3. (a) Implementá el TAD Cola utilizando la siguiente representación, donde N es una constante de tipo nat:

implement Queue **of** T **where**

type Queue **of** T = **tuple**
elems : array[0..N-1] **of** T
size : nat
end tuple

```
implement Queue of T where

type Queue of T = tuple
                    elems: array[0..N-1] of T
                    size: nat
                  end tuple

constructors
    fun empty_queue() ret q: Queue of T
        q.size := 0
    end fun

    proc enqueue(in/out q: Queue of T, in e: T)
        q.size := q.size + 1
        q.elems[q.size] := e
    end proc

operations
    fun is_empty_queue(q: Queue of T) ret b: bool
        b := (q.size = 0)
    end fun

    {- PRE: not is_empty_queue(q) -}
    fun first(q: Queue of T) ret e: T
        e := q.elems[0]
    end fun

    {- PRE: not is_empty_queue(q) -}
    proc dequeue(in/out q: Queue of T)
        for i := 0 to N-1 do
            q.elems[i] := q.elems[i+1]
        od
        q.size := q.size - 1
    end proc

end implement
```

(b) Implementá el TAD Cola utilizando un arreglo como en el inciso anterior, pero asegurando que todas las operaciones estén implementadas en orden constante.

Ayuda1: Quizás convenga agregar algún campo más a la tupla. ¿Estamos obligados a que el primer elemento de la cola esté representado con el primer elemento del arreglo?

Ayuda2: Buscar en Google *aritmética modular*.

```
implement Queue of T where

type Queue of T = tuple
                    elems: array[0..N-1] of T
                    size: nat
                    start: nat
                  end tuple

constructors
    fun empty_queue() ret q: Queue of T
        q.size := 0
        q.start := 0
    end fun

    proc enqueue(in/out q: Queue of T, in e: T)
        var last: nat

        last := q.size `mod` N
        q.elems[last] := e
        q.size := q.size + 1
    end proc

operations
    fun is_empty_queue(q: Queue of T) ret b: bool
        b := (q.size = 0)
    end fun

    {- PRE: not is_empty_queue(q) -}
    fun first(q: Queue of T) ret e: T
        e := q.elems[q.start]
    end fun

    {- PRE: not is_empty_queue(q) -}
    proc dequeue(in/out q: Queue of T)
        if q.start = N-1 then
            q.start := 0
        else
            q.start := q.start + 1
            q.size := q.size - 1
        fi
    end proc

end implement
```