CSCE 312 – Lab 1 Report

Texas A&M University

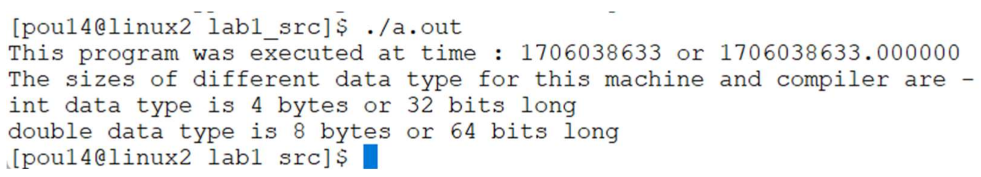
January 25, 2024

Poyi Ou

Problem 1:

a) **Tag 1:** This line of code is to check whether the text file successfully opened or not.
Tag 2: This line of code is to get current time of the day and save to variable this_instant.
Tag 3: This line of code saves the output to the lab1_prob1_out.txt file, and it prints what the data type is and how many bits and how many bits long. And in this case, it using the data type int.
Tag 4: This line of code prints the output to console, and it prints what the data type is and how many bits and how many bits long. And in this case, it's using the data type int.

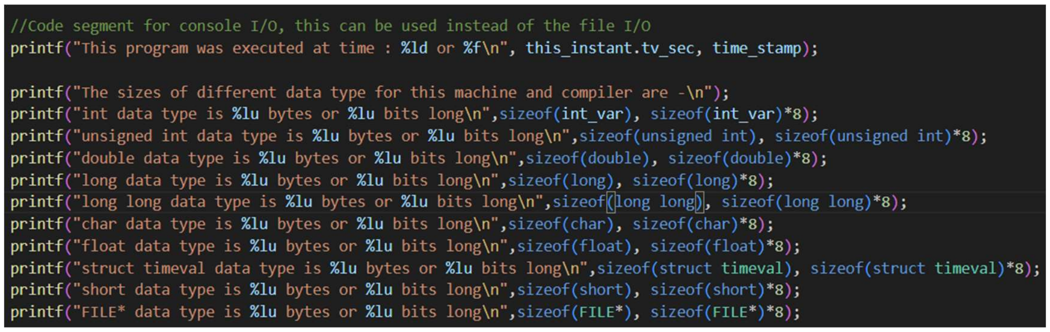
b)



```
[pou14@linux2 lab1_src]$ ./a.out
This program was executed at time : 1706038633 or 1706038633.000000
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
[pou14@linux2 lab1 src]$
```

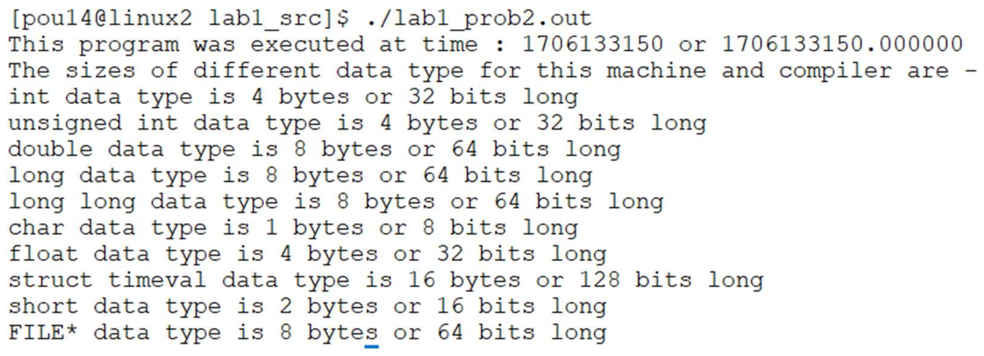
c) The structure type of timeval got a variable name called tv_sec with its data type of time_t. Variable named tv_sec got an data type of suseconds_t.

Problem 2:

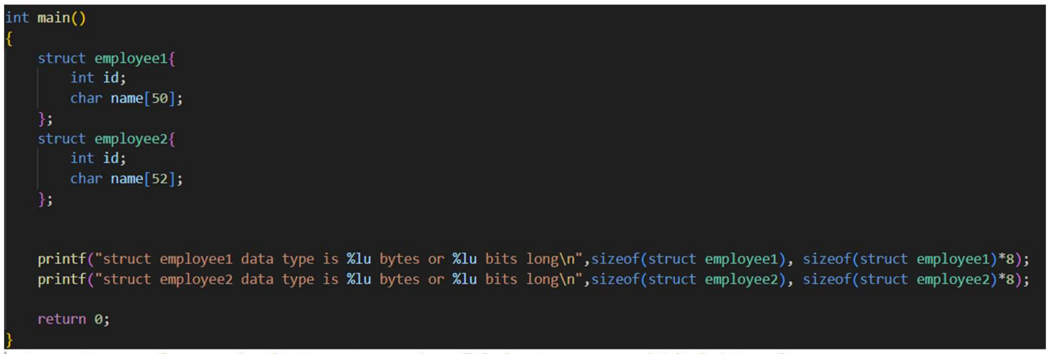


```
//Code segment for console I/O, this can be used instead of the file I/O
printf("This program was executed at time : %ld or %f\n", this_instant.tv_sec, time_stamp);

printf("The sizes of different data type for this machine and compiler are -\n");
printf("int data type is %lu bytes or %lu bits long\n",sizeof(int_var), sizeof(int_var)*8);
printf("unsigned int data type is %lu bytes or %lu bits long\n",sizeof(unsigned int), sizeof(unsigned int)*8);
printf("double data type is %lu bytes or %lu bits long\n",sizeof(double), sizeof(double)*8);
printf("long data type is %lu bytes or %lu bits long\n",sizeof(long), sizeof(long)*8);
printf("long long data type is %lu bytes or %lu bits long\n",sizeof(long long), sizeof(long long)*8);
printf("char data type is %lu bytes or %lu bits long\n",sizeof(char), sizeof(char)*8);
printf("float data type is %lu bytes or %lu bits long\n",sizeof(float), sizeof(float)*8);
printf("struct timeval data type is %lu bytes or %lu bits long\n",sizeof(struct timeval), sizeof(struct timeval)*8);
printf("short data type is %lu bytes or %lu bits long\n",sizeof(short), sizeof(short)*8);
printf("FILE* data type is %lu bytes or %lu bits long\n",sizeof(FILE*), sizeof(FILE*)*8);
```

a)



```
[pou14@linux2 lab1_src]$ ./lab1_prob2.out
This program was executed at time : 1706133150 or 1706133150.000000
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 8 bytes or 64 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 16 bytes or 128 bits long
short data type is 2 bytes or 16 bits long
FILE* data type is 8 bytes or 64 bits long
```



```c
int main()
{
    struct employee1{
        int id;
        char name[50];
    };
    struct employee2{
        int id;
        char name[52];
    };


    printf("struct employee1 data type is %lu bytes or %lu bits long\n",sizeof(struct employee1), sizeof(struct employee1)*8);
    printf("struct employee2 data type is %lu bytes or %lu bits long\n",sizeof(struct employee2), sizeof(struct employee2)*8);

    return 0;
}
```

b)

```
struct employee1 data type is 56 bytes or 448 bits long
struct employee2 data type is 56 bytes or 448 bits long
```

The result shows that the data type of employee1 and employee2 were both 56 bytes or 448 bits long because of the memory alignment that came in place. Memory alignment depends on the complier and architecture. Where in my case, I had 64-bit type computer so it will align 8-byte boundaries each time, and since 50 and 52 were in the same chunk of 48~56 bytes, it will ended up that they used the same amount of memory.

Problem 3:

a)

Truth table for BELL

DSBF	ER	DC	DLC	DOS	KIC	BP	CM	BELL
1	1	1	X	X	X	X	X	0
1	1	0	X	X	X	X	X	1
1	0	1	X	X	X	X	X	0
1	0	0	X	X	X	X	X	0
0	1	1	X	X	X	X	X	1
0	1	0	X	X	X	X	X	1
0	0	1	X	X	X	X	X	0
0	0	0	X	X	X	X	X	0

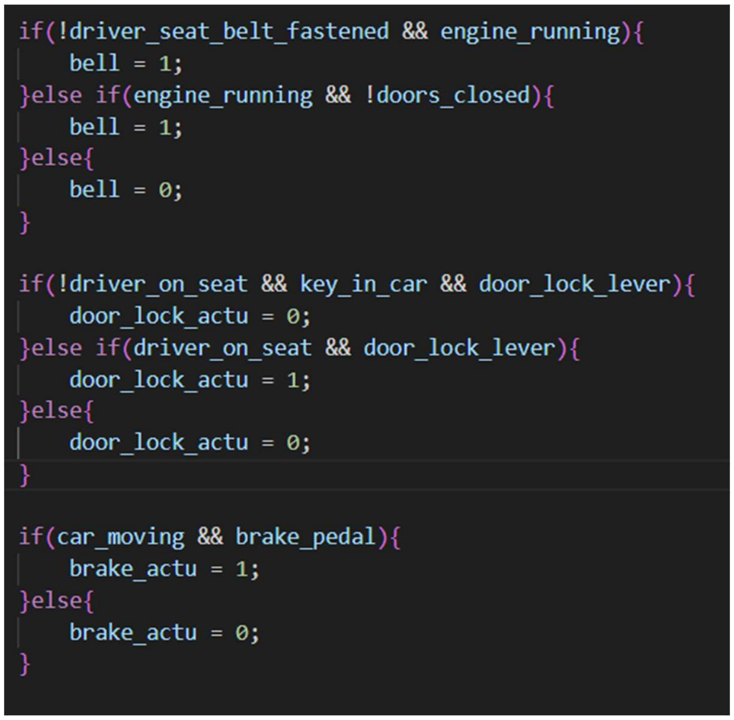
Truth table for DLA

DSBF	ER	DC	DLC	DOS	KIC	BP	CM	DLA
X	X	X	1	1	1	X	X	1
X	X	X	1	1	0	X	X	1
X	X	X	1	0	1	X	X	0
X	X	X	1	0	0	X	X	1
X	X	X	0	1	1	X	X	0
X	X	X	0	1	0	X	X	0
X	X	X	0	0	1	X	X	0
X	X	X	0	0	0	X	X	0

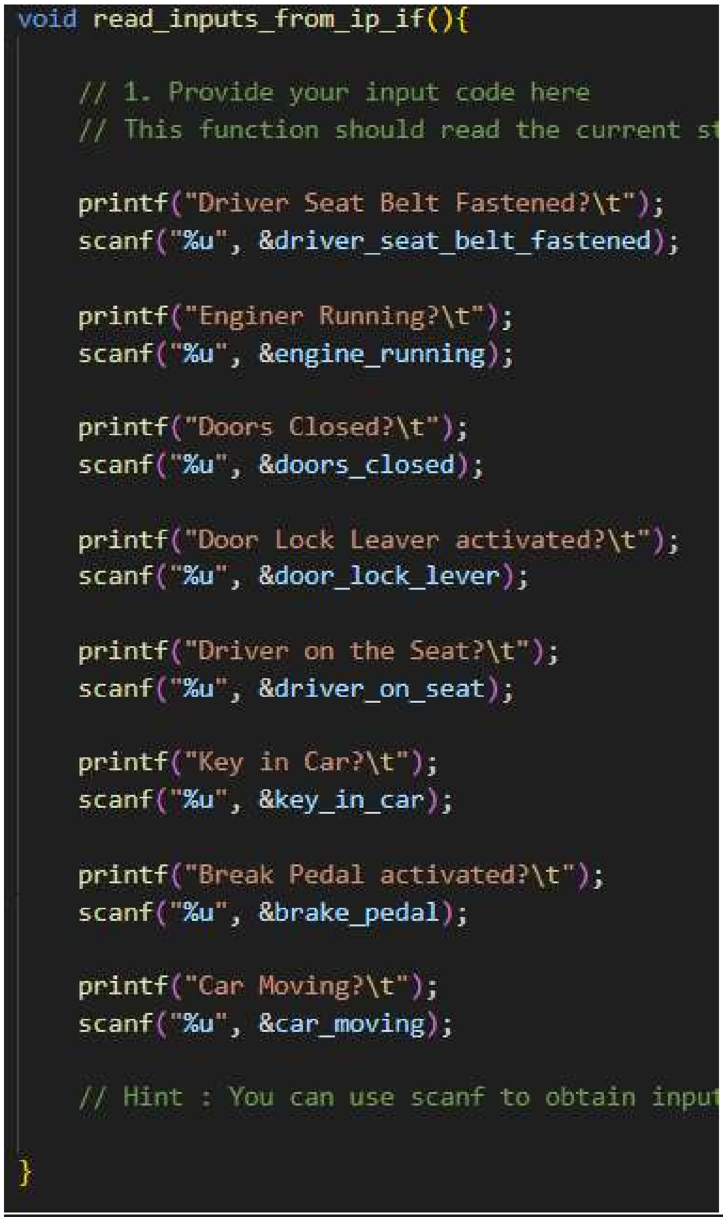
Truth table for BA

DSBF	ER	DC	DLC	DOS	KIC	BP	CM	BA
X	X	X	X	X	X	1	1	1
X	X	X	X	X	X	1	0	0
X	X	X	X	X	X	0	1	0
X	X	X	X	X	X	0	0	0

b) BELL = ER * (DSBF' + DC')
DLA = DLC * (DOS + KIC')
BA = BP * CM

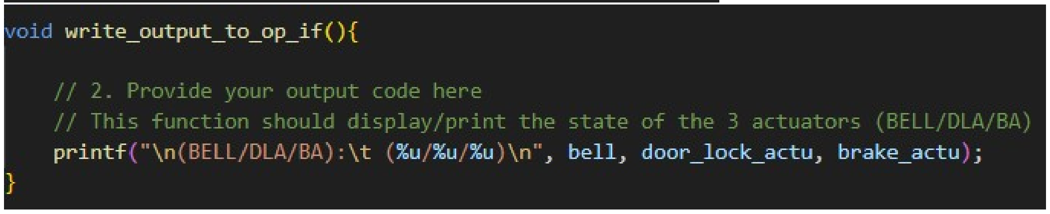
c) 

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

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

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

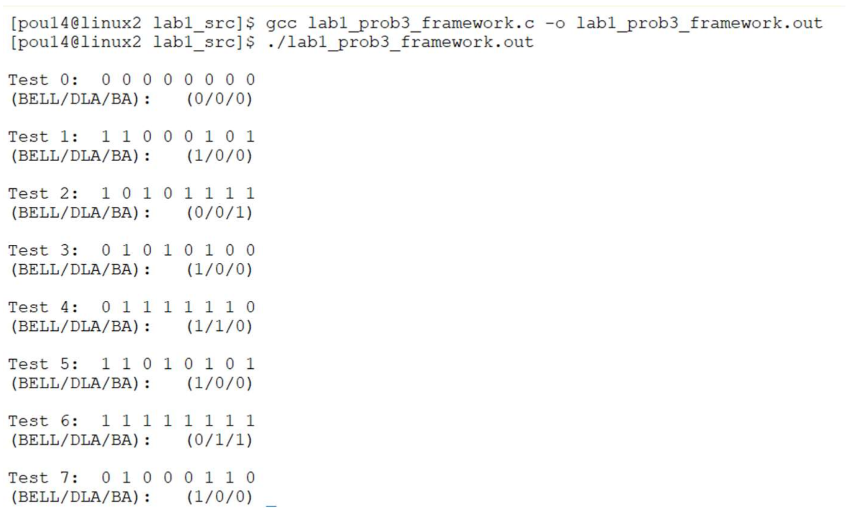


```
void read_inputs_from_ip_if(){

    // 1. Provide your input code here
    // This function should read the current st

    printf("Driver Seat Belt Fastened?\t");
    scanf("%u", &driver_seat_belt_fastened);

    printf("Enginer Running?\t");
    scanf("%u", &engine_running);

    printf("Doors Closed?\t");
    scanf("%u", &doors_closed);

    printf("Door Lock Leaver activated?\t");
    scanf("%u", &door_lock_lever);

    printf("Driver on the Seat?\t");
    scanf("%u", &driver_on_seat);

    printf("Key in Car?\t");
    scanf("%u", &key_in_car);

    printf("Break Pedal activated?\t");
    scanf("%u", &brake_pedal);

    printf("Car Moving?\t");
    scanf("%u", &car_moving);

    // Hint : You can use scanf to obtain input

}
```

d)

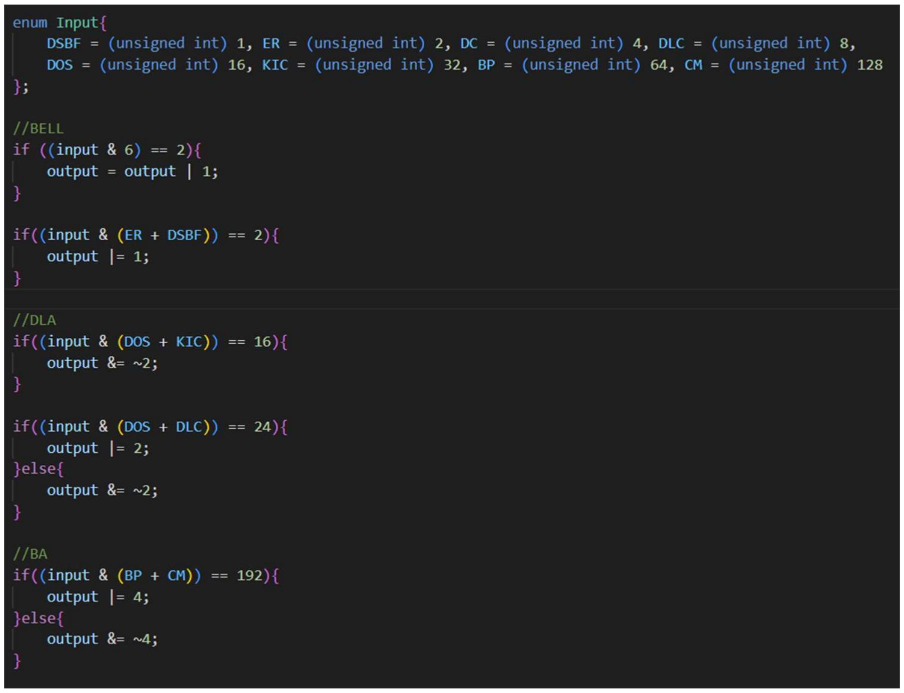


```
void write_output_to_op_if(){

    // 2. Provide your output code here
    // This function should display/print the state of the 3 actuators (BELL/DLA/BA)
    printf("\n(BELL/DLA/BA):\t (%u/%u/%u)\n", bell, door_lock_actu, brake_actu);
}
```

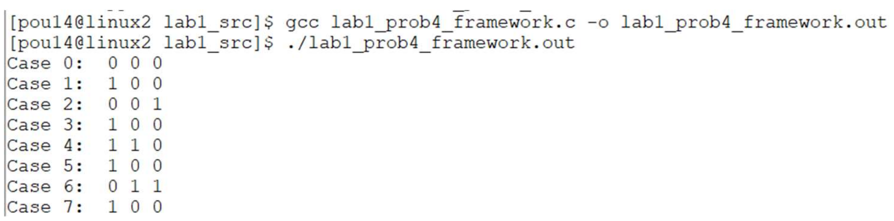


```
[pou14@linux2 lab1_src]$ gcc lab1_prob3_framework.c -o lab1_prob3_framework.out
[pou14@linux2 lab1_src]$ ./lab1_prob3_framework.out

Test 0:  0 0 0 0 0 0 0 0
(BELL/DLA/BA):   (0/0/0)

Test 1:  1 1 0 0 0 1 0 1
(BELL/DLA/BA):   (1/0/0)

Test 2:  1 0 1 0 1 1 1 1
(BELL/DLA/BA):   (0/0/1)

Test 3:  0 1 0 1 0 1 0 0
(BELL/DLA/BA):   (1/0/0)

Test 4:  0 1 1 1 1 1 1 0
(BELL/DLA/BA):   (1/1/0)

Test 5:  1 1 0 1 0 1 0 1
(BELL/DLA/BA):   (1/0/0)

Test 6:  1 1 1 1 1 1 1 1
(BELL/DLA/BA):   (0/1/1)

Test 7:  0 1 0 0 0 1 1 0
(BELL/DLA/BA):   (1/0/0) _
```

Problem 4:

a) 

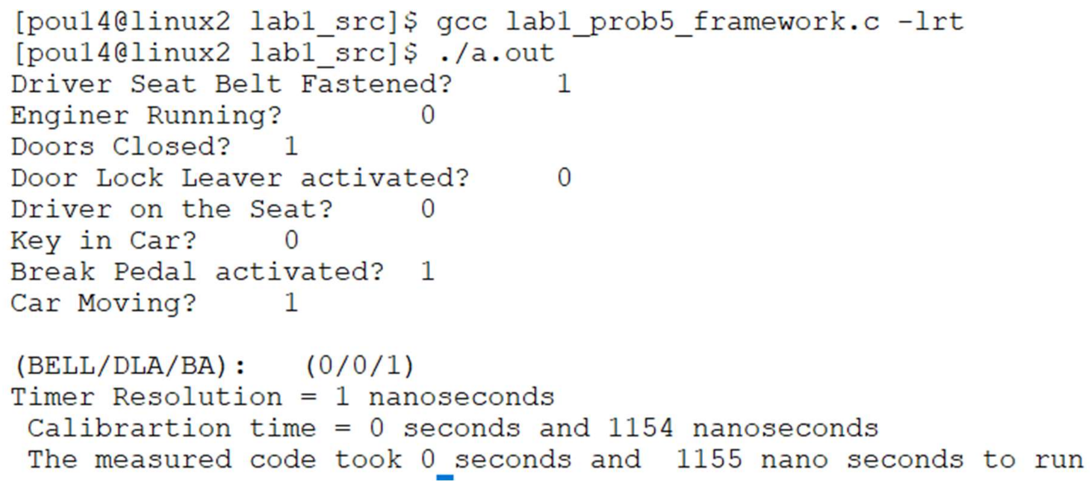
```
enum Input{
    DSBF = (unsigned int) 1, ER = (unsigned int) 2, DC = (unsigned int) 4, DLC = (unsigned int) 8,
    DOS = (unsigned int) 16, KIC = (unsigned int) 32, BP = (unsigned int) 64, CM = (unsigned int) 128
};

//BELL
if ((input & 6) == 2){
    output = output | 1;
}

if((input & (ER + DSBF)) == 2){
    output |= 1;
}

//DLA
if((input & (DOS + KIC)) == 16){
    output &= ~2;
}

if((input & (DOS + DLC)) == 24){
    output |= 2;
}else{
    output &= ~2;
}

//BA
if((input & (BP + CM)) == 192){
    output |= 4;
}else{
    output &= ~4;
}
```

b) 

```
[pou14@linux2 lab1_src]$ gcc lab1_prob4_framework.c -o lab1_prob4_framework.out
[pou14@linux2 lab1_src]$ ./lab1_prob4_framework.out
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
```

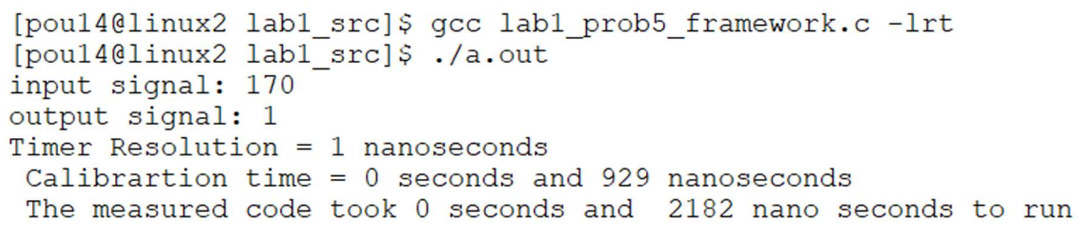
Problem 5:

a) Compile time using the code from problem 3:



```
[pou14@linux2 lab1_src]$ gcc lab1_prob5_framework.c -lrt
[pou14@linux2 lab1_src]$ ./a.out
Driver Seat Belt Fastened?      1
Enginer Running?        0
Doors Closed?   1
Door Lock Leaver activated?     0
Driver on the Seat?     0
Key in Car?     0
Break Pedal activated?  1
Car Moving?     1

(BELL/DLA/BA):   (0/0/1)
Timer Resolution = 1 nanoseconds
 Calibrartion time = 0 seconds and 1154 nanoseconds
 The measured code took 0 seconds and  1155 nano seconds to run
```

b) Compile time using the code from problem 4:



```
[pou14@linux2 lab1_src]$ gcc lab1_prob5_framework.c -lrt
[pou14@linux2 lab1_src]$ ./a.out
input signal: 170
output signal: 1
Timer Resolution = 1 nanoseconds
 Calibrartion time = 0 seconds and 929 nanoseconds
 The measured code took 0 seconds and  2182 nano seconds to run
```

c) It seems like the code from problem 4 (bit masking) compiled faster than the code from problem 3, around 200 nano seconds faster.