Problem 1

a) The statement in “Tag1” creates an integer variable named “int_var” and the statement in “Tag2” prints the size of “int_var”, in other words how many bits the variable occupies in the memory

b) The sizeof() function receives as parameters an identifier name or a datatype and give as output the size of the datatype or the size of the datatype of the variable associated to the identifier. This is a standard library, I recompiled the program without including sys libraries and the sizeof function still worked

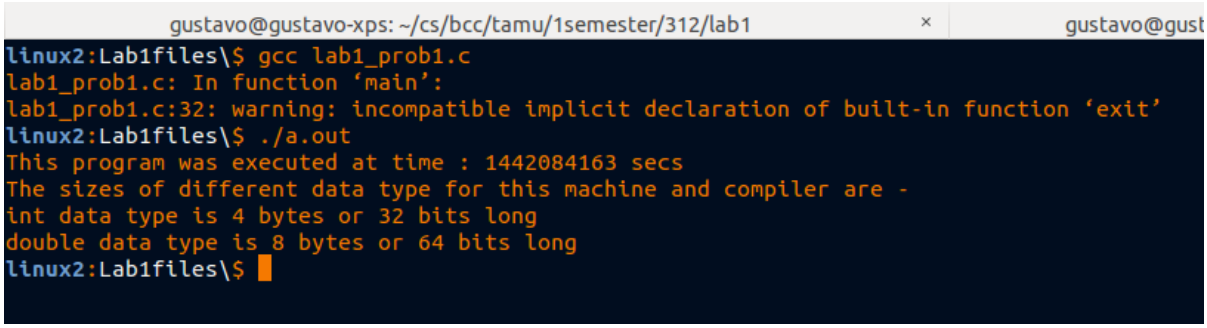
c) Print:

```
linux2:Lab1files\$ ./a.out
This program was executed at time : -254189366 secs
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
```

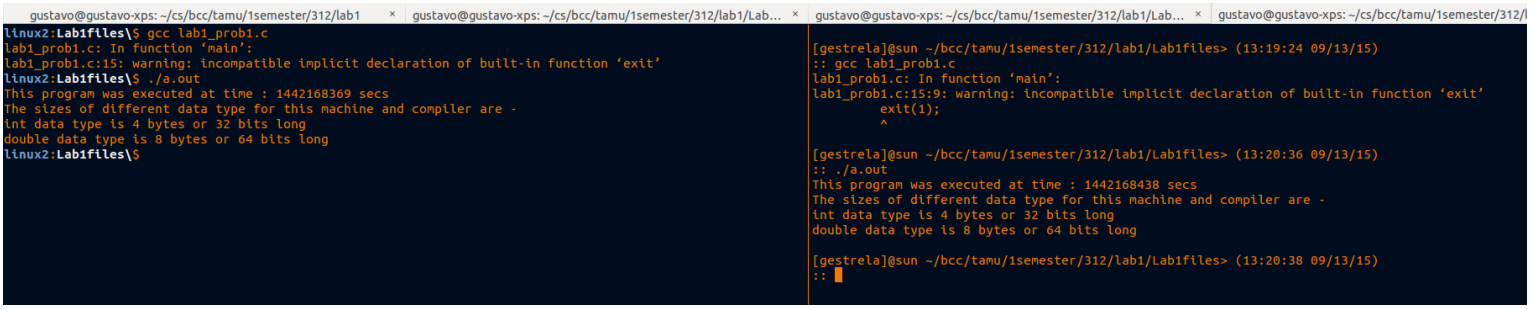
d) Print:

```
[gestrela]@sun ~/bcc/tamu/1semester/312/lab1/Lab1files> (12:02:33 09/07/15)
:: ./a.out
This program was executed at time : 1104509808 secs
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
```

e) The value was negative because we used double variable and tried to print it as an integer, but integer and double variables have different representations in memory. A possible fix for that is using a casting “(int)” before the variable, which will put the double value in a temporary place using the int data representation. The problem is that, this fix won’t work for numbers bigger then the biggest positive number an int can represent (2^31 – 1). We can also try using “unsigned” and “long”, but the type double would still be able to represent numbers bigger then the biggest integer.
Using only casting we get the result:



f) The values are consistent with the one observed in the fixed version of the algorithm seen on question e).



g) The timeval structure may have different implementations or different platforms. On windows, timeval is defined as:

typedef struct timeval {

long tv_sec;

long tv_usec;

} timeval;

While in linux:

struct timeval {

time_t tv_sec;

suseconds_t tv_usec;

};

Problem 2

- The type long is actually “long int”, otherwise long is a prefix for other types that creates a new type, which is not necessarily bigger than the prefixed type.
- The type “long long” has the same size as “long”. Since “long long” can store larger number than “long” there are probably gaps between the integers that “long long” can store.
- Since the “long” and “long long” types have the same size, either two option could occour: there could be gaps between integers that can be representable in “long long“ or “long long“ and “long“ could represent the same range of numbers. As saw in limits.h, the “long long“ and “long” represent the same range of number in this case.

```
/* Minimum and maximum values a `signed long int' can hold.  */
#  if __WORDSIZE == 64
#   define LONG_MAX     9223372036854775807L
#  else
#   define LONG_MAX     2147483647L
#  endif
#  define LONG_MIN      (-LONG_MAX - 1L)

/* Maximum value an `unsigned long int' can hold.  (Minimum is 0.)  */
#  if __WORDSIZE == 64
#   define ULONG_MAX    18446744073709551615UL
#  else
#   define ULONG_MAX    4294967295UL
#  endif

#  ifdef __USE_ISOC99

/* Minimum and maximum values a `signed long long int' can hold.  */
#   define LLONG_MAX    9223372036854775807LL
#   define LLONG_MIN    (-LLONG_MAX - 1LL)

/* Maximum value an `unsigned long long int' can hold.  (Minimum is 0.)  */
#   define ULLONG_MAX   18446744073709551615ULL
```

Problem 3

a) Boolean functions for actuators
   a. BELL = ER * (DSBF)’
   b. BELL = ER * (DC)’
   c. BELL = ER * ((DSBF)’ + (DC)’)
   d. DLA = DLC * ((DOS)’ * (KIC))’ = DLC * ((KIC)’ + DOS)
   e. BA = BP * CM

b) Truth table:

ER	DSBF	DC	DLC	KIC	DOS	BP	CM	BELL	DLA	BA
0	x	x	x	x	x	x	x	0	x	x
1	0	x	x	x	x	x	x	1	x	x
1	1	0	x	x	x	x	x	1	x	x
1	1	1	x	x	x	x	x	0	x	x
x	x	x	0	x	x	x	x	x	0	x
x	x	x	1	0	x	x	x	x	1	x
x	x	x	1	1	0	x	x	x	0	x
x	x	x	1	1	1	x	x	x	1	x
x	x	x	x	x	x	0	x	x	x	0
x	x	x	x	x	x	1	0	x	x	0
x	x	x	x	x	x	1	1	x	x	1

Problem 4

b) The use of enum improved the readability of the code because it give “names” to masks used in the code. In my code, the function mask (x, b) (returns 1 if x & b) have its calls “better explained” because, instead of using a number, we use constants with meaningfull names. If, by instance we would like to know if the ER bit is active in x, we should use mask (x, ERM). Other pro of using enum is that, if we would like to change the order of the bits we wouldn't need to modify all the code, but just the enumeration.

Problem 5

a) The execution time are very different in both machines, and the calibration time is almost the same.

b) No, because the sun machine was faster with the program from problem 3. I also believe that running the code only once is not enough to determine which code is faster, and for a fair comparison we should make a lot of repetitions measuring the running time. Also, the code in the problem 3 involves the read of 8 variables from the standard in while the program from problem 4 reads only one, therefore changing the way problem 3 reads the input without changing the core of the code (the part that evaluates the outputs) may make the problem 3 program faster than problem 4 program.

c) Changing the processor will probably make the execution faster, but changing the processor to make it faster have physical limits and may cost. I this case changing the implementation may be a better solution if possible.

d) A first and more simple solution would be changing the implementation of the break system to a circuit using logic gates. Other solution would be make a different program for the break system, but this solution would depend on how processes run in the microprocessors used.

Problem 6

b) The speed of the circuits are: 10.2 ns for the BELL; 13.6 for the DLA and 6.8 ns for the brakes, totaling 30.6 for all the circuits and also fulfilling the brake speed circuit requirement.

c) Yes, the use of a combinational circuit, when the functions are simple like in our example, makes the development of the system simpler and faster.