Alex Zaharia

Task 1

main

first	111	0x7fffffffe50c
Second	222	0x7fffffffe51c
P	0x7fffffffe50c	0x7fffffffe510

Start →

function_one

~~input~~	~~357~~	~~0x7fffffffe4dc~~
~~function_one_local~~	~~357~~	~~0x7fffffffe4ec~~

function_two

input	0	0x7fffffffe4dc
function_two_local	0	0x7fffffffe4ec

Pause →

NOTE

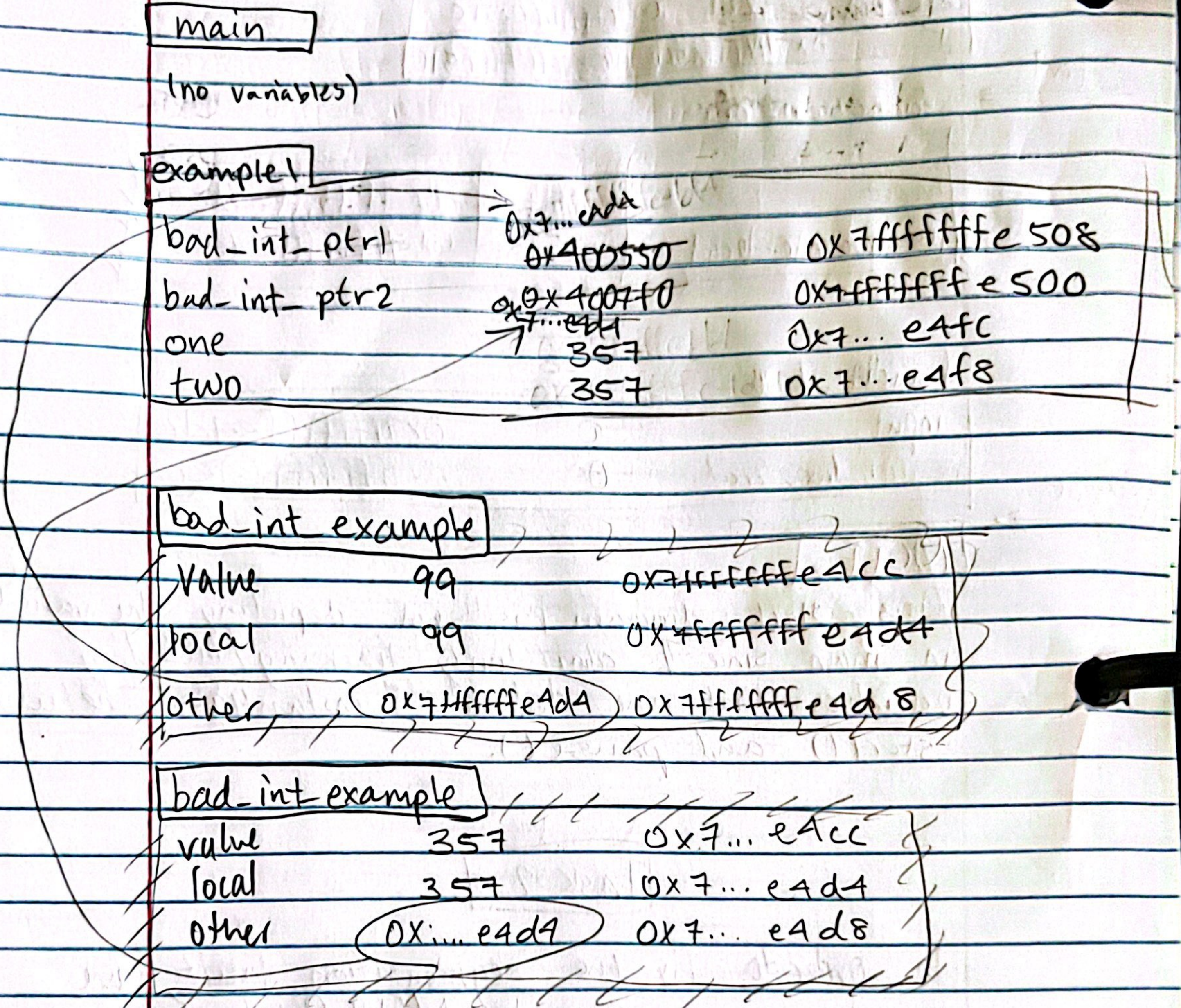
This stack diagram is not depicting the variables in main since I am only tracking / including variables that are declared & instantiated between start() and pause().

Task 2

In order to fix the segmentation fault we must make sure to NEVER dereference NULL pointers.

```
... if (p == NULL) {
        printf("Error! null pointer");
    } else {
        *p = 0;
    }
```

Task 3



main

(no variables)

example1

bad_int_ptr1	0x7...e4d4 ~~0x400550~~	0x7fffffffe508
bad_int_ptr2	~~0x400710~~ 0x7...e4d4	0x7fffffffe500
one	357	0x7... e4fc
two	357	0x7... e4f8

bad_int_example

value	99	0x7fffffffe4cc
local	99	0x7fffffffe4d4
other	0x7fffffffe4d4	0x7fffffffe4d8

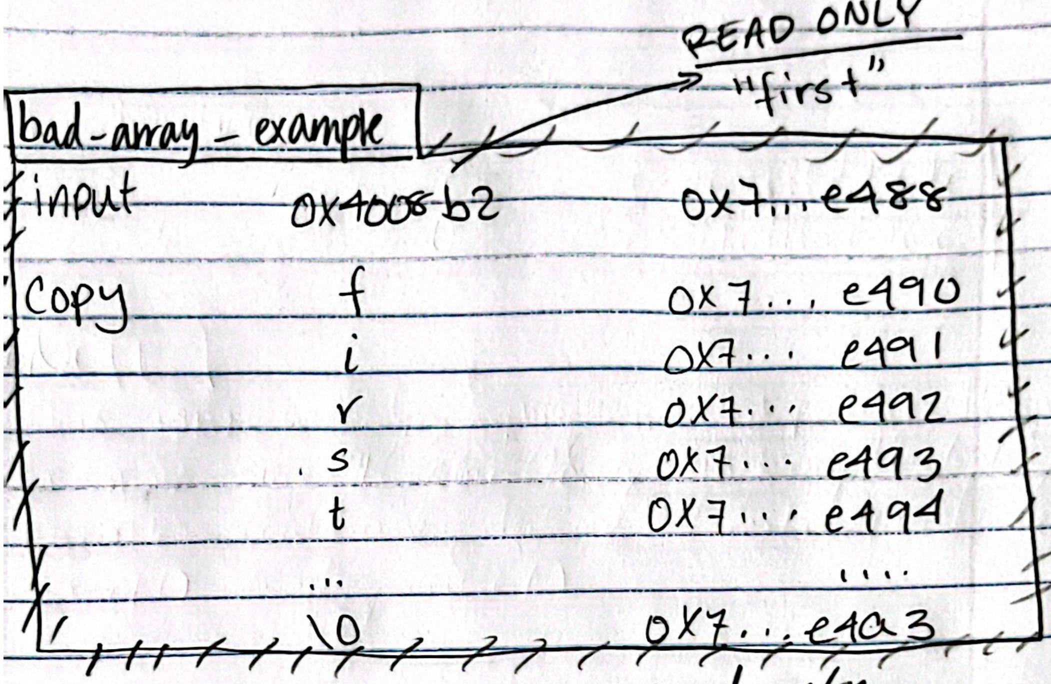
bad_int_example

value	357	0x7... e4cc
local	357	0x7... e4d4
other	0x:... e4d4	0x7... e4d8

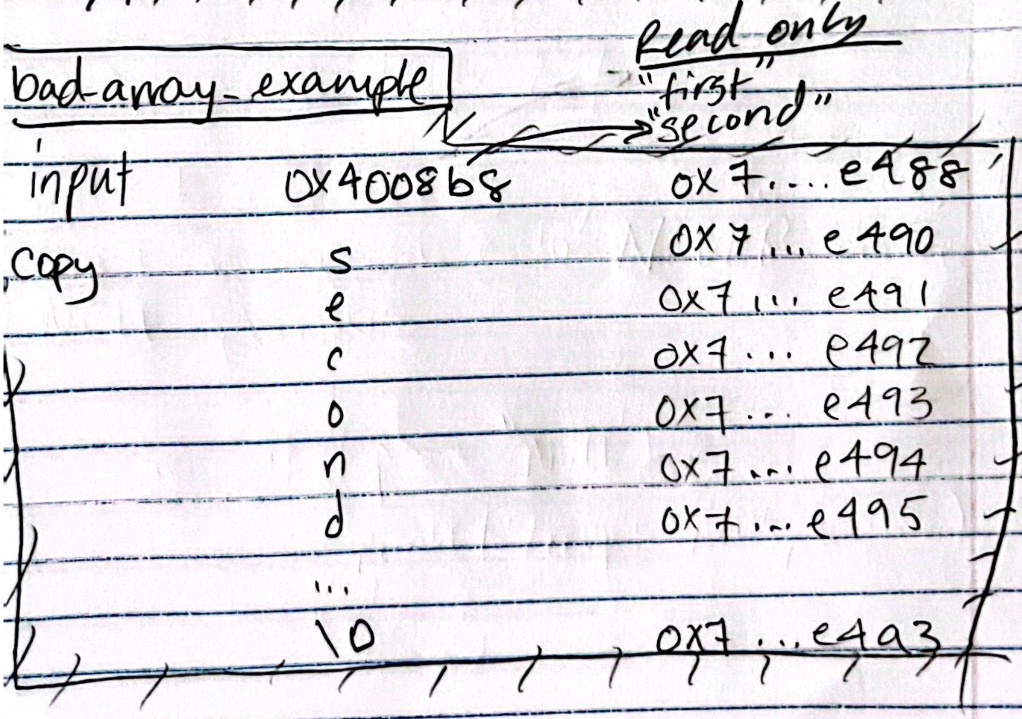
☆ For function example1, since we are popping the bad_int_example frame after running it for the first time, the memory addresses used here are overridden by the next time this function gets called. We therefore lose value '99' and variables 'one' and 'two' will now be the same and both will hold '357'

example 2

bad_string1		0X7...e4e0
bad_string2		0X7... e4c0
bad_string_result1	0X7...e490	0X7... e508
bad_string_result2		0X7...e500



input	0X4008b2	0x7...e488
copy	f	0x7... e490
	i	0X7... e491
	r	0X7... e492
	s	0X7... e493
	t	0X7... e494
	...	....
	\0	0X7...e4a3



input	0X4008b8	0x7....e488
copy	s	0X7... e490
	e	0x7... e491
	c	0x7... e492
	o	0x7... e493
	n	0x7... e494
	d	0x7... e495
	...	
	\0	0x7...e4a3

For function example2, since we are running the bad-array-example twice, it is overwriting the "first" value from being properly stored. Similar to what is happening in example 1.