1. (a) Dado el siguiente arreglo, muestre cómo queda el mismo luego de cada modificación que realiza el algoritmo **insertion sort**.

$$[3, 17, 2, 10, 5, 8, 3]$$

(b) Escriba un algoritmo que, dado un arreglo y un elemento e que no pertenece al mismo, ubique los menores a e al comienzo del arreglo y devuelva la cantidad de elementos menores a e. Utilice para ello el siguiente encabezado:

proc p (**in/out** a: **array**[1..n] **of Int**, **in** e : **Int**, **out** k : **Nat**)

2. Dado el siguiente procedimiento

```
proc p (in/out l: list)
    var a, b: pointer to node
    a:= l
    while a ≠ null do
        b:= a→next
        if b ≠ null then
            a→next:= b→next
            free(b)
        fi
        a:= a → next
    od
end proc
```

donde los tipos node y list se definen como sigue

```
type node = tuple
                value: elem
                next: pointer to node
            end
type list = pointer to node
```

(a) Explique qué hace el procedimiento p.

(b) ¿Cuál es el orden del procedimiento p? Justificá la respuesta.

(c) Si se llama al procedimiento p con una lista que tiene los valores 1, 2, 3, 4, 5, 6 y 7, en ese orden, ¿qué valores tendrá la lista luego de la llamada?

3. Para el siguiente algoritmo, plantee la recurrencia en función de n y calcule el orden contando la cantidad de asignaciones a la variable t:

```
fun f (n: nat) ret t: nat
    if n ≤ 1 then t:= 1
    else
        t:= f(n/2)
        t:= t + f(n/2)
        for i:= 1 to n do
            for j:= 1 to n do
                t:= t+1
            od
        od
    fi
end fun
```

4. Un **multiconjunto** M con elementos en S, es un subconjunto de S donde cada elemento tiene asociado un número natural que indica cuántas veces el elemento ocurre. En otras palabras, un multiconjunto puede pensarse como un conjunto en el cual los elementos pueden estar incluidos más de una vez, indicando cuántas veces lo hace.

   (a) Especificar el TAD **Multiset of T** de multiconjuntos con elementos de tipo T, incluyendo constructores para crear un multiconjunto vacío y otro para agregar un elemento a un multiconjunto. Y operaciones para saber si un multiconjunto es vacío o no, para saber si un elemento dado existe en el multiconjunto, para obtener cuántas veces un elemento dado ocurre en un multiconjunto, y para eliminar una ocurrencia de un elemento en un multiconjunto.

   (b) Implementar el TAD **Multiset of T** especificado en el punto anterior utilizando como representación interna una lista donde cada elemento es un par consistente de un elemento de tipo **T** y un número natural que indica la cantidad de ocurrencias.

   (c) Implementar una función que reciba un arreglo de N naturales y devuelva el multiconjunto que contiene todos los elementos pares que ocurren en el arreglo. Todo tipo de datos utilizado en la función debe utilizarse de manera **abstracta**.

004 → 40

HOJA N° 1

FECHA

Tomás Achával Benito

Ejercicio 1:

PARTE ORDENADA = ■

a) PASO A PASO DE INSERTION SORT

A = [3, 17, 2, 10, 5, 8, 3] → [3, 17, 2, 10, 5, 8, 3]

i=1 EL SEGMENTO A[1..1] ESTÁ ORDENADO, NO HAGO CAMBIOS

i=2 COMO 17 > 3, EL SEGMENTO A[1..2) ESTÁ ORDENADO, NO HAGO CAMBIOS

→ [3, 17, 2, 10, 5, 8, 3] → [3, 2, 17, 10, 5, 8, 3]

i=3

COMO 2 < 17, EL PROCEDIMIENTO INSERT REALIZA UN SWAP ENTRE ELLOS. LUEGO COMO 2 < 3, EL PROCEDIMIENTO INSERT REALIZA OTRO SWAP Y FINALIZA.

→ [2, 3, 17, 10, 5, 8, 3] → [2, 3, 17, 10, 5, 8, 3]

i=4

COMO 10 < 17, EL PROC INSERT LOS INTERCAMBIA

→ [2, 3, 10, 17, 5, 8, 3] → [2, 3, 10, 17, 5, 8, 3]

i=5

COMO 5 < 17, INSERT LOS INTERCAMBIA

→ [2, 3, 10, 5, 17, 8, 3] → [2, 3, 5, 10, 17, 8, 3]

COMO 5 < 10, INSERT LOS INTERCAMBIA

→ [2, 3, 5, 10, 17, 8, 3] → [2, 3, 5, 10, 8, 17, 3] → [2, 3, 5, 8, 10, 17, 3]

i=6 COMO 8 < 17 y 8 < 10, INSERT LOS INTERCAMBIA EN ORDEN

→ [2, 3, 5, 8, 10, 17, 3] → [2, 3, 5, 8, 10, 3, 17] → [2, 3, 5, 8, 3, 10, 17]

i=7 COMO 3 < 17, 3 < 10, 3 < 8 y 3 < 5 ENTONCES EL ELEMENTO ES "SWAPPEADO" HACIA LA IZQUIERDA 4 VECES.

→ [2, 3, 5, 3, 8, 10, 17] → [2, 3, 3, 5, 8, 10, 17]

FIN (ARREGLO ORDENADO)

NOTA

Tomás Achával Berreto

10

HOJA Nº 3

FECHA

Ejercicio 3:

```
FUN f(n:nat) ret t:nat
IF n≤1 then t:=1
else
    T:= f(n/2)
    T:= T + f(n/2)
    For i:=1 to n do
        For j:=1 to n do
            T:= t+1
        OD
    OD
FI
END FUN
```

FUNCIÓN DE RECURRENCIA

$$T(n) = \begin{cases} 1 & \text{si } n \leq 1 \\ \underbrace{2}_{a} \cdot T(n/\underbrace{2}_{b}) + n^{\overbrace{2}^{k}} & \text{EN OTRO CASO} \end{cases}$$

$a = 2$
$b = 2$
$k = 2$

Como $a < b^k$ $(2 < 2^2)$ ENTONCES EL PROCEDIMIENTO TIENE ORDEN n^k, ES DECIR, n^2.

ESTE RESULTADO SALE DE QUE EL ORDEN DE LA FUNCIÓN DE RECURRENCIA ESTÁ DADO POR:

$$\begin{cases} n^{(\log_b(a))} & \text{si } a > b^k \\ n^k \log(n) & \text{si } a = b^k \\ n^k & \text{si } a < b^k \end{cases}$$

HOJA N° 2

FECHA

...más Achaval Becerro

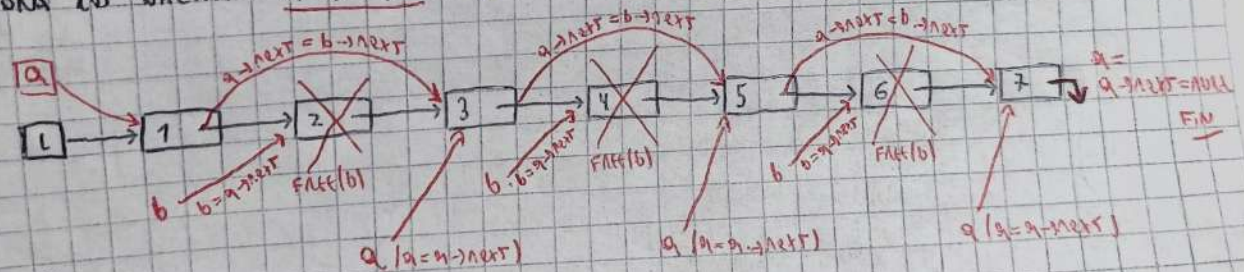
100%

Ejercicio 2:

a) El procedimiento P elimina los nodos en posiciones impares de la lista (si el primer elemento está en la posición 0).

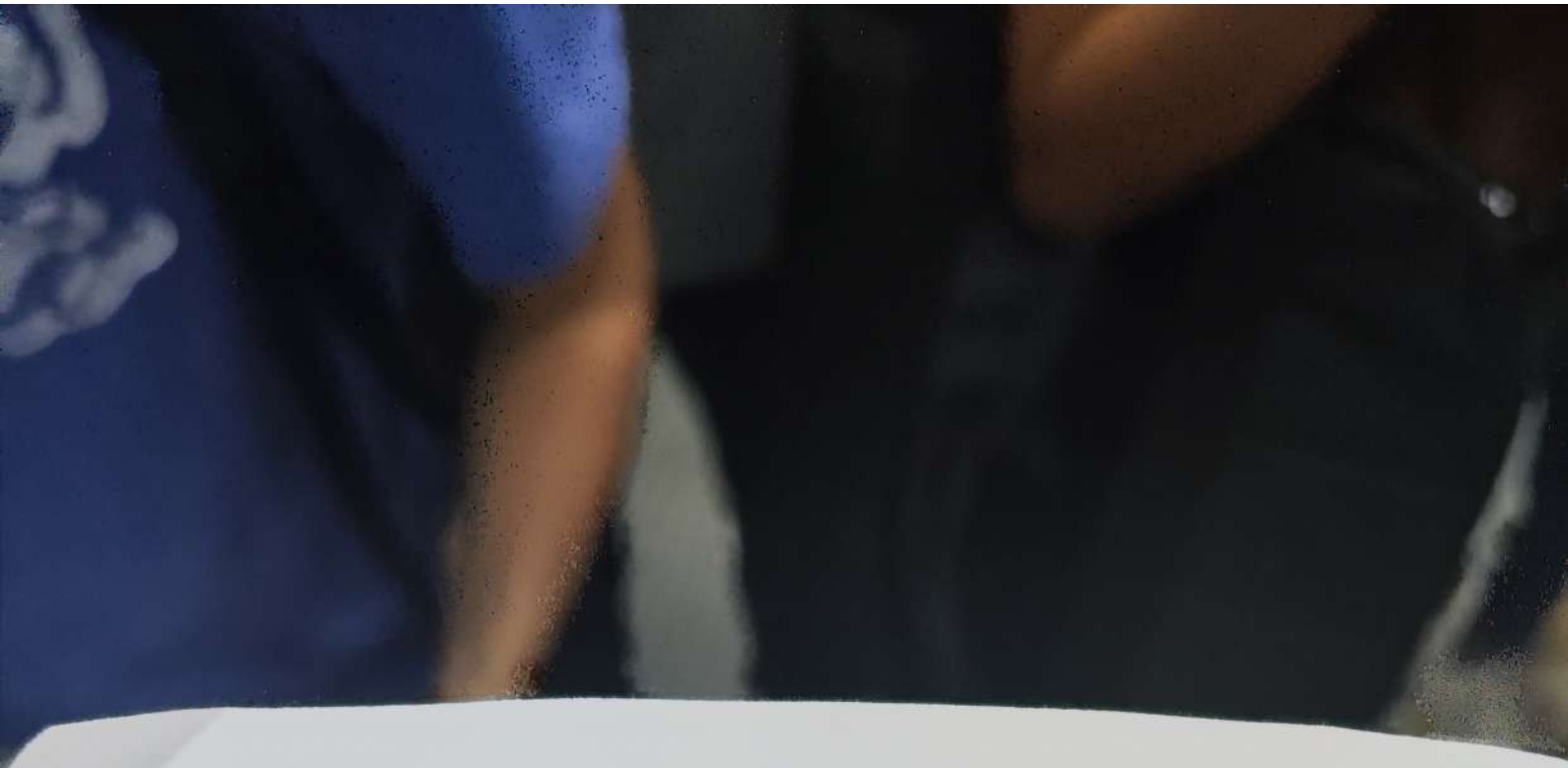
b) El orden del procedimiento P es n, donde n es la longitud de la lista. Esto es debido a que el ciclo while recorre una única vez a todos los elementos de la lista, y no hay recursión sobre P. Esto significa que la máxima cantidad de asignaciones no será más que un múltiplo de n, y por lo tanto el procedimiento es del orden de n. (Tampoco se utilizan funciones de orden ≥ n)

c) Si se llama a P con la lista 1, 2, 3, 4, 5, 6, 7, luego de su ejecución la lista tendrá los valores 1, 3, 5 y 7.



↗ Explicación gráfica

(escribí yo con rojo, no es que está mal)



b)

```
Proc P (in/out a: ARRAY [1..n] of Int, in e: Int, out K:nat)
   var lft:nat
   lft:=1
   For i:=1 to n do
       IF (a[i] < e) then
          SWAP(a, i, lft)
          lft:= lft+1
       FI
   OD
   K:= lft-1
END PROC
```

Tomás Achával Becerro

Ejercicio 4:

HOJA Nº 4

FECHA

a) SPEC Multiset of T where

CONSTRUCTORS

```
FUN empty_multiset () ret m: Multiset of T
{ Crea un multiconjunto vacío }

Proc addelem (in/out m: Multiset of T, in e:T)
{ Agrega el elemento e al multiconjunto m }
```

OPERATIONS

```
FUN is_empty_multiset (m:Multiset of T) ret b:bool
{ Devuelve true sii el multiconjunto m está vacío }

FUN elem_exists (m:Multiset of T, e:T) ret b:bool
{ Devuelve true si el elemento e está en m }

FUN elem_amount (m:Multiset of T, e:T) ret a:nat
{ Devuelve la cantidad "a" de veces que e está en m }

Proc delete_one (in/out m:Multiset of T, in e:T)
{ Elimina una ocurrencia de e en m }

Proc destroyM (in/out m: Multiset of T)
{ Destruye el multiconjunto m eliminando todos sus elementos }
```

b) IMPLEMENT Multiset of T where

```
TYPE Multiset of T = List of (Sets of T)

TYPE Sets = Tuple
  of T    element : T
          amount : nat
        END TUPLE

FUN empty_multiset() ret m: Multiset of T
   m := empty()
END FUN
```

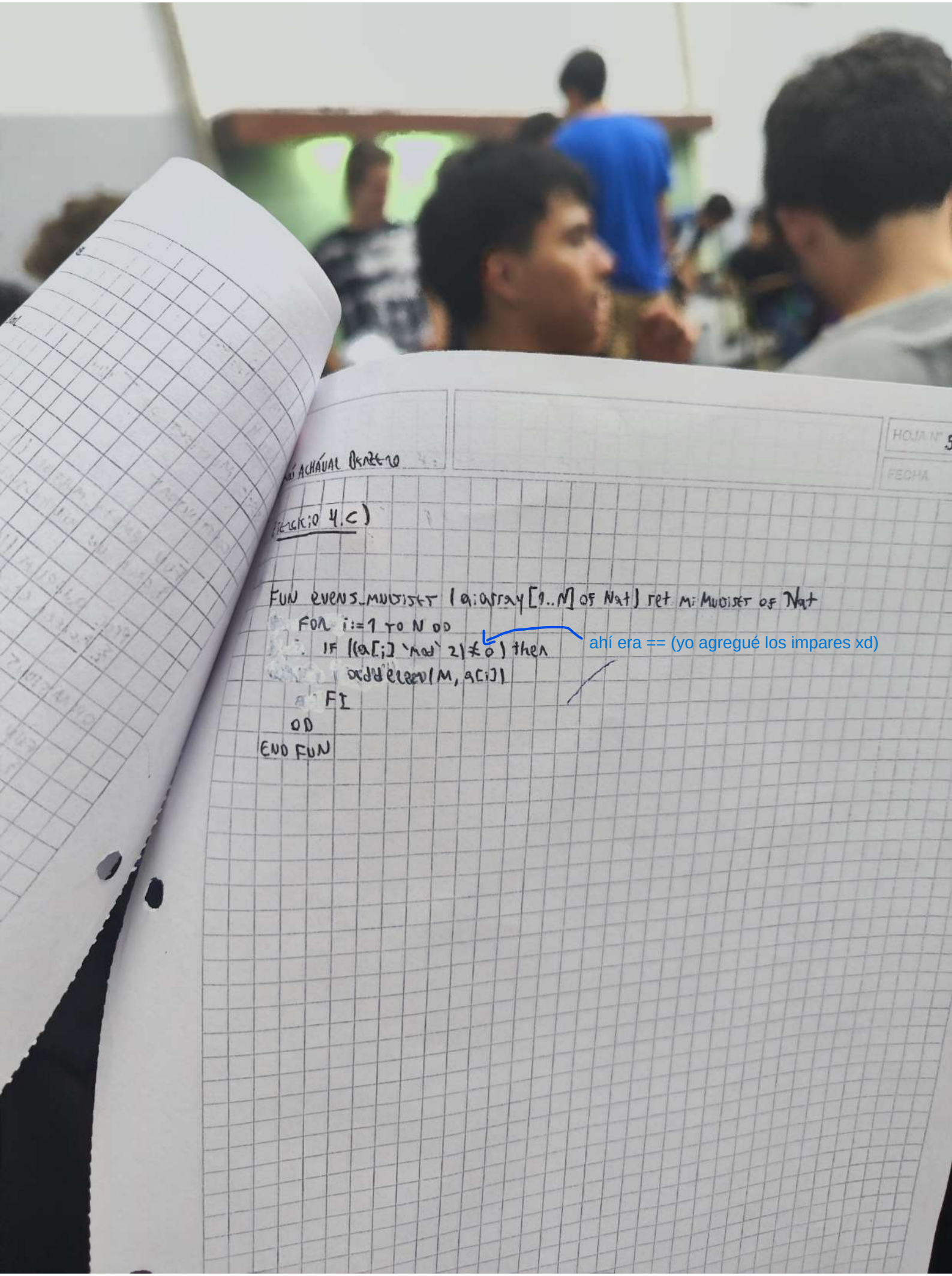
```
Proc addelem (in/out m:Multiset of T, in e:T)
   var k: Sets of T
   k.elem := e
   IF (elem_exists(m,e)) then
      while (head(m).element ≠ e) do
            addr (head(m))
            tail (m)
      od
      k.amount := head(m).amount + 1
      tail(m)
      addl(m, k)
   ELSE
      k.amount := 1
      addl(m, k)
   FI
END PROC
```

Además eliminarás los elementos de la lista.

Deberías corregir

```
FUN IS_EMPTY_MULTISET (M: MULTISET OF T) ret b: bool
   b:= IS_EMPTY (M)
END FUN

FUN elem_exists (M: MULTISET OF T, e:T) ret b:bool
   b:= FALSE
   var i:nat
   i:= 0
   while (not b ∧ i < length(M).) do
      b:= b ∨ (index(M,i).element = e)
      i:= i+1
   OD
END FUN

FUN ELEM_AMOUNT (M: MULTISET OF T, e:T) ret a:nat
   IF (not elem_exists(M,e)) then a:=0
   ELSE
      var i:nat
      i:=0
      while (index(M,i).element ≠ e) do
         i:= i+1
      OD
      a:= index(M,i).AMOUNT
   FI
END FUN

proc delete_one (in/out M: MULTISET OF T, in e:T)
   var K:nat
   K:= elem_AMOUNT (M, e)
   IF K=0 THEN SKIP
   ELSE
      while (head(M).element ≠ e) do
         addr(head(M))
         TAIL(M)
      OD
      IF K=1 then TAIL(M)
         ELSE
            var aux: SETT OF T
            aux.element := e
            aux.AMOUNT := head(M).AMOUNT - 1
            TAIL(M)
            ADDL(M, AUX)
      FI
   FI
END proc

Proc destroyM(M: MULTISET OF T)
   destroy(M)
END PROC
```

*NOTA: LAS FUNCIONES destroy, addl, addr, head, tail, etc son las funciones del TAD list of T.


HOJA N°
FECHA
Ejercicio 4.c)
FUN evens_multiset (a:array[1..N] of Nat) ret M: multiset of Nat
FOR i:=1 to N do
IF (a[i] mod 2 ≠ 0) then
addElem(M, a[i])
FI
OD
END FUN
ahí era == (yo agregué los impares xd)