Ej 1) hecho en TADS - PARCIALES

Ej 2)

2. Utilizando la siguiente implementación del tipo *list* con punteros, se quiere implementar el procedimiento *lshift*. Este procedimiento toma una lista y efectua una rotación circular a izquierda de los elementos. Por ejemplo, dada la lista de números [0,0,1,1,0,1], al aplicar el procedimiento *lshift*, el resultado será [1,0,0,1,1,0].

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

Implementá el procedimiento *lshift* dando correctamente su encabezado y definición.

La idea es que todos se muevan adelante salvo el ultimo de la lista que esc va al comienzo, tambien hay que modificar en next de last y ante ultimo

list → node 1 → Node 2 → .... → node n

node = | elem | next → |

Spec

proc lshift (in/out l : List)

}- Efectua una rotacion circular a la izq en la lista -{

```
// La idea es que el ultimo elemento de la lista pase a ser el primero, y el resto de elementos se desplacen hacia la derecha
// Ejemplo: si la lista es [1,2,3,4,5], luego de la rotacion debe quedar [5,1,2,3,4]
// Para esto, primero recorro la lista hasta llegar al ultimo elemento, y luego hago que el ultimo elemento apunte al primer elemento
// y el penultimo elemento apunte a NULL. Luego, el primer elemento pasa a ser el segundo, el segundo pasa a ser el tercero, y asi sucesivamente
proc lshift(in/out l : List)
    ultimo : pointer to Node
    penultimo : pointer to Node
    alloc(last)
    alloc(aux)
    penultimo := l
    ultimo->next := l
    // Recorro la lista hasta llegar al penultimo elemento
    do (penultimo->next->next != NULL){
        penultimo := penultimo->next
    }
    od
    penultimo->next := NULL

end proc
```

3. Para cada uno de los siguientes algoritmos determinar por separado cada uno de los siguientes incisos.

(a) ¿Qué hace? ¿Cuáles son las precondiciones necesarias para que haga eso?

(b) ¿Cómo lo hace?

(c) El orden del algoritmo, analizando los distintos casos posibles.

(d) Proponer nombres más adecuados para los identificadores (de variables, funciones y procedimientos).

```
proc p (in/out a: array[1..N] of T)
    var i: nat
    var x: nat
    i:= 2
    do i <= N
        x:= f(a,i)
        swap(a,i,x)
        i:= i+2
    od
end proc
```

```
fun f (a: array[1..N] of T, i: nat) ret x: nat
    var j: nat
    j:= i+2
    x:= i
    do j <= N
        if a[j] < a[x] then x:= j fi
        j:= j+2
    od
end fun
```

a) La fun f retorna la posición del elemento más chico del array, en las posiciones pares del arreglo,

- El proc p, ordena el arreglo swapeando los elementos en las posiciones dadas (todas pares)

La precondición es que el arreglo tenga al menos 4 elementos porque se inicia la var i en 2 y luego f itera desde i+2 = 4

b) Lo explicado, p ordena usando swap en los pos pares y f itera para encontrar la pos del elem. más chico en a

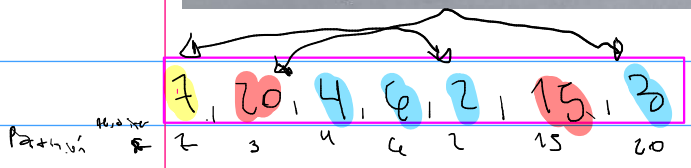
c) El orden es $\Theta(n^2)$, vemos que tenemos un ciclo anidado en p, que itera desde i=2 hasta n, osea n-2 veces, y luego dentro de ese ciclo ocurre otro ciclo por f, que itera desde 4 hasta n, osea n-4 $\Rightarrow$ tenemos

$n-2 \cdot (n-4) = n^2 - 4n - 2n + 8 = n^2 - 6n + 8$ y nos quedamos solo con los términos significativos $= n^2 - n$ en $\Theta(n^2)$

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

$$[7, 20, 4, 6, 2, 15, 3]$$

(b) Da las secuencias de llamadas al procedimiento *quick_sort_rec* indicando correctamente sus argumentos.



quick_sort_rec

q_sr(A, 0, n)

piv = 5

2, 3, 4, 6, 7, 15, 20

q_sr(2, 1, 4)

2, 3, 4, 4, 7, 15, 20

q_sr(2, 6, 7)