1. Completá la implementación de listas dada en el teórico usando punteros.

```
implement List of T where

type Node of T = tuple
                   elem: T
                   next: pointer to (Node of T)
                 end tuple

type List of T = pointer to (Node of T)

constructors
    fun empty_list() ret l: List of T
        l := null
    end fun

    proc addl(in/out l: List of T, in e: T)
        var p: pointer to (Node of T)

        alloc(p)
        p→elem := e
        p→next := l
        l := p
    end proc

destroy
    proc destroy_list(in/out l: List of T)
        var p: pointer to (Node of T)

        while l ≠ null do
            p := l→next
            l := p
            free(l)
        od
    end proc

operations
    fun is_empty_list(l: List of T) ret b: bool
        b := (l = null)
    end fun

    {- PRE: not is_empty_list(l) -}
    fun head(l: List of T) ret e: T
        e := l→elem
    end fun

    {- PRE: not is_empty_list(l) -}
    proc tail(in/out l: List of T)
        var p: pointer to (Node of T)

        p := l
        l := l→next
        free(p)
    end proc

    proc addr(in/out l: List of T, e: T)
        var p,q: pointer to (Node of T)
```

```
        alloc(q)
        q→elem := e
        q→next := null

        if not is_empty_list(l) then
            p := l
            while p→next ≠ null do
                p := p→next
            od
            p→next := q
        else
            l := q
        fi
    end proc

    fun length(l: List of T) ret n: nat
        var p: pointer to (Node of T)

        n := 0
        p := l

        while p ≠ null do
            n := n+1
            p := p→next
        od
    end fun

    proc concat(in/out l: List of T, in l0: List of T)
        var p: pointer to (Node of T)

        if is_empty_list(l) then
            l := l0
        else
            p := l
            while p→next ≠ null do
                p := p→next
            od
            p→next := l0
        fi
    end proc

    {- PRE: length(l) > n -}
    fun index(l: List of T, n: nat) ret e: T
        var p: pointer to (Node of T)

        p := l
        for i := 1 to n do
            p := p→next
        od
        e := p→elem
    end fun

    proc take(in/out l: List of T, in n: nat)
        var p,q: pointer to (Node of T)
        var i: nat

        i := 0
```

```
        if not is_empty_list(l) then
            if n = 0 then
                destroy_list(l)
            else if n > 0 then
                p := l
                while i < n ∧ not is_empty_list(l) do
                    p := p→next
                    i := i+1
                od

                while p ≠ null do
                    q := p
                    p := p→next
                    free(q)
                od
            fi
        fi
    end proc

    proc drop(in/out l: List of T, in n: nat)
        var p: pointer to (Node of T)
        var i: nat

        i := 0

        while i < n ∧ not is_empty_list(l) do
            p := l
            l := p→next
            free(p)
            i := i+1
        od
    end proc

    fun copy_list(l1: List of T) ret l2: List of T
        var p: pointer to (Node of T)
        var copy_size: nat

        copy_size := length(l1)

        if is_empty_list(l1) then
            l2 := empty_list()
        else
            alloc(l2)
            p := l1

            for i := 1 to copy_size do
                l2→elem := p→elem
                l2→next := p→next
                p := p→next
            od
        fi
    end fun

end implement
```