

② type List of T = tuple

listToArray : array[1..m] of T  
tam : nat

end tuple

Constructors

fun empty ( ) ret l : List of T  
l.tam := 0

end fun

Proc addl (in e : T, in/out l : List of T)

l.listToArray[N-1.tam] := e  
l.tam := tam + 1

end Proc

③ Proc (in/out l : List of T, in m : nat, in e : T)

Var laux1, laux2 : List of T

1 2 3 4 5  
1 2

laux1 := copy(l)

laux2 := copy(l)

take(laux1, m) } - laux1 almacena los primeros m elementos - }

drop(laux2, m) } - laux2 almacena los elementos a partir de m-ésimo - }

addl(laux1, e)

concat(laux1, laux2)

l := copy(laux1)

destroy(laux1)

destroy(laux2)

end Proc

5) spec set of  $T$  where

constructors

fun empty() ret  $S$  set of  $T$   
{- CREA UN CONJUNTO VACÍO -}

Proc add( $in e: T, in/out s: set of T$ )  
{- AGREGA EL ELEMENTO  $e$  AL CONJUNTO -}

destroy

Proc destroy( $in/out s: set of T$ )  
{- LIBERA MEMORIA EN CASO DE SET MEMORIA -}

Operations

fun pertenece( $e: T, c: set of T$ ) ret  $b: bool$   
{- Devuelve true si  $e$  pertenece al conj  $c$  -}

fun is\_empty( $c: set of T$ ) ret  $b: bool$   
{- Devuelve true si el conj  $c$  es vacío -}

Proc union( $in/out a: set of T, in b: set of T$ )  
{- AGREGA TODOS LOS ELEMENTOS DE  $b$  AL CONJUNTO  $a$  -}

Proc intersectar( $in/out a: set of T, in b: set of T$ )  
{- ELIMINA LOS ELEMENTOS DE  $a$  QUE NO ESTÉN EN  $b$  -}

6) implement set of  $T$  where

type set of  $T$  = list of  $T$

fun empty() ret  $s: set of T$   
 $s := emptyList$

Proc add( $in e: T, in/out s: set of T$ )

var  $j, length$  nat  
 $length := length(s)$   
 $j := 0$

while  $j < length$  ^  $index(s, j) = e$  do  
 $j := j + 1$   
od

if  $j = length$  v  $index(s, j) \neq e$  then

add\_at( $s, j, e$ )  
fi

end Proc



⑥ b)  $\text{fun pertenece}(c: T, \text{ set of } r) \text{ ret } b: \text{bool}$

```

var i: nat
i := 1
res := false
while (i ≤ index(c, i)) do
    if c = index(c, i) then
        res := true
    fi
    i := i + 1
od
end fun
    
```

$\text{fun is empty}(c: \text{ set of } T) \text{ ret } b: \text{bool}$

$\text{ret} := \text{is\_empty}(c)$

end fun

Proc unir (input a: set of T, in b: set of T)

```

var length: nat
length := length(b)
    
```

```

for i := 1 to length do
    if not pertenece(index(b, i), a) then
        adds(index(b, i), a)
    fi
od
end Proc
    
```

fi  
od  
end Proc