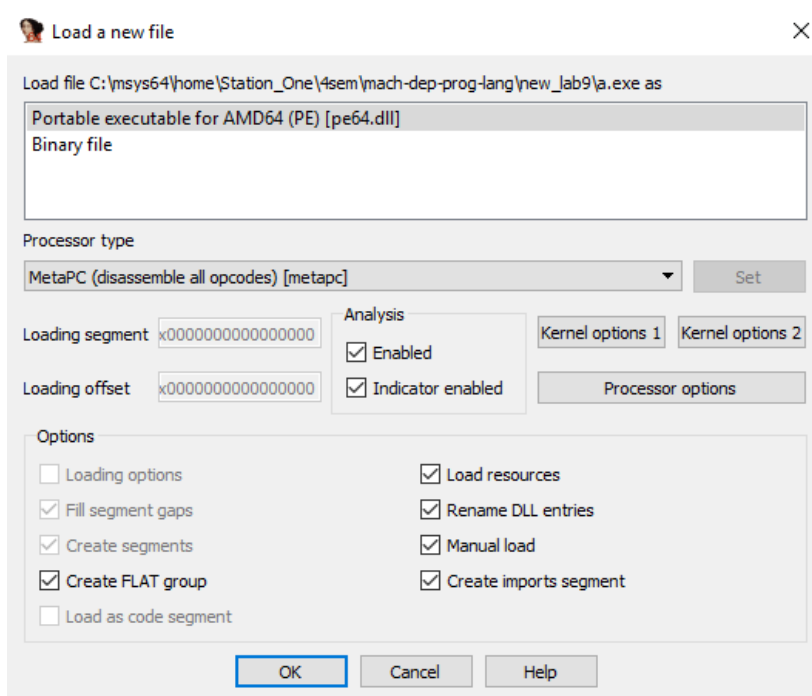
```
C:\msys64\home\Station_One\4sem\mach-dep-prog-lang\new_lab9>a.exe
Enter flag to check: r
Wrong position 0!

C:\msys64\home\Station_One\4sem\mach-dep-prog-lang\new_lab9>a.exe
Enter flag to check: qewe
Wrong position 0!

C:\msys64\home\Station_One\4sem\mach-dep-prog-lang\new_lab9>a.exe
Enter flag to check: anime
Wrong position 0!

C:\msys64\home\Station_One\4sem\mach-dep-prog-lang\new_lab9>_
```

Ладно, видимо придется открыть IDA Freeware



```

; Attributes: bp-based frame fpd=0D0h

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

Str= byte ptr -12Fh
var_127= dword ptr -127h
var_123= word ptr -123h
var_121= byte ptr -121h
var_120= byte ptr -120h
var_14= dword ptr -14h

push    rbp
push    rbx
sub     rsp, 148h
lea     rbp, [rsp+80h]
call    __main
mov     rax, 7A6F646C77683266h
mov     qword ptr [rbp+0D0h+Str], rax
mov     [rbp+0D0h+var_127], 773A7C67h
mov     [rbp+0D0h+var_123], 7162h
mov     [rbp+0D0h+var_121], 0
lea     rcx, Format          ; "Enter flag to check: "
call    printf
lea     rax, [rbp+0D0h+var_120]
mov     rdx, rax
lea     rcx, aS              ; "%s"
call    scanf
mov     [rbp+0D0h+var_14], 0
jmp     short loc_40160E

```

```

loc_40160E:
mov     eax, [rbp+0D0h+var_14]
movsxd  rbx, eax
lea     rax, [rbp+0D0h+Str]
mov     rcx, rax             ; Str
call    strlen
cmp     rbx, rax
jb      short loc_4015C0

```

```

loc_4015C0:
mov     eax, [rbp+0D0h+var_14]

```

jmp short loc_4015C0

```

loc_4015C0:
mov     eax, [rbp+0D0h+var_14]
cdq     eax
movzx   eax, [rbp+rax+0D0h+var_120]
movsx   edx, al
mov     eax, [rbp+0D0h+var_14]
add     edx, eax
mov     eax, [rbp+0D0h+var_14]
cdq     eax
movzx   eax, [rbp+rax+0D0h+Str]
movsx   eax, al
cmp     edx, eax
jz      short loc_401607

```

```

mov     eax, [rbp+0D0h+var_14]
mov     edx, eax
lea     rcx, aWrongPositionD ; "Wrong position %d!\n"
call    printf
mov     eax, 0
jmp     short loc_401640

```

```

lea     rax, [rbp+0D0h+var_120]
mov     rdx, rax
lea     rcx, aYesCorrectFlag ; "Yes! Correct flag is %s\n"
call    printf
mov     eax, 0

```

```

loc_401607:
add     [rbp+0D0h+var_14], 1

```

```

loc_401640:
add     rsp, 148h
pop     rbx
pop     rbp
retn
main endp

```

Первое, что бросается в глаза — то, что флаг может быть верным. Это уже хорошо

В первом блоке мы видим строку приглашения

```
mov     [rbp+0D0h+var_120], 0
lea     rcx, Format      ; "Enter flag to check: "
call    printf
lea     rax, [rbp+0D0h+var_120]
mov     rdx, rax
lea     rcx, aS          ; "%s"
call    scanf
mov     [rbp+0D0h+var_14], 0
jmp     short loc_40160E
```

Во втором вызов strlen, что как бы намекает на получение длины строки

```
loc_40160E:
mov     eax, [rbp+0D0h+var_14]
movsxd  rbx, eax
lea     rax, [rbp+0D0h+Str]
mov     rcx, rax          ; Str
call    strlen
cmp     rbx, rax
jb      short loc_4015C0
```

Заметим и то, что видимо есть какая-то зависимость от введенной информации: ошибки обрабатываются по разному — значит есть вероятность найти несколько ошибочных ситуаций

```
mov     eax, [rbp+0D0h+var_14]
mov     ecx, eax
lea     rcx, aWrongPositionD ; "Wrong position %d!\n"
call    printf
mov     eax, 0
jmp     short loc_401640

lea     rax, [rbp+0D0h+var_120]
mov     rdx, rax
lea     rcx, aYesCorrectFlag ; "Yes! Correct flag is %s\n"
call    printf
mov     eax, 0

loc_401640:
add     [rbp+0D0h+var_14], 1
```

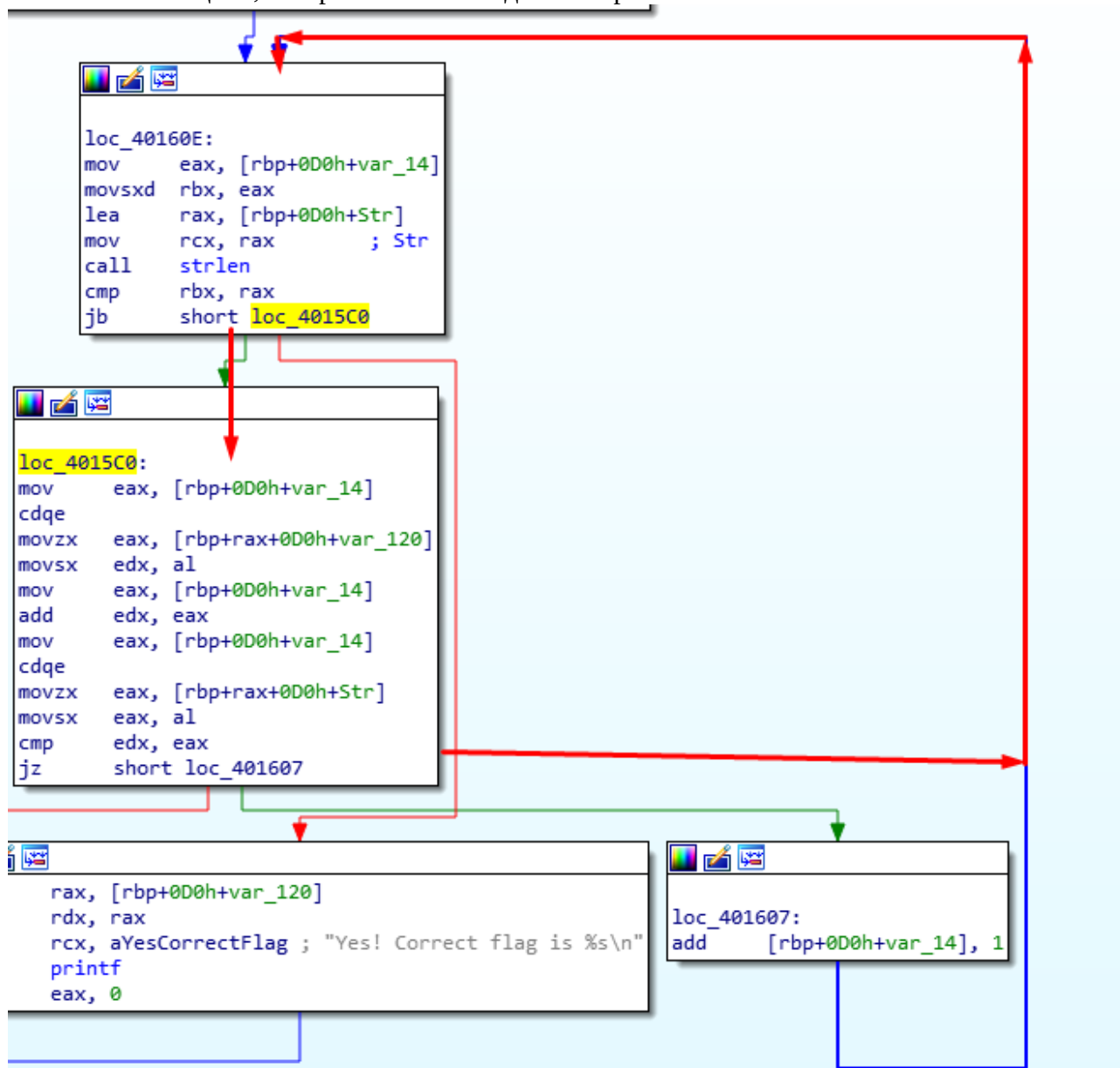
Посмотрим, что там в памяти

```
); char Format[]
Format      db 'Enter flag to check: ',0
; DATA XREF: HEADER:00000000004001E4fo
; main+35fo
); char aS[]
aS          db '%s',0
; DATA XREF: main+48fo
); char aWrongPositionD[]
aWrongPositionD db 'Wrong position %d!',0Ah,0
; DATA XREF: main+94fo
); char aYesCorrectFlag[]
aYesCorrectFlag db 'Yes! Correct flag is %s',0Ah,0
; DATA XREF: main+CFfo
```

Строки. Что самое интересное — `char aS[]` выходит та строка, которую мы заполняем

Установим пока точки останова в начале, где происходит ввод строки и в конце, где находится `ret` процедуры `main`

Так же заметен цикл, который зависит от длины строки



Осмелюсь предположить, что в таком случае у нас имеется обработка строки вот каким образом

```
for (i = 0; i < strlen( некая строка ); i++) {  
    if (условие) {  
        Wrong position %d!\n  
    }  
}
```

Yes! Correct flag is %s\n

Так как есть указание, что позиция неверна, можно предположить, что там вставляется индекс, указывающий на неверную позицию

Значит существует некоторая строка, с которой программа сверяет наши введенные данные

lea позволяет вычислять адреса, которые получаются в результаты выполнения арифметических операций

cdqe = RAX ← sign-extend EAX

Преобразование EAX в RAX

movsxd R*X, E*X – перемещение в R*X из E*X с расширением

movzx – переслать с расширением нулями

movsx – тот же самый mov, только с расширением значения до атрибута размера операнда и сохраняет в регистре назначении

Пример адресации в массиве для получения некоторого элемента:

```
int foo(char * buf, int index) {  
    return buf[index];  
}
```

Оно же:

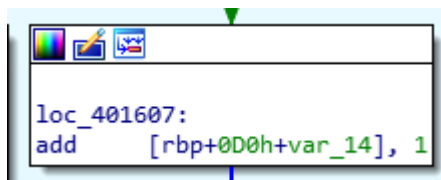
```
push    rbp  
mov     rbp, rsp  
mov     qword ptr [rbp - 8], rdi  
mov     dword ptr [rbp - 12], esi  
mov     rax, qword ptr [rbp - 8] ; rax is buf  
movsxd  rcx, dword ptr [rbp - 12] ; rcx is index  
movsx   eax, byte ptr [rax + rcx] ; store buf[index] into eax  
pop     rbp  
ret
```

var_14 – ничто иное, как индекс, по которому адресуемся в строке

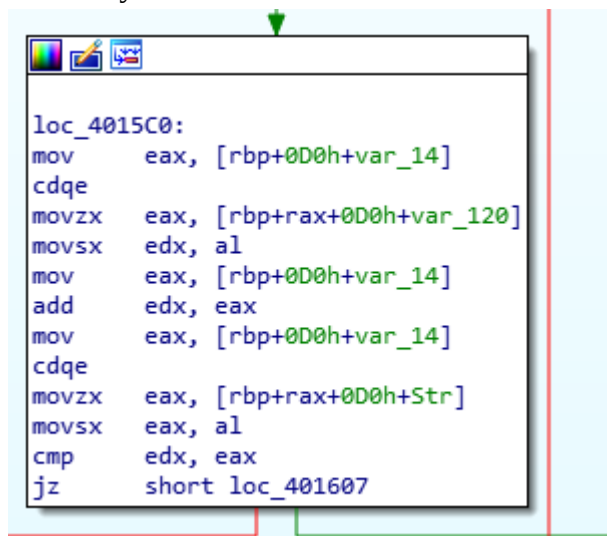
Выходит так, что мы пробегаем по вводимой строке до момента, пока встроенная строка не закончится.

var_14 – ничто иное, как индекс (адрес) , по которому адресуемся в строке

Причем каждый раз итерируем:



А вот тут:



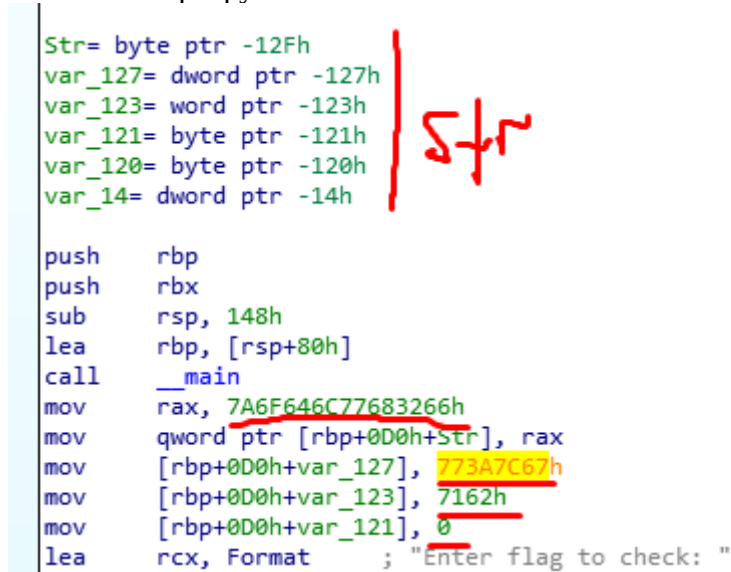
```
loc_4015C0:
mov     eax, [rbp+0D0h+var_14]
cdqe
movzx   eax, [rbp+rax+0D0h+var_120]
movsx   edx, al
mov     eax, [rbp+0D0h+var_14]
add     edx, eax
mov     eax, [rbp+0D0h+var_14]
cdqe
movzx   eax, [rbp+rax+0D0h+Str]
movsx   eax, al
cmp     edx, eax
jz      short loc_401607
```

Выходит так, что мы сравниваем значения:

наша строка [i] + i и встроенная строка[i]

Осталось получить эту строку, что справа

А она в самом начале прогружается в стек:



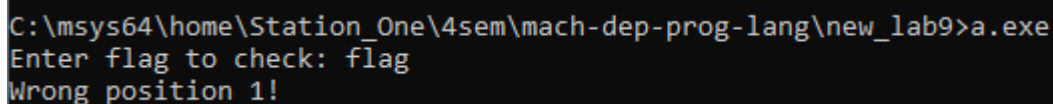
```
Str= byte ptr -12Fh
var_127= dword ptr -127h
var_123= word ptr -123h
var_121= byte ptr -121h
var_120= byte ptr -120h
var_14= dword ptr -14h

push    rbp
push    rbx
sub     rsp, 148h
lea     rbp, [rsp+80h]
call    __main
mov     rax, 7A6F646C77683266h
mov     qword ptr [rbp+0D0h+Str], rax
mov     [rbp+0D0h+var_127], 773A7C67h
mov     [rbp+0D0h+var_123], 7162h
mov     [rbp+0D0h+var_121], 0
lea     rcx, Format ; "Enter flag to check: "
```

Если перевести в ASCII – получим f2hwldozg|:wbq

Вот тут уже немного нужно подумать

Из-за того, что на ранней стадии индекс равен 0, то можем смело вставлять f как первый символ флага



```
C:\msys64\home\Station_One\4sem\mach-dep-prog-lang\new_lab9>a.exe
Enter flag to check: flag
Wrong position 1!
```

Бинго. Позиция поменялась!

Значит нам нужно теперь будет подобрать такой флаг

Тогда

$f - 0 = f$

$2 - 1 = 1$

$h - 2 = \dots$

...

$w - 11 = \dots$

$b - 12 = \dots$

$q - 13 = \dots$

Получим вот такой flag = f1fth_is_s0lVd

```
C:\msys64\home\Station_One\4sem\mach-dep-prog-lang\new_lab9>a.exe
Enter flag to check: flag
Wrong position 2!
```

Подставим:

```
C:\msys64\home\Station_One\4sem\mach-dep-prog-lang\new_lab9>a.exe
Enter flag to check: f1fth_is_s0lVd
Yes! Correct flag is f1fth_is_s0lVd
```

Мы смогли, ура