## Recursive Functions Lab 2

Write the following functions:

1. functions over Bool: and logical, or logical, negation, nand, cloud, implication, double implication;

```
and :: Bool -> Bool -> Bool
and False _ = False
and _ False = False
and _ _ = True
```

2. primality test: isPrime :: Integer -> Bool;

Use a helper function hasDivisors so that hasDivisors n a b tests if the natural number n has divisors between the natural numbers a and b.

```
3. hasDivisors :: Integer -> Integer -> Integer -> Bool
hasDivisors n a b | a > b = False
hasDivisors n a b | n 'mod' a == 0 = ...
hasDivisors n a b = ...
isPrime :: Integer -> Bool
isPrime n = hasDivisors n ...
```

- 4. implement various algorithms for computing the gcd (Euclid's algorithm by repeated subtractions/divisions, the algorithm binary).
- 5. it is possible to apply an optimization for fetching recursive calls in tail position?
- 6. implement algorithms for computing the nth number Fibonacci.

```
fibo :: Integer -> Integer
fibo ... = ...
...
```

Implement the version that uses accumulators:

```
fiboaux :: Integer -> Integer -> Integer -> Integer
fiboaux 0 a _ = a
fiboaux n a b = fiboaux ... ...
-- a si b sunt doua numere Fibonacci consecutive
fibo' :: Integer -> Integer
fibo' n = fiboaux n 0 1
```

Implement the version that works in time O(log(n)).

- 7. implement the extended Euclidian algorithm.
- 8. implement the function succ :: Integer -> Integer.
- 9. recursively (even if they are inefficient) implement the following functions: addition of two natural numbers, using succ; multiplication of two natural numbers, using addition;
   power, using multiplication.
- 10. implement the mod and div functions for natural numbers.