4. (a) Especificá un TAD *tablero* para mantener el tanteador en contiendas deportivas entre dos equipos (equipo A y equipo B). Deberá tener un constructor para el comienzo del partido (tanteador inicial), un constructor para registrar un nuevo tanto del equipo A y uno para registrar un nuevo tanto del equipo B. El tablero sólo registra el estado actual del tanteador, por lo tanto el orden en que se fueron anotando los tantos es irrelevante.

   Además se requiere operaciones para comprobar si el tanteador está en cero, si el equipo A ha anotado algún tanto, si el equipo B ha anotado algún tanto, una que devuelva verdadero si y sólo si el equipo A va ganando, otra que devuelva verdadero si y sólo si el equipo B va ganando, y una que devuelva verdadero si y sólo si se registra un empate.

   Finalmente habrá una operación que permita anotarle un número n de tantos a un equipo y otra que permita "castigarlo" restándole un número n de tantos. En este último caso, si se le restan más tantos de los acumulados equivaldrá a no haber anotado ninguno desde el comienzo del partido.

```
spec Tablero where

constructors
    fun comienzo_del_partido() ret t: Tablero
    {- devuelve un tablero inicial donde ambos equipos no tienen puntos -}

    proc punto_para_A(in/out t: Tablero)
    {- agrega al tablero t un punto para el equipo A -}

    proc punto_para_B(in/out t: Tablero)
    {- agrega al tablero t un punto para el equipo B -}

destroy
    proc destruir_tablero(in/out t: Tablero)
    {- libera memoria en caso de ser necesario -}

operations
    fun tanteador_en_cero(t: Tablero) ret b: bool
    {- devuelve true en caso de que el tablero esté en 0 para ambos equipos -}

    fun anotó_equipoA(t: Tablero) ret b: bool
    {- devuelve true si y solo si el equipo A anotó por lo menos un punto -}

    fun anotó_equipoB(t: Tablero) ret b: bool
    {- devuelve true si y solo si el equipo B anotó por lo menos un punto -}

    fun gana_equipoA(t: Tablero) ret b: bool
    {- devuelve true si y solo si el equipo A va ganando -}

    fun gana_equipoB(t: Tablero) ret b: bool
    {- devuelve true si y solo si el equipo B va ganando -}

    fun empate(t: Tablero) ret b: bool
    {- devuelve true si y solo si A y B están empatando -}

    proc dar_puntos_al_equipoA(in/out t: Tablero, in n: nat)
    {- da n puntos al equipo A -}

    proc dar_puntos_al_equipoB(in/out t: Tablero, in n: nat)
    {- da n puntos al equipo B -}

    proc quitar_puntos_al_equipoA(in/out t: Tablero, in n: nat)
    {- quita n puntos al equipo A. Si A no tiene puntos, queda en 0 -}
```

```
    proc quitar_puntos_al_equipoB(in/out t: Tablero, in n: nat)
    {- quita n puntos al equipo B. Si B no tiene puntos, queda en 0 -}

end spec
```

(b) Implementá el TAD Tablero utilizando una tupla con dos contadores: uno que indique los tantos del equipo A, y otro que indique los tantos del equipo B.

```
implement Tablero where

type Tablero = tuple
                 puntos_A: Counter
                 puntos_B: Counter
               end tuple

constructors
    fun comienzo_del_partido() ret t: Tablero
        t.puntos_A := init()
        t.puntos_B := init()
    end fun

    proc punto_para_A(in/out t: Tablero)
        incr(t.puntos_A)
    end proc

    proc punto_para_B(in/out t: Tablero)
        incr(t.puntos_B)
    end proc

destroy
    proc destruir_tablero(in/out t: Tablero)
        destroy(t.puntos_A)
        destroy(t.puntos_B)
    end proc

operations
    fun tanteador_en_cero(t: Tablero) ret b: bool
        b := is_init(t.puntos_A) ∧ is_init(t.puntos_B)
    end fun

    fun anotó_equipoA(t: Tablero) ret b: bool
        b := not is_init(t.puntos_A)
    end fun

    fun anotó_equipoB(t: Tablero) ret b: bool
        b := not is_init(t.puntos_B)
    end fun

    fun gana_equipoA(t: Tablero) ret b: bool
        var ta,tb: Counter

        ta := t.puntos_A
        tb := t.puntos_B

        while not is_init(ta) ∧ not is_init(tb) do
            decr(ta)
```

```
        decr(tb)
    od

    if not is_init(ta) ∧ is_init(tb) then
        b := true
    else
        b := false
    fi
    destroy(ta)
    destroy(tb)
end fun

fun gana_equipoB(t: Tablero) ret b: bool
    var ta,tb: Counter

    ta := t.puntos_A
    tb := t.puntos_B

    while not is_init(ta) ∧ not is_init(tb) do
        decr(ta)
        decr(tb)
    od

    if is_init(ta) ∧ not is_init(tb) then
        b := true
    else
        b := false
    fi
    destroy(ta)
    destroy(tb)
end fun

fun empate(t: Tablero) ret b: bool
    var ta,tb: Counter

    ta := t.puntos_A
    tb := t.puntos_B

    while not is_init(ta) ∧ not is_init(tb) do
        decr(ta)
        decr(tb)
    od

    if is_init(ta) ∧ is_init(tb) then
        b := true
    else
        b := false
    fi
    destroy(ta)
    destroy(tb)
end fun

proc dar_puntos_al_equipoA(in/out t: Tablero, in n: nat)
    for puntos_extra := 1 to n do
        incr(t.puntos_A)
    od
end proc
```

```
    proc dar_puntos_al_equipoB(in/out t: Tablero, in n: nat)
        for puntos_extra := 1 to n do
            incr(t.puntos_B)
        od
    end proc

    proc quitar_puntos_al_equipoA(in/out t: Tablero, in n: nat)
        for puntos_quitados := 1 to n do
            if not is_init(t.puntos_A) then
                decr(t.puntos_A)
            else
                //skip
            fi
        od
    end proc

    proc quitar_puntos_al_equipoB(in/out t: Tablero, in n: nat)
        for puntos_quitados := 1 to n do
            if not is_init(t.puntos_B) then
                decr(t.puntos_B)
            else
                //skip
            fi
        od
    end proc

end implement
```

(c) Implementá el TAD Tablero utilizando una tupla con dos naturales: uno que indique los tantos del equipo A, y otro que indique los tantos del equipo B. ¿Hay alguna diferencia con la implementación del inciso anterior? ¿Alguna operación puede resolverse más eficientemente?

```
implement Tablero where

type Tablero = tuple
                 puntos_A: nat
                 puntos_B: nat
               end tuple

constructors
    fun comienzo_del_partido() ret t: Tablero
        t.puntos_A := 0
        t.puntos_B := 0
    end fun

    proc punto_para_A(in/out t: Tablero)
        t.puntos_A := t.puntos_A + 1
    end proc

    proc punto_para_B(in/out t: Tablero)
        t.puntos_B := t.puntos_B + 1
    end proc

destroy
    proc destruir_tablero(in/out t: Tablero)
        //skip
    end proc
```

```
operations
    fun tanteador_en_cero(t: Tablero) ret b: bool
        b := t.puntos_A = 0 ∧ t.puntos_B = 0
    end fun

    fun anotó_equipoA(t: Tablero) ret b: bool
        b := t.puntos_A > 0
    end fun

    fun anotó_equipoB(t: Tablero) ret b: bool
        b := t.puntos_B > 0
    end fun

    fun gana_equipoA(t: Tablero) ret b: bool
        b := t.puntos_A > t.puntos_B
    end fun

    fun gana_equipoB(t: Tablero) ret b: bool
        b := t.puntos_A < t.puntos_B
    end fun

    fun empate(t: Tablero) ret b: bool
        b := t.puntos_A = t.puntos_B
    end fun

    proc dar_puntos_al_equipoA(in/out t: Tablero, in n: nat)
        t.puntos_A := t.puntos_A + n
    end proc

    proc dar_puntos_al_equipoB(in/out t: Tablero, in n: nat)
        t.puntos_B := t.puntos_B + n
    end proc

    proc quitar_puntos_al_equipoA(in/out t: Tablero, in n: nat)
        if n ≤ t.puntos_A then
            t.puntos_A := t.puntos_A - n
        else
            t.puntos_A := 0
        fi
    end proc

    proc quitar_puntos_al_equipoB(in/out t: Tablero, in n: nat)
        if n ≤ t.puntos_B then
            t.puntos_B := t.puntos_B - n
        else
            t.puntos_B := 0
        fi
    end proc

end implement
```

Con esta implementación se vuelven más eficientes las operaciones de asignar y quitar *n* puntos a algún equipo pues no estoy limitado por el tipo *Counter*, el cual me llevaba a trabajar con ciclos para incrementar o decrementar de a 1 los puntos totales por cada iteración hasta *n*. De esta forma puedo restar o sumar directamente los puntos sin ir de a uno, y si en el caso de la quita de puntos el marcador es negativo, simplemente se lo cambia a 0. Sucede algo parecido con las operaciones de indicar si va ganando el equipo A, el B o empate, puesto que comparo directamente los valores en vez de decrementarlos y luego comparar.