# Segurança em Computação
# Trabalho Individual II

João Paulo Taylor Ienczak Zanette

13 de junho de 2019

**Quanto ao entregável**
Você deve entregar no Moodle um único arquivo PDF com os seguintes requisitos:

- Uma seção para **Números Aleatórios**;
- Uma seção para **Números Primos**;
- Os códigos devem estar no PDF e devidamente documentados;
- Referências a cada um dos algoritmos.

OBS: Incluir tabelas, saídas, etc.

## 1 Números Aleatórios

Algoritmo escolhido: Xorshift com $(a, b, c) = (13, 7, 17)$.

| Nº de bits | Tempo (ms) |
|---|---|
| 32 | 0.0045 |
| 64 | 0.0050 |
| 128 | 0.0056 |
| 256 | 0.0053 |
| 512 | 0.0064 |
| 1024 | 0.0054 |
| 2048 | 0.0050 |
| 4096 | 0.0052 |

Tabela 1: Temporização para o algoritmo de Xorshift. O tempo foi calculado utilizando uma média aritmética de 1000 execuções.

## 2 Números Primos

## 3 Código

```
'''Module for Random Number Generation functions.'''
from dataclasses import dataclass
from enum import auto, Enum
from functools import partial
```

```
from operator import mul
from typing import Callable


@dataclass
class RNG:
    '''Generates new random numbers from range [0, limit) through iteration.

    function: The step function (which shall give the next random number from a
              given seed).
    seed: Initial seed.
    limit: The upper limit of generated numbers range.
    '''
    function: Callable[[int], int]
    seed: int = 88172645463325252
    limit: int = 2 ** 32

    def __iter__(self):
        return self

    def __next__(self):
        self.seed = self.function(seed=self.seed) % self.limit
        return self.seed


#-------------------------------------------------------------------------------
# Xorshift
#-------------------------------------------------------------------------------
def xorshift(
        seed: int = RNG.seed,
        a: int = 13,
        b: int = 7,
        c: int = 17
) -> int:
    '''Gives next random number from given seed using Xorshift algorithm. (a,
    b, c) are Xorshift's specific parameters.'''

    parts, i, n = [], 0, seed

    # Works in blocks of 32 bits
    while n != 0:
        _n = n & 0xffffffff

        _n ^= _n << a
        _n ^= _n >> b
        _n ^= _n << c
        _n = _n & 0xffffffff

        parts.append(_n)
        i += 1
        n >>= 32

    n = 0
    for i, part in enumerate(parts):
```

```
        n |= part << (32 * i)

    return n


class BitLength(Enum):
    '''How many bits the generated numbers should have.'''
    RNG32 = 32
    RNG64 = 64
    RNG128 = 128
    RNG256 = 256
    RNG512 = 512
    RNG1024 = 1024
    RNG2048 = 2048


_XORSHIFT_PARAMS = {
    BitLength.RNG32: [13, 17, 5],
    BitLength.RNG64: [13, 7, 17],
    BitLength.RNG128: [19, 11, 8],
    BitLength.RNG256: [49, 21, 28],
    BitLength.RNG512: [252, 42, 148],
    BitLength.RNG1024: [601, 465, 957],
    BitLength.RNG2048: [1601, 865, 1257],
}


def xorshift_init(bits: BitLength = BitLength.RNG64, seed=RNG.seed):
    '''Initializes RNG with xorshift using preset parameters.'''
    a, b, c = _XORSHIFT_PARAMS[bits]
    bits = bits.value
    return RNG(function=partial(xorshift, a=a, b=b, c=c), limit=2**bits - 1)
```