6. Implementá el TAD Conjunto finito de elementos de tipo T utilizando:

   (a) una lista de elementos de tipo T, donde el constructor para agregar elementos al conjunto se implementa directamente con el constructor **addl** de las listas.

```
implement Set of T where

type Set of T = List of T

constructors
    fun empty_set() ret s: Set of T
        s := empty_list()
    end fun

    {- PRE: e ∉ s -}
    proc add(in/out s: Set of T, in e: T)
        addl(s,e)
    end proc

destroy
    proc destroy_set(in/out s: Set of T)
        destroy_list(s)
    end proc

operations
    fun member(s: Set of T, e: T) ret b: bool
        var i: nat

        b := false
        i := 0

        while i < length(s) ∧ not b do
            b := index(s,i) = e
            i := i+1
        od
    end fun

    fun is_empty_set(s: Set of T) ret b: bool
        b := length(s) = 0
    end fun

    proc union(in/out s1: Set of T, in s2: Set of T)
        var i,j: nat
        var exists: bool

        i := 0

        while i < length(s2) do
            exists := false
            j := 0

            while j < length(s1) ∧ exists = false do
                if index(s1,j) = index(s2,i) then
                    exists := true
                fi
                j := j+1
            od
```

```
        if exists = false then
            addr(s1,index(s2,i))
        fi
        i := i+1
    od
end proc

proc inters(in/out s1: Set of T, in s2: Set of T)
    var s_aux: Set of T
    var i,j: nat
    var exists: bool

    s_aux := empty_list()
    i := 0

    while i < length(s1) do
        exists := false
        j := 0

        while j < length(s2) ∧ exists = false do
            if index(s1,i) = index(s2,j) then
                exists := true
            fi
            j := j+1
        od

        if exists = true then
            addr(s_aux,index(s1,i))
        fi
        i := i+1
    od

    s1 := s_aux
    destroy_list(s_aux)
end proc

proc diff(in/out s1: Set of T, in s2: Set of T)
    var s_aux: Set of T
    var i,j: nat
    var exists: bool

    s_aux := empty_list()
    i := 0

    while i < length(s1) do
        exists := false
        j := 0

        while j < length(s2) ∧ exists = false do
            if index(s1,i) = index(s2,j) then
                exists := true
            fi
            j := j+1
        od

        if exists = false then
            addr(s_aux,index(s2,j))
        fi
```

```
            i := i+1
        od

        s1 := s_aux
        destroy_list(s_aux)
    end proc

end implement
```

---

(b) una lista de elementos de tipo T, donde se asegure siempre que la lista está ordenada crecientemente y no tiene elementos repetidos. Debes tener cuidado especialmente con el constructor de agregar elemento y las operaciones de unión, intersección y diferencia. A la propiedad de mantener siempre la lista ordenada y sin repeticiones le llamamos *invariante de representación*.

Ayuda: Para implementar el constructor de agregar elemento puede serte muy útil la operación *add_at* implementada en el punto 3.

```
implement Set of T where

type Set of T = List of T

constructors
    fun empty_set() ret s: Set of T
        s := empty_list()
    end fun

    {- PRE: e ∉ s -}
    proc add(in/out s: Set of T, in e: T)
        var s_aux: Set of T
        var n: nat

        s_aux := copy_list(s)
        n := 0

        while not is_empty_list(s_aux) ∧ head(s_aux) < e do
            n := n+1
            tail(s_aux)
        od

        if is_empty_list(s_aux) ∨ head(s_aux) > e then
            add_at(s,n,e)
        fi

        destroy_list(s_aux)
    end proc

destroy
    proc destroy_set(in/out s: Set of T)
        destroy_list(s)
    end proc

operations
    fun member(s: Set of T, e: T) ret b: bool
        var i: nat

        b := false
        i := 0
```

```
        while i < length(s) ∧ not b do
            b := index(s,i) = e
            i := i+1
        od
    end fun

    fun is_empty_set(s: Set of T) ret b: bool
        b := length(s) = 0
    end fun

    proc union(in/out s1: Set of T, in s2: Set of T)
        var i,j: nat
        var n: nat
        var exists: bool
        var s1_aux: Set of T

        i := 0

        while i < length(s2) do
            exists := false
            j := 0

            while j < length(s1) ∧ exists = false do
                if index(s1,j) = index(s2,i) then
                    exists := true
                fi
                j := j+1
            od

            if exists = false then
                s1_aux := copy_list(s1)
                n := 0

                while not is_empty_list(s1_aux) ∧ head(s1_aux) < index(s2,i) do
                    n := n+1
                    tail(s1_aux)
                od

                if is_empty_list(s_aux) ∨ head(s_aux) > e then
                    add_at(s1,n,index(s2,i))
                fi

                destroy_list(s1_aux)
            fi
            i := i+1
        od
    end proc

    proc inters(in/out s1: Set of T, in s2: Set of T)
        var s_aux: Set of T
        var i,j: nat
        var exists: bool

        s_aux := empty_list()
        i := 0

        while i < length(s1) do
            exists := false
```

```
            j := 0

            while j < length(s2) ∧ exists = false do
                if index(s1,i) = index(s2,j) then
                    exists := true
                fi
                j := j+1
            od

            if exists = true then
                addr(s_aux,index(s1,i))
            fi
            i := i+1
        od

        s1 := s_aux
        destroy_list(s_aux)
    end proc

    proc diff(in/out s1: Set of T, in s2: Set of T)
        var s_aux: Set of T
        var i,j: nat
        var exists: bool

        s_aux := empty_list()
        i := 0

        while i < length(s1) do
            exists := false
            j := 0

            while j < length(s2) ∧ exists = false do
                if index(s1,i) = index(s2,j) then
                    exists := true
                fi
                j := j+1
            od

            if exists = false then
                addr(s_aux,index(s2,j))
            fi
            i := i+1
        od

        s1 := s_aux
        destroy_list(s_aux)
    end proc

end implement
```