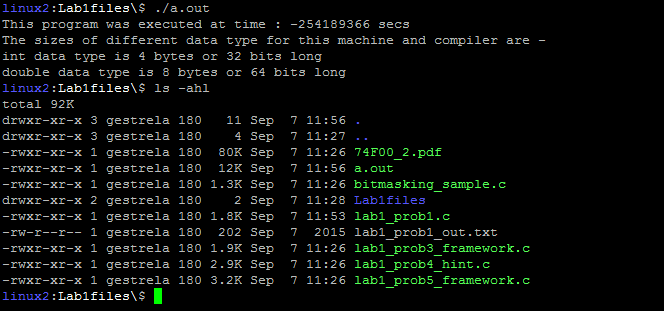
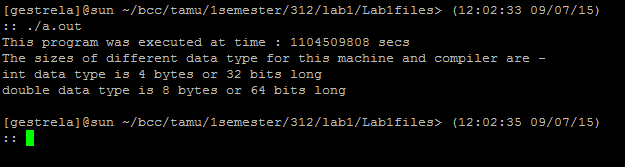
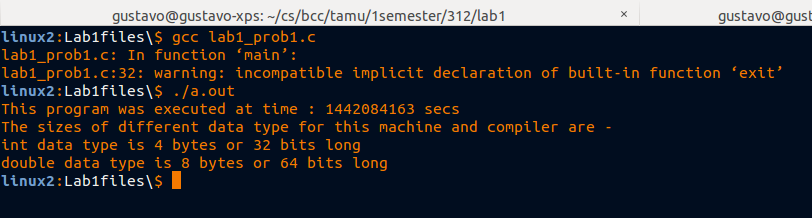
**eGustavo Estrela de Matos CSCE312 LAB1 7 September 2015**

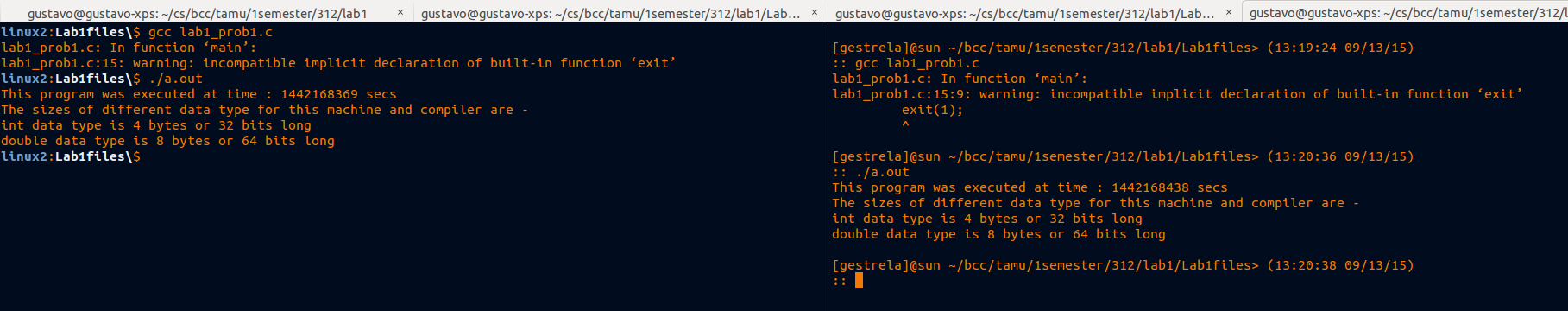
**Problem 1**

1. 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
2. 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
3. Print:
4. Print:
5. 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:



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



1. 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**

1. Boolean functions for actuators
   1. BELL = ER \* (DSBF)’
   2. BELL = ER \* (DC)’
   3. BELL = ER \* ((DSBF)’ + (DC)’)
   4. DLA = DLC \* ((DOS)’ \* (KIC))’ = DLC \* ((KIC)’ + DOS)
   5. BA = BP \* CM
2. Truth table:



**Problem 4**

a) 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.