

## Ejercicio práctico a entregar: TAD Conjunto Finito

Dada la siguiente especificación del TAD Lista, completar la implementación del **TAD Conjunto finito** con el código correspondiente a las operaciones *cardinal*, *member*, *elim* e *inters*.

Representamos un conjunto con una lista, asegurando en la implementación de constructores y operaciones que la misma siempre está **ordenada crecientemente** y sin **repeticiones**. Se puede observar el cumplimiento de este *invariante de representación* en la implementación del constructor **add**, en el cual primero buscamos el lugar donde debe ir el nuevo elemento si es que no está, y lo agregamos mediante la operación `list_add_at`. Al completar las operaciones que faltan se debe respetar el invariante.

Como ejemplo consideremos las siguientes listas de enteros:

1. [1,6,3,8] ----> NO cumple invariante
2. [1,3,4,9] ----> SI cumple invariante
3. [1,3,3,5] ----> NO cumple invariante

Las listas 1 y 3 no cumplen el invariante de representación: la primera no tiene sus elementos en orden creciente y la tercera tiene el elemento 3 repetido. La segunda sí lo cumple y representaría el conjunto {1,4,3,9}. Si quiero unir a este conjunto el {2,4,3,7}, la lista correspondiente que representa dicha unión sería [1,2,3,4,7,9]. Y si los quiero intersectar, la lista que lo representa sería [3,4].

Es fundamental escribir los algoritmos correctamente en el lenguaje de la materia y respetar la abstracción del TAD Lista. No conocemos su representación interna sino sólo las operaciones que provee, y con ellas debe implementarse el TAD Conjunto finito. Recomendación **fuerte** leer primero todo el código que les pasamos antes de implementar las operaciones que se piden.

**Fecha máxima de entrega:** Lunes 4 de Mayo de 2020

**spec List of T where**

**constructors**

```
fun list_empty() ret l : List of T
  {- Crea una lista sin ningún elemento -}
```

```
proc addl(in/out l : List of T, in e : T)
  {- Agrega el elemento e al comienzo de la lista l -}
```

**operations**

```
proc addr(in/out l : List of T, in e : T)
  {- Agrega el elemento e al final de la lista l -}
```

```
fun length(l : List of T) ret n : nat
  {- Devuelve la cantidad de elementos que tiene l -}
```

```
fun is_empty(l : List of T) ret b : bool
  {- Devuelve True si l no tiene ningún elemento -}
```

```
fun head(l : List of T) ret e : T
  {- Devuelve el primer elemento de la lista l -}
  {- PRE: not is_empty(l) -}
```

```
proc tail(in/out l : List of T)
  {- Elimina el primer elemento de la lista l -}
  {- PRE: not is_empty(l) -}
```

```
proc concat(in/out l : List of T, in l' : List of T)
  {- Le agrega al final de l todos los elementos de l' en el mismo
  orden -}
```

```
fun index(l : List of T, n : nat) ret e : T
  {- Devuelve el elemento de l que se encuentra en la posición n -}
  {- PRE: n < length(l) -}
```

```
proc take(in/out l : List of T, in n : nat)
  {- Deja en l sólo los primeros n elementos, eliminando el resto. -}
```

```
proc drop(in/out l : List of T, in n : nat)
  {- Elimina los primeros n elementos de l -}
```

```
proc list_add_at(in/out l : List of T, in n : nat, in e : T)
  {- Agrega a la lista l el elemento e en la posición n -}
  {- PRE: n < length(l) -}
```

```
proc list_elim_at(in/out l : List of T, in n : nat)
  {- Elimina de la lista l el elemento ubicado en la posición n -}
  {- PRE: n < Length(L) -}
```

```
fun list_copy(l1 : List of T) l2 : List of T
  {- Copia la lista l1 -}
```

```
proc list_destroy(in/out l : List of T)
  {- Libera memoria en caso que haya sido pedida -}
```

**spec Set of T where**

**constructors**

```
fun empty_set() ret s : Set of T
  {- Crea un conjunto vacío -}
```

```
proc add(in/out s : Set of T, in e : T)
  {- Agrega el elemento e al conjunto s -}
```

**operations**

```
fun cardinal(s : Set of T) ret n : nat
  {- Devuelve la cantidad de elementos que tiene s -}
```

```
fun is_empty_set(s : Set of T) ret b : bool
  {- Devuelve True si s es vacío -}
```

```
fun member(e : T, s : Set of T) ret b : Bool
  {- Devuelve True si el elemento e pertenece al conjunto s -}
```

```
proc elim(in/out s : Set of T, in e : T)
  {- Elimina el elemento e del conjunto s, en caso que esté -}
```

```
proc union(in/out s : Set of T, in s0 : Set of T)
  {- Agrega a s todos los elementos de s0 -}
```

```
proc inters(in/out s : Set of T, in s0 : Set of T)
  {- Elimina de s todos los elementos que NO pertenezcan a s0 -}
```

```
proc dif(in/out s : Set of T, in s0 : Set of T)
  {- Elimina de s todos los elementos que pertenecen a s0 -}
```

```
fun get(in/out s : Set of T) ret e : T
  {- Obtiene algún elemento cualquiera del conjunto s -}
  {- PRE: not is_empty_set(s) -}
```

```
fun set_copy(s1 : Set of T) s2 : Set of T
  {- Copia el conjunto s1 -}
```

```
proc set_destroy(in/out s : Set of T)
  {- Libera memoria en caso que haya sido pedida -}
```

```
{- IMPLEMENTACIÓN -}
```

```
implement Set of T where
```

```
type Set of T = List of T
```

```
fun empty_set() ret s : Set of T  
  s := list_empty()  
end fun
```

```
proc add(in/out s : Set of T, in e : T)
```

```
  var n : nat  
  var saux : Set of T  
  var is_member : bool  
  var d : T  
  n := 0  
  saux := list_copy(s)  
  is_member := false
```

```
  {- encuentro el lugar donde va el elemento -}
```

```
  do (not is_empty(saux) /\ not is_member)
```

```
    d := head(saux)
```

```
    if d = e -> is_member := true
```

```
      d < e -> n := n+1
```

```
      d > e -> skip
```

```
    fi
```

```
    tail(saux)
```

```
  od
```

```
  if (not is_member)
```

```
    then list_add_at(s,n,e)
```

```
  fi
```

```
  list_destroy(saux)
```

```
end proc
```

```
fun cardinal(s : Set of T) ret n : nat
```

```
  {- COMPLETAR -}
```

```
end fun
```

```
fun is_empty_set(s : Set of T) ret b : bool
```

```
  b := is_empty_list(s)
```

```
end fun
```

```
fun member(e : T, s : Set of T) ret b : bool
```

```
  {- COMPLETAR -}
```

```
end fun
```

```

proc elim(in/out s : Set of T, in e : T)
  {- COMPLETAR -}
end proc

proc union(in/out s : Set of T, in s0 : Set of T)
  var saux : List of T
  var d : T

  saux := copy_list(s0)

  do (not is_empty_list(saux))
    d := head(saux)
    add(d,s)
    tail(saux)
  od
  list_destroy(saux)

end proc

proc inters(in/out s : Set of T, in s0 : Set of T)
  {- COMPLETAR -}
end proc

proc dif(in/out s : Set of T, in s0 : Set of T)
  var saux : List of T
  var d : T

  saux := copy_list(s0)

  do (not is_empty_list(saux))
    d := head(saux)
    if (member(d,s))
      then elim(s,d)
    fi
    tail(saux)
  od
  list_destroy(saux)

end proc

{- PRE: not is_empty_set(s) -}
fun get(s : Set of T) ret e : T
  e := head(s)
end fun

fun set_copy(s1 : Set of T) s2 : Set of T
  s2 := copy_list(s1)
end fun

```

```
proc set_destroy(in/out s : Set of T)
  destroy_list(s)
end proc
```