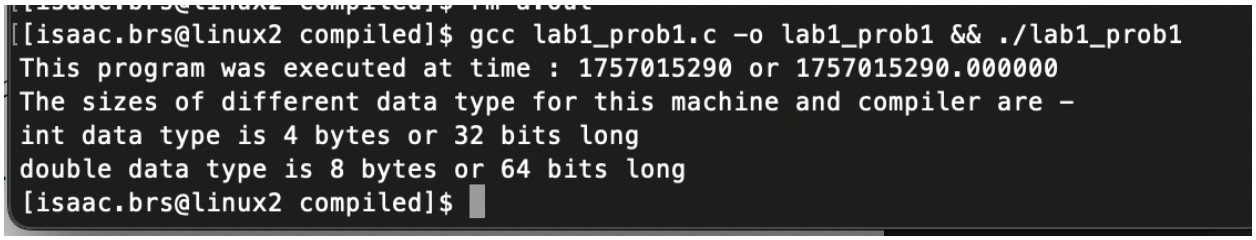
Prob1

- a)
  - Tag1: checks if the return of fopen, which is a FILE pointer, is null.
  - Tag2: gets the time of day assuming timezone “0”, and places it in this_instant.
  - Tag3: prints the size of int in bits and bytes to the file using the file pointer. It is worth noting that it is printing them as “%lu” -> long unsigned.
  - Tag4: prints the size of int in bits and bytes to the console. It is worth noting that it is printing them as “%ld” -> long signed.
- b)



- c) timeval is of struct {
  - time_t tv_sec; //time in seconds
  - suseconds_t tv_usec; //time in microseconds
  - }
  - tv_sec is the time in seconds.

Prob2

- a)

```
int data type is 4 bytes or 32 bits long
double data type is 8 bytes or 64 bits long
[[isaac.brs@linux2 compiled]$ gcc lab1_prob2.c -o lab1_prob2 && ./lab1_prob2
Size of employee1 structure: 56 bytes
Size of employee2 structure: 56 bytes
[isaac.brs@linux2 compiled]$
```

```c
#include <stdio.h>
#include <stdlib.h>



int main(int argc, char *argv[])
{
    struct employee1 {
        int id;
        char name[50];
    };

    struct employee2 {
        int id;
        char name[52];
    };

    printf("Size of employee1 structure: %lu bytes\n", sizeof(struct employee1));
    printf("Size of employee2 structure: %lu bytes\n", sizeof(struct employee2));

    return 0;
}
```

- b) The reason both size structs are 56 bytes is because the compiler is padding the structs to the closest 4 bytes. Ints are 4 bytes, and 50 or 52 both rounded to the nearest 4 are 52 bytes. 52 + 4 = 56 bytes.
    - Following this, if you set char name[48];, sizeof will return 52 bytes instead.

Prob3 a)

DSBF - Driver Seatbelt Fastened	ER - Engine Running	DC - Doors Closed	DLC - Door Lock Lever	DOS - Driver on Seat	KIC - Key in Car	BP - Brake Pedal	CM - Car Moving	BELL - Bell Actuator
0	1	X	X	X	X	X	X	1
X	1	0	X	X	X	X	X	1
1	1	1	X	X	X	X	X	0
X	0	X	X	X	X	X	X	0

BELL = ER & (!DSBF | !DC)

DSBF - Driver Seatbelt Fastened	ER - Engine Running	DC - Doors Closed	DLC - Door Lock Lever	DOS - Driver on Seat	KIC - Key in Car	BP - Brake Pedal	CM - Car Moving	DLA - Door Lock Actuator
X	X	X	1	1	X	X	X	1
X	X	X	X	0	1	X	X	0
X	X	X	X	X	X	X	X	0

DLA = DOS & DLC

DSBF - Driver Seatbelt Fastened	ER - Engine Running	DC - Doors Closed	DLC - Door Lock Lever	DOS - Driver on Seat	KIC - Key in Car	BP - Brake Pedal	CM - Car Moving	BA - Brake Actuator
X	X	X	X	X	X	1	1	1
X	X	X	X	X	X	0	X	0

BA = BP & CM

- b)
  - BELL = ER & (!DSBF | !DC)
  - DLA = DOS & DLC
  - BA = BP & CM
- c)

```c
void read_inputs_from_ip_if(){

    // 1. Provide your input code here
    // This function should read the current state of the available sensors (8 in total)
    //if scanf != 1 -> fails silently
    scanf("%u", &driver_seat_belt_fastened);
    scanf("%u", &engine_running);
    scanf("%u", &doors_closed);
    scanf("%u", &door_lock_lever);
    scanf("%u", &driver_on_seat);
    scanf("%u", &key_in_car);
    scanf("%u", &brake_pedal);
    scanf("%u", &car_moving);
    // Hint : You can use scanf to obtain inputs for the sensors

}

void write_output_to_op_if(){
    // 2. Provide your output code here
    // This function should display/print the state of the 3 actuators (BELL/DLA/BA)

    printf("BELL: %u\n", bell);
    printf("DOOR LOCK ACTUATOR: %u\n", door_lock_actu);
    printf("BRAKE PEDAL ACTUATOR: %u\n", brake_actu);
}
```

- d)

```
BRAKE PEDAL ACTUATOR: 0
kingisaac@Johns-MacBook-Pro-5 lab1_src % gcc -std=c99 lab1_prob3_framework.c -o lab1_prob3 && ./lab1_prob3

Test 0:  0 0 0 0 0 0 0 0
BELL: 0
DOOR LOCK ACTUATOR: 0
BRAKE PEDAL ACTUATOR: 0

Test 1:  1 1 0 0 0 1 0 1
BELL: 1
DOOR LOCK ACTUATOR: 0
BRAKE PEDAL ACTUATOR: 0

Test 2:  1 0 1 0 1 1 1 1
BELL: 0
DOOR LOCK ACTUATOR: 0
BRAKE PEDAL ACTUATOR: 1

Test 3:  0 1 0 1 0 1 0 0
BELL: 1
DOOR LOCK ACTUATOR: 0
BRAKE PEDAL ACTUATOR: 0

Test 4:  0 1 1 1 1 1 1 0
BELL: 1
DOOR LOCK ACTUATOR: 1
BRAKE PEDAL ACTUATOR: 0

Test 5:  1 1 0 1 0 1 0 1
BELL: 1
DOOR LOCK ACTUATOR: 0
BRAKE PEDAL ACTUATOR: 0

Test 6:  1 1 1 1 1 1 1 1
BELL: 0
DOOR LOCK ACTUATOR: 1
BRAKE PEDAL ACTUATOR: 1

Test 7:  0 1 0 0 0 1 1 0
BELL: 1
DOOR LOCK ACTUATOR: 0
BRAKE PEDAL ACTUATOR: 0
```

Prob4

a)

```
20
21  // Define bit positions
22  #define DRIVER_SEAT_BELT 0  // bit 0
23  #define ENGINE_RUNNING   1  // bit 1
24  #define DOORS_CLOSED     2  // bit 2
25  #define DOOR_LOCK_LEVER  3  // bit 3
26  #define DRIVER_ON_SEAT   4  // bit 4
27  #define KEY_IN_CAR       5  // bit 5
28  #define BRAKE_PEDAL      6  // bit 6
29  #define CAR_MOVING       7  // bit 7
30
31
32  #include <stdio.h> //This is useful to do i/o to test the code
33
34  unsigned int input = 0;
35  unsigned int output = 0;
36
37  void read_inputs_from_ip_if(){
38      //This read the current state of the available sensors
39
40      printf("input signal: ");
41      scanf("%d", &input);
42  }
43
44  void write_output_to_op_if(){
45
```

b)

```
60
61      //   7  6  5   4   3   2  1  0
62      //  CM BP KIC DOS DLC DC ER DSBF
63
64      //   2   1   0
65      //   BA DLA BELL
66
67      //    if (engine_running && !doors_closed) bell = 1;
68
69      //    extract the relevant bits using the mask [00000110] and compare with condition [00000010]
70      //    if we need to change the last bit to 1, we use bitwise OR
71      //    if we need to change the last bit to 0, we use bitwise AND
72      //    if (input & [00000110] == [00000010]) output = output | [001]
73      if((input & (1 << ENGINE_RUNNING))
74      && (!(input & (1 << DRIVER_SEAT_BELT)) || !(input & (1 << DOORS_CLOSED)))){
75          output = output | 1;
76      }
77      if((input & ((1 << DRIVER_ON_SEAT) + (1 << DOOR_LOCK_LEVER)))
78      == (1 << DRIVER_ON_SEAT) + (1 << DOOR_LOCK_LEVER)){
79          output = output | 2;
80      }
81      if((input & ((1 << BRAKE_PEDAL) + (1 << CAR_MOVING)))
82      == (1 << BRAKE_PEDAL) + (1 << CAR_MOVING)){
83          output = output | 4;
84      }
85  }
```

c)

```
kingisaac@Johns-MacBook-Pro-5 lab1_src % gcc -std=c99 lab1_prob4_framework.c -o lab1_prob4 && ./lab1_prob4
Case 0:  0 0 0
Case 1:  1 0 0
Case 2:  0 0 1
Case 3:  1 0 0
Case 4:  1 1 0
Case 5:  1 0 0
Case 6:  0 1 1
Case 7:  1 0 0
```

Prob5

NOTE: When

Using prob3's code:

```
[[isaac.brs@linux2 lab1_src]$ gcc lab1_prob5_framework.c -lrt
[[isaac.brs@linux2 lab1_src]$ ./a.out
[0 1 0 1 0 1 1 0
BELL: 1
DOOR LOCK ACTUATOR: 0
BRAKE PEDAL ACTUATOR: 0
Timer Resolution = 1 nanoseconds
 Calibrartion time = 0 seconds and 1283 nanoseconds
 The measured code took 0 seconds and  1241 nano seconds to run
```

Using prob4's code:

```
[isaac.brs@linux2 lab1_src]$ gcc lab1_prob5_framework.c -lrt
[isaac.brs@linux2 lab1_src]$ ./a.out
input signal: 01010110
output signal: 3
Timer Resolution = 1 nanoseconds
 Calibrartion time = 0 seconds and 1343 nanoseconds
 The measured code took 0 seconds and  1528 nano seconds to run
```

Turns out, problem 3's code is faster than problem 4's. Strange.