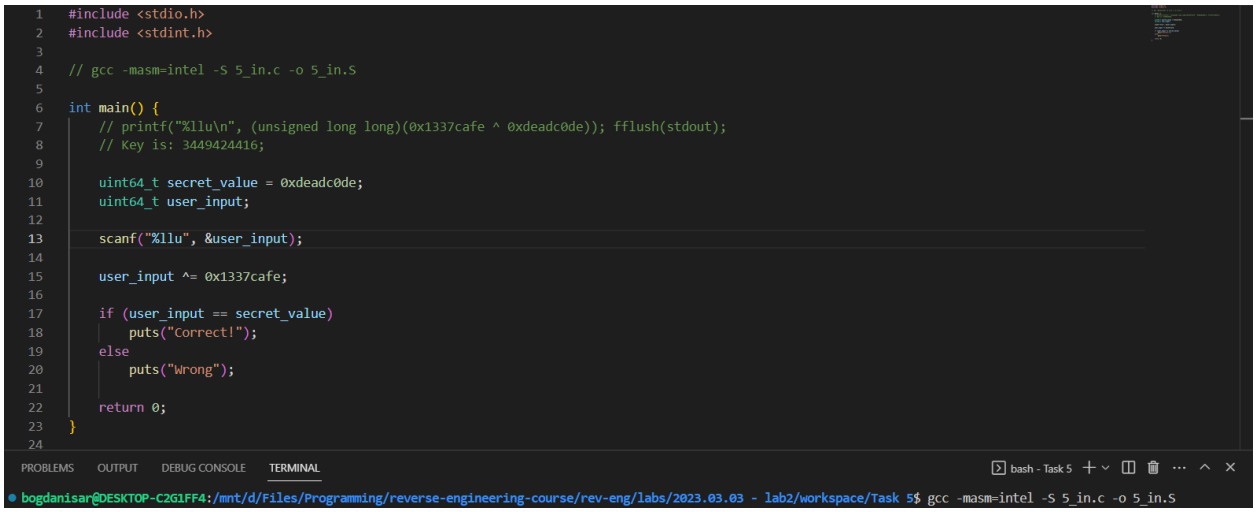
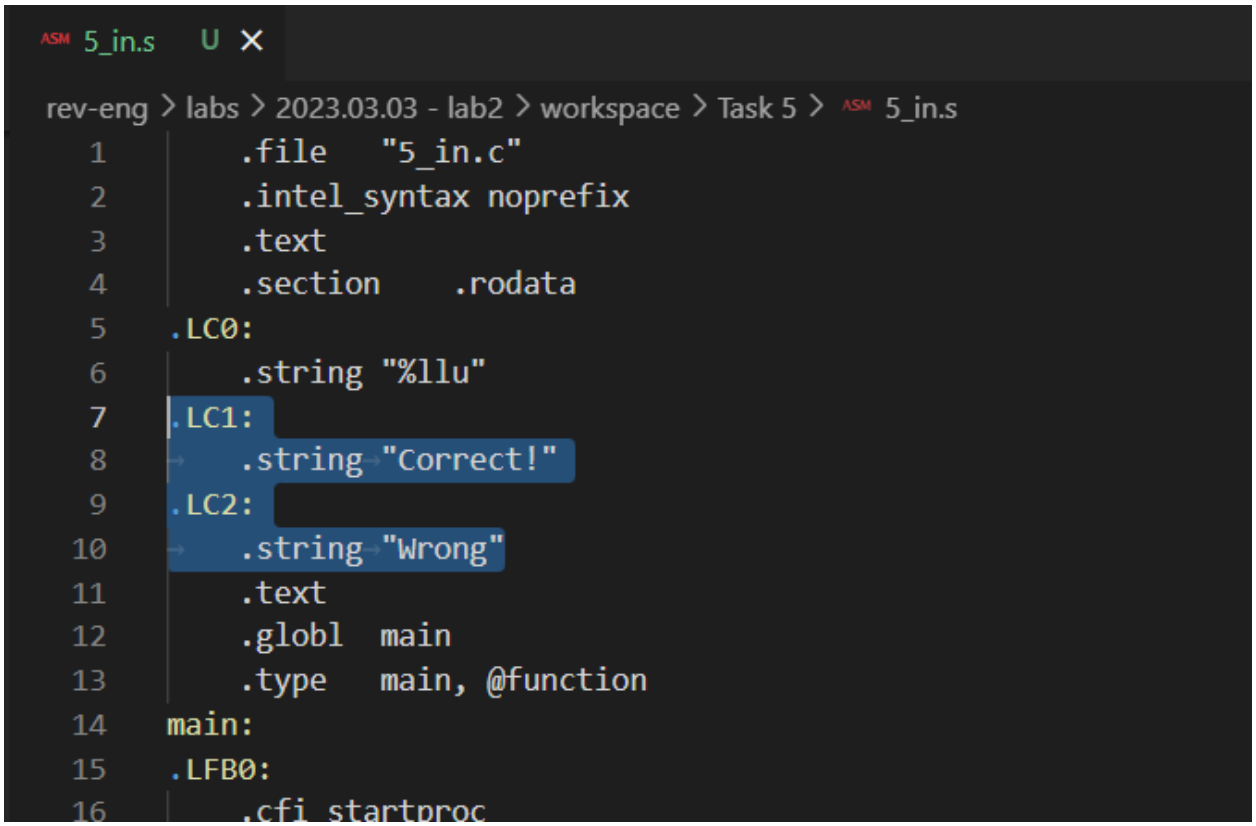
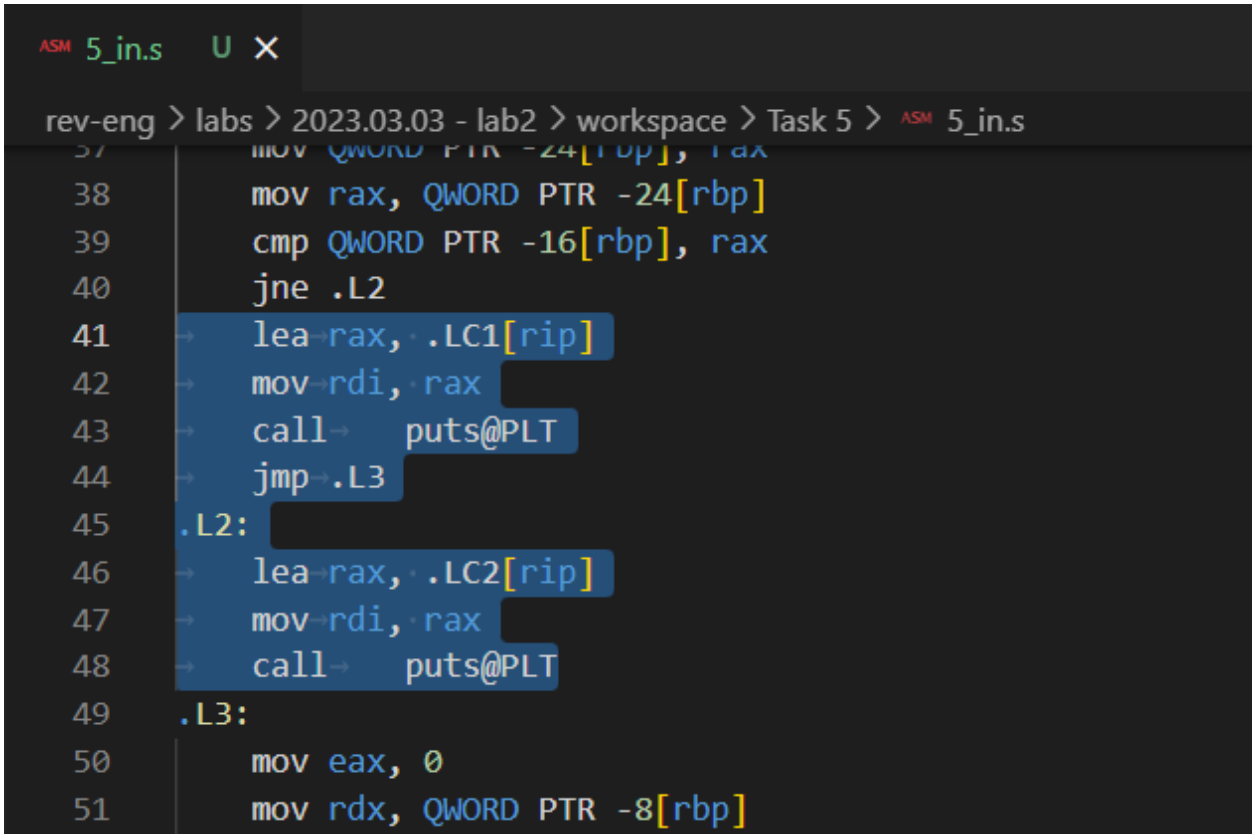
In order to figure out what we need to change in the binary file, it's useful to first compile the code into the corresponding assembly instructions.



From the assembly file, the relevant parts are the String declarations:



And the calls to **printf** (which are actually optimized to call to **puts**):



The idea is that we want to change the string pointers used in the binary as arguments to puts() so that the pointer that was originally targeted at string "Correct!" then points to string "Wrong" and vice versa. The instructions that load these pointer values into RAX are the following:

```
lea     rax, .LC1[rip]
lea     rax, .LC2[rip]
```

So we want to look in the binary file for instructions of the type

```
lea     rax, VALUE[rip]
```

But how do they look as binary opcodes?

By disassembling this example instruction:

```
lea     rax, 0x0F0F0F0F[rip]
```

at

https://defuse.ca/online-x86-assembler.htm#disassembly

we can see that this instruction begins with:

0x48, 0x8D, 0x05

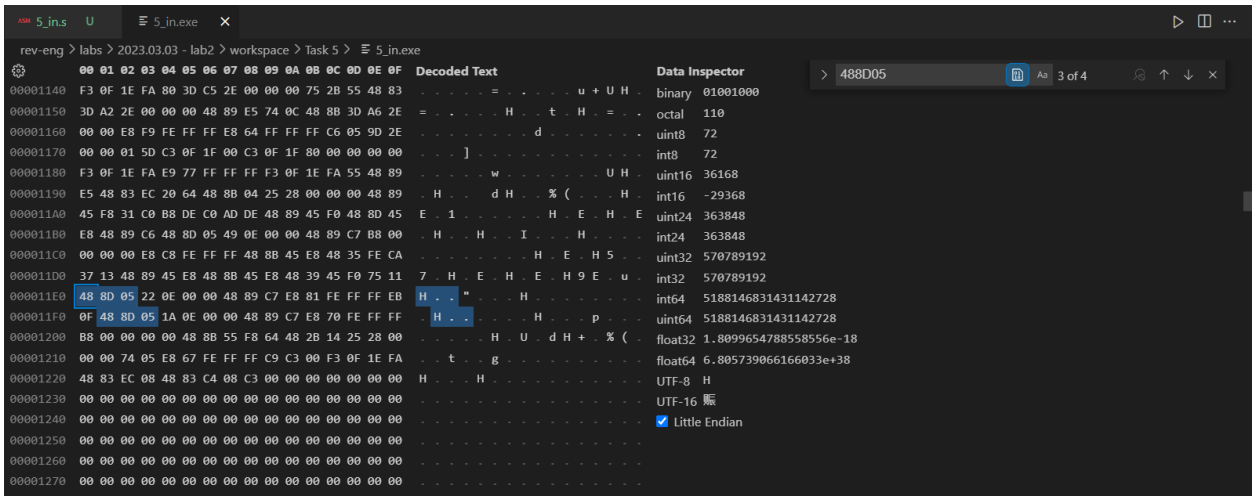
so we look for these bytes in the binary file.

We actually find 4 results if we search for that sequence of bytes in the file, but that is because there are 4 instructions of the type

```
lea     rax, VALUE[rip]
```

In the binary file. The LEA instructions that we are interested in are the 3rd and the 4th LEA instructions in the assembly file we generated at first, so that means we must look for the 3rd and 4th occurrence of the { 0x48, 0x8D, 0x05 } byte sequence.

We find:
- third LEA (that loads string "Correct!"):
    48 8D 05 22 0E 00 00
- fourth LEA (that loads string "Wrong"):
    48 8D 05 1A 0E 00 00

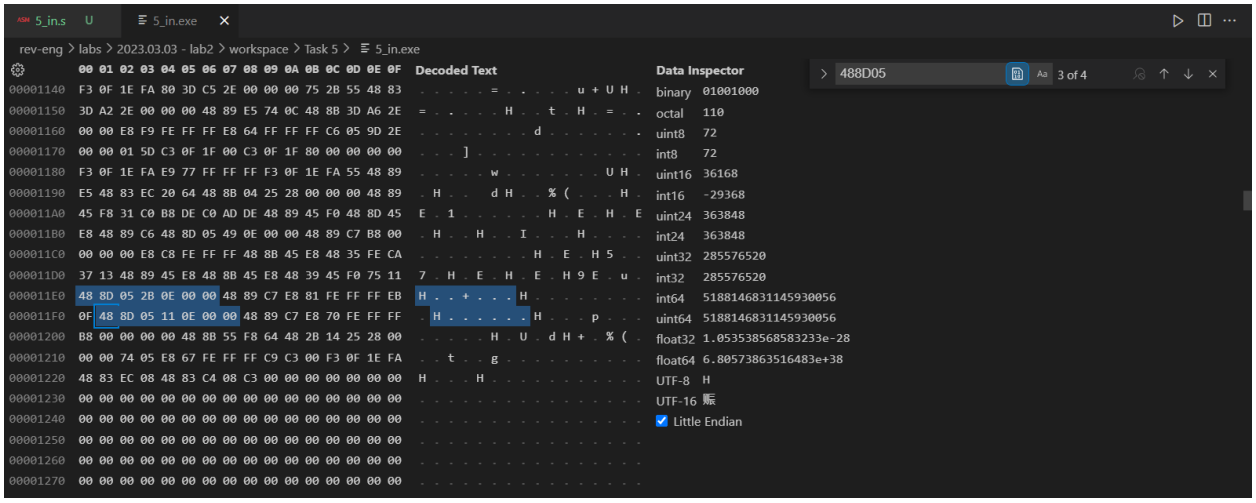


The fourth bytes in each instruction (22 and 1A) are part of the offset value used to compute the addresses of the two strings. We want to swap these two values, but since the LEA constants are relative offsets we also need to take into account the offset between the position of these two instructions in the file. The third LEA instruction is placed **17 bytes** before the fourth LEA instruction. So we need to swap the bytes 22(hex) and 1A(hex) and take into account an offset of 17(decimal).

1A(hex) + 17(decimal) = 2B(hex)
So we replace '22' with '2B' in the third LEA instruction.

22(hex) - 17(decimal) = 11(hex)
So we replace '1A' with '11' in the fourth LEA instruction.

So we get:
- modified third LEA (that now loads "Wrong"):
    48 8D 05 2B 0E 00 00
- modified fourth LEA (that now loads "Correct!"):
    48 8D 05 11 0E 00 00



After this change, we can see that the binary behavior has been reverted:

```
bogdanisar@DESKTOP-C2G1FF4:/mnt/d/Files/Programming/reverse-engineering-course/rev-eng/labs/2023.03.03 - lab2/workspace/Task 5$ ./5_in.exe
123
Correct!
bogdanisar@DESKTOP-C2G1FF4:/mnt/d/Files/Programming/reverse-engineering-course/rev-eng/labs/2023.03.03 - lab2/workspace/Task 5$ ./5_in.exe
0
Correct!
bogdanisar@DESKTOP-C2G1FF4:/mnt/d/Files/Programming/reverse-engineering-course/rev-eng/labs/2023.03.03 - lab2/workspace/Task 5$ ./5_in.exe
537485968087967546463542
Correct!
bogdanisar@DESKTOP-C2G1FF4:/mnt/d/Files/Programming/reverse-engineering-course/rev-eng/labs/2023.03.03 - lab2/workspace/Task 5$ ./5_in.exe
3449424416
Wrong
```