EXERCISES UNIT 5

1. In C++11 a new class template has been defined, tuple (defined in <tuple>), that allows to define a tuple of data types. Using this class, define the function dividir(x,y) that returns in a tuple the quotient and the remainder of the division of x by y. Write an example of the use of the function.

2. We have created a new class Complejo that represents complex numbers. Write a traits class for this new type similar to the type_traits. It is enough to write only 2 or 3 characteristics.

3. Write a function in C++ that sums the elements of a container of the STL passing as input parameters the begin iterator and end iterator. The elements can be of any summable type (integers, reals, complex numbers, …). Remember that all iterators of the STL have defined an ***iterator_traits.*** Assume that the result of the addition always fits in the same data type.

4. Write a function template for creating a container of random numbers of any integer type in the range given by the parameters *begin* and *end*. The template parameter is the type of the container. Prevent the call to this function with a type different from an integer type by using *static asserts*.

5. Write a function template that calculates the mean of the elements of a container. The function should only receive iterators to the container. Write a policy class to allow the calculation of the geometric mean and the harmonic mean with the same function.

6. Write a class template (or function template) that converts an integer representing a binary number in its equivalence in base 10 (1101 -> 13). The calculation should be done in compilation time.

7. Rewrite the previous function using a constexpr function (in C++11).

8. We have a program that executes many times the product of the elements of small vectors. With the idea of optimizing the program, you should write a template that unfolds the product.

9. We have a program that executes many times the initialization of small C-style vectors. With the objective of optimizing the program, you should write a template that unfolds the following initialization loop.

```
template<typename T>
inline void inicializa(T* v, T x)
{
  for(unsigned i=0; i<TAM; i++)
    v[i] = x;
}
```

10. Modify the exercise 4 to allow integer and real types. You will have to create two functions with the same name and enable only one of them depending on the type of the container. You will need to use *enable_if* of the library *type_traits*. The *enable_if* construction will substitute the return type of the functions.