

Ejercicio 1. Quizas la forma mas simple de implementar un conjunto acotado sea mediante un array de tamaño fijo, utilizando la siguiente estructura:

```
modulo ConjAcotadoArr<T> implementa ConjAcotado<T>{
  var datos: Array<T>
  var largo: int
}
```

En la variable *datos* guardaremos los elementos. Como el tamaño del arreglo es fijo, necesitaremos otra variable, a la que llamaremos *largo*, que nos indique cuantas casillas del arreglo *datos* estan siendo usadas. Con esta misma estrucutra tenemos dos opciones, permitir o no que haya repetidos.

- Escriba el invariante de representacion y la funcion de abstraccion para ambos casos.
- ¿Cual es la mas eficiente? En que casos usaria cada una.
- Escriba los algoritmos para las operaciones **agregar** un elemento y **sacar** un elemento para ambas versiones.
- Respecto a la operacion sacar, piense en un algoritmo que no requiera generar un nuevo arreglo para reemplazar datos, sino que lo resuelva modificando alguna de sus posiciones.

```
modulo ConjAcotadoArr<T> implementa ConjAcotado<T>{
  var datos: Array<T>
  var largo: int
  Invariantes de representacion para con y sin repetidos respecticamente:
  pred InvRep (c: ConjAcotadoArr<T>){
     $0 \leq c.largo \leq c.datos.length$ 
  }
  Acá tomo la decision de que los slots vacios de mi array sean los ultimos.
  pred InvRep (c: ConjAcotadoArr<T>){
     $0 \leq c.largo \leq c.datos.length \wedge (\forall i, j : \mathbb{Z}) (0 \leq i, j < c.largo \wedge_L c.datos[i] \neq c.datos[j])$ 
  }
  La funcion de abstraccion es la misma para ambos casos:
  pred Abs (c': ConjAcotadoArr<T>, c: ConjAcotado<T>){
     $c.cap = c'.array.length \wedge_L (\forall e : T) (e \in c.elems \leftrightarrow (\exists i : \mathbb{Z}) (0 \leq i < c'.largo \wedge_L c'.data[i] = e))$ 
  }

  impl agregar(inout c: ConjAcotadoArr<T>, in e: T):{
    if  $c.largo \geq c.datos.length$  then
      return
    else
      if  $c.pertenece(e)$  then
        return
      else
         $c.datos[c.largo] := e$ 
      end if
    end if
  }

  impl agregar(inout c: ConjAcotadoArr<T>, in e: T):{
    if  $c.largo < c.datos.length$  then
       $c.datos[c.largo] := e$ 
    else
      skip
    end if
  }
}
```

```

impl sacar(inout c: ConjAcotadoArr<T>, in e: T):{
    i:= 0
    while c.datos[i] ≠ e && i < c.datos.length do
        i++
    end while
    if c.datos[i]=e then
        c.datos[i]:= c.datos[c.largo]
        c.largo:= c.largo-1
    else
        skip
    end if
}
}

```

Ejercicio 2. ¿Como implementaria una pila no acotada utilizando arreglos?

Escriba la estructura propuesta, su invariante de representacion, funcion de abstraccion y las operaciones **apilar** y **desapilar**

```

modulo PilaArray<T> implementa Pila<T>{
    var datos: Array<T>
    var usados: int
    pred InvRep (c': PilaArray<T>){
        No estoy seguro de necesitar un invariante de representacion.
    }
    pred Abs (c': PilaArray<T>, c: Pila<T>){
        c'.datos.length = c.s.length ∧ (∀i: ℤ) (0 ≤ i < c'.datos.length)
    }
    impl apilar(inout c: PilaArray<T>, in e: T):{
        if c.usados ≤ c.datos.length then
            if c.usados < c.datos.length then
                c.datos[c.usados]:= e
            else

                end if
            else

                end if
            end if
        }
    }
}

modulo Nombre implementa Implementa{
    var :
    pred InvRep (args){
        cuerpo
    }
    pred Abs (args){
        cuerpo
    }
    impl nombre(args):retorna{

    }
}
}

```