1a.

Tag 1: It creates the lab1_prob1_oute.txt and sets it to write. If the file cannot be opened for some reason it creates an error message.

Tag 2: It Stores the number of seconds into the structure of “this_instant” since the Epoch of January 1st 1970.

Tag 3: It writes the size of an int data type in both bytes and bits into lab1_prob1_out.txt.

Tag 4: Prints the size of an int data type in both bytes and bits to the console output.

1b.

```
[shouroshuvit@linux2 lab1_src]$ ./lab1_prob1.out
This program was executed at time : 1724860527 or 1724860527.000000
The sizes of different data type for this machine and compiler are -
int data type is 4 bytes or 32 bits long
double data type is 8 bytes or 64 bits long
```

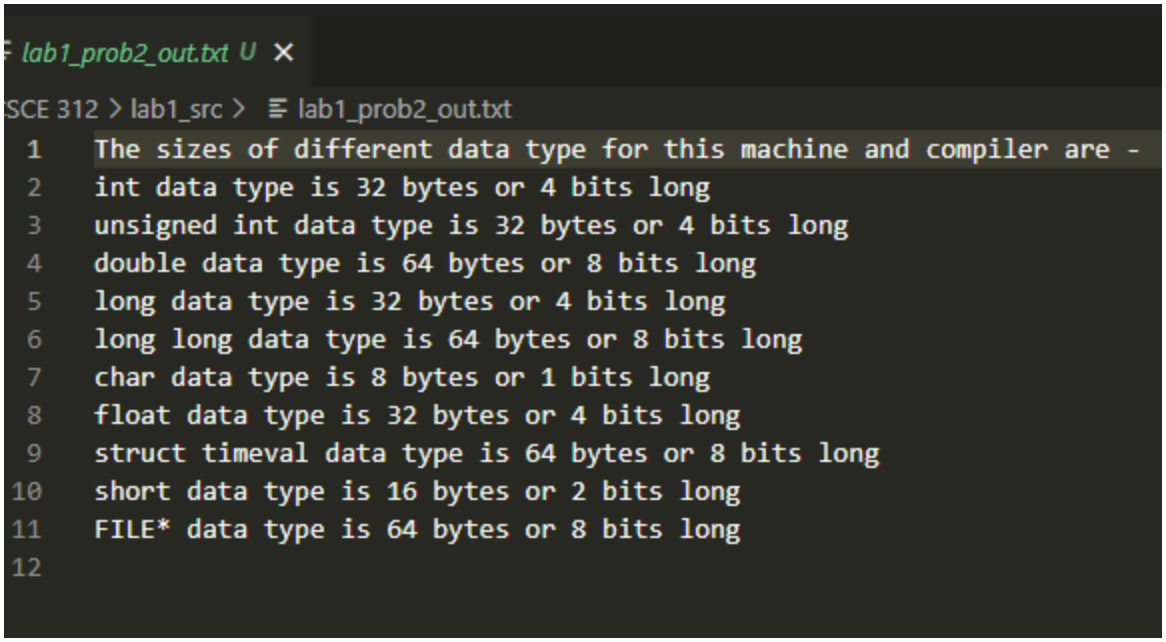
1c.

```
struct timeval {
        long tv_sec;     /* seconds */
        long tv_usec;    /* microseconds */
};
```

The structure of timeval is above where tv_sec is a long integer representing the number of seconds since the Epoch of January 1, 1970 and long tv_usec is that same time in milliseconds.
tv_sec is long because it has enough bits to accommodate the billions of seconds since the start of 1970.

2a.

```
[Running] cd "c:\Users\ssjed\OneDrive\Documents\GitHub\Fall-2024\CSCE 312\lab1_src\" && gcc lab1_prob2.c -o lab1_prob2 && "c:\Users\ssjed\OneDrive\Documents\GitHub\Fall-2024\CSCE 312\lab1_src\"lab1_prob2
The sizes of different data type for this machine and compiler are -
int data type is 4 bytes or 32 bits long
unsigned int data type is 4 bytes or 32 bits long
double data type is 8 bytes or 64 bits long
long data type is 4 bytes or 32 bits long
long long data type is 8 bytes or 64 bits long
char data type is 1 bytes or 8 bits long
float data type is 4 bytes or 32 bits long
struct timeval data type is 8 bytes or 64 bits long
short data type is 2 bytes or 16 bits long
FILE* data type is 8 bytes or 64 bits long
Size of struct employee1: 56 bytes
Size of struct employee2: 56 bytes
Output file generated successfully.
```



2b.

```
short data type is 16 bytes or 2 bits long
FILE* data type is 64 bytes or 8 bits long
```

3a.

Bell Truth Table

DSBF (Driver Seat Belt Fastened)	ER (Engine Running)	DC (Doors Closed)	BELL
0	1	0	1
0	1	1	1
0	0	0	0
0	0	1	0
1	1	0	1
1	1	1	0
1	0	0	0
1	0	1	0

Door Lock Actuator Truth Table

DOS (Driver On Seat)	KIC (Key In Car)	DLC (Door Lock Lever)	DC (Doors Closed)	DLA
1	0	1	1	1
1	0	1	0	0
1	1	1	1	1
1	1	1	0	0
0	0	1	1	0
0	0	0	0	0
0	1	1	1	0
0	1	0	0	0

Brake Actuator Truth Table

BP (Brake Pedal)	CM (Car Moving)	BA(Brake Actuator)
1	1	1
1	0	0
0	1	0
0	0	0

3b.

$$BELL = \overline{DSBF} \cdot ER + \overline{DC} \cdot ER$$

$$DLA = DOS \cdot DLC \cdot DC + \overline{KIC} \cdot \overline{DOS} \cdot \overline{DLC}$$

$$BA = BP \cdot CM$$

3c.

```
// BELL logic
if (engine_running && !driver_seat_belt_fastened) {
    bell = 1;
}
else if (engine_running && !doors_closed) {
    bell = 1;
}
else {
    bell = 0;
}

// DLA logic
if (driver_on_seat && door_lock_lever && doors_closed) {
    door_lock_actu = 1;
}
else if (!key_in_car && !driver_on_seat && door_lock_lever) {
    door_lock_actu = 0;
}
else {
    door_lock_actu = 0;
}

// BA logic
if (brake_pedal && car_moving) {
    brake_actu = 1;
}
else {
    brake_actu = 0;
}
```

3d.

```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)

    // Hint : You can use scanf to obtain inputs for the sensors
    scanf("%d %d %d %d %d %d %d %d", &driver_seat_belt_fastened, &engine_running, &doors_closed,
                                     &door_lock_lever, &driver_on_seat, &key_in_car,
                                     &brake_pedal, &car_moving);
}

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: %d, Door Lock Actuator: %d, Brake Actuator: %d\n", bell, door_lock_actu, brake_actu);
}
```

```c
void control_action(){

    /*
       The code given here sounds the bell when driver starts the engine and
       hasn't closed the doors. (Requirement-2)

       3. Provide your own code to do problems 3, which satisfies 5 requirements
    */

    // BELL logic
    if (engine_running && !driver_seat_belt_fastened) {
        bell = 1;
    }
    else if (engine_running && !doors_closed) {
        bell = 1;
    }
    else {
        bell = 0;
    }

    // DLA logic
    if (driver_on_seat && door_lock_lever && doors_closed) {
        door_lock_actu = 1;
    }
    else if (!key_in_car && !driver_on_seat && door_lock_lever) {
        door_lock_actu = 0;
    }
    else {
        door_lock_actu = 0;
    }

    // BA logic
    if (brake_pedal && car_moving) {
        brake_actu = 1;
    }
    else {
        brake_actu = 0;
    }


}
```

```
Test 0:  0 0 0 0 0 0 0 0 BELL: 0, Door Lock Actuator: 0, Brake Actuator: 0

Test 1:  1 1 0 0 1 0 1 0 BELL: 1, Door Lock Actuator: 0, Brake Actuator: 0

Test 2:  1 0 0 1 1 1 1 0 BELL: 0, Door Lock Actuator: 0, Brake Actuator: 1

Test 3:  0 1 1 0 1 0 0 0 BELL: 1, Door Lock Actuator: 0, Brake Actuator: 0

Test 4:  0 1 1 1 1 1 0 0 BELL: 1, Door Lock Actuator: 1, Brake Actuator: 0

Test 5:  1 1 1 0 1 0 1 0 BELL: 1, Door Lock Actuator: 0, Brake Actuator: 0

Test 6:  1 1 1 1 1 1 1 0 BELL: 0, Door Lock Actuator: 1, Brake Actuator: 1

Test 7:  0 1 0 0 1 1 0 0 BELL: 1, Door Lock Actuator: 0, Brake Actuator: 0
```

4a.

```
inline unsigned int aANDb_function(unsigned int passed_input)
{
    unsigned int returned_output;

    //To implement  a AND b
    if ( (passed_input & 0x3) == 3) //To mask the bit position 1 and 0
        returned_output = 1; // output is 1, when both a =1, b =1
    else returned_output = 0; //output is 0, when for combination a= 0, b=0; a=0, b=1; a=1, b=0

    return returned_output;
}
      You, last week • added CSCE lab 1 files

//To implement a OR b
inline unsigned int aORb_function(unsigned int passed_input)
{
    unsigned int returned_output;

    //To implement a OR b
    returned_output = ( (passed_input & 0x3) == 0) ?  0: 1 ; //This one implements the OR's truth table

    return returned_output;
}


//To implement c OR (a AND b)
inline unsigned int cOR_aNADb_function(unsigned int passed_input)
{
    unsigned int returned_output;

    //To implement c OR (a AND b)
    return returned_output = ( (passed_input & 0x7) > 2) ?  1: 0 ; //This one implements the (c OR (a AND b)) 's truth table

}
```

```
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
```

5.

Problem 3 Execution TIme

```
bell = 0
door_lock_actu = 0
brake_actu = 1
Timer Resolution = 1 nanoseconds
 Calibrartion time = 0 seconds and 1092 nanoseconds
 The measured code took 0 seconds and  4294966605 nano seconds to run
```

Problem 4 Execution Time

```
bell = 0
door_lock_actu = 0
brake_actu = 1
Timer Resolution = 1 nanoseconds
 Calibrartion time = 0 seconds and 981 nanoseconds
 The measured code took 0 seconds and  4294966823 nano seconds to run
```

The else-if block runs marginally faster by around 200 nanoseconds.