

Práctico 2.1

Funciones generales:

```
fun (!!!)(a : array[1..m] of A, j, k : nat) out res : array[1..k-j] of A
  for i = j to k do
    res[i-j] = a[i]
  od
end fun
```


Ejercicios:

1)

1. Escribir un algoritmo que dada una matriz a: **array[1..n,1..m] of int** calcule el elemento mínimo. Escribir otro algoritmo que devuelva un arreglo **array[1..n]** con el mínimo de cada fila de la matriz a.

```
fun mínimo2(in a : array[1..n, 1..m] of int) out res : int
  res = a[1, 1]
  for j = 1 to n do
    for k = 1 to m do
      res = min(res, a[j, k])
    od
  od
end fun
```

```
fun mínimoParaCada(in a : array[1..n, 1..m] of int) out res : array[1..n] of int
  for j = 1 to n do
    res[j] = a[j, 1]
    for k = 1 to m do
      res[j] = min(a[j, k], res[j])
    od
  od
end fun
```

2) 

2. Dados los tipos enumerados

```
type mes = enumerate
    enero
    febrero
    ...
    diciembre
end enumerate
```

```
type clima = enumerate
    Temp
    TempMax
    TempMin
    Pres
    Hum
    Prec
end enumerate
```

El arreglo `med:array[1980..2016,enero..diciembre,1..28,Temp..Prec]` **of nat** es un arreglo multidimensional que contiene todas las mediciones estadísticas del clima para la ciudad de Córdoba desde el 1/1/1980 hasta el 28/12/2016. Por ejemplo, `med[2014,febrero,3,Pres]` indica la presión atmosférica que se registró el día 3 de febrero de 2014. Todas las mediciones están expresadas con números enteros. Por simplicidad asumiremos que todos los meses tienen 28 días.

- (a) Dar un algoritmo que obtenga la menor temperatura mínima (TempMin) histórica registrada en la ciudad de Córdoba según los datos del arreglo.
- (b) Dar un algoritmo que devuelva un arreglo que registre para cada año entre 1980 y 2016 la mayor temperatura máxima (TempMax) registrada durante ese año.
- (c) Dar un algoritmo que devuelva un arreglo que registre para cada año entre 1980 y 2016 el mes de ese año en que se registró la mayor cantidad mensual de precipitaciones (Prec).
- (d) Dar un algoritmo que utilice el arreglo devuelto en el inciso anterior (además de med) para obtener el año en que ese máximo mensual de precipitaciones fue mínimo (comparado con los de otros años).
- (e) Dar un algoritmo que obtenga el mismo resultado sin utilizar el del inciso (c).

2a)

```
fun mínimoHistórico(med : array[1980..2016, enero..diciembre, 1..28, Temp..Prec])
out res : int
    mínimo3
end fun
```

4)

4. Dados dos punteros p, q : **pointer to int**

- (a) escribí un algoritmo que intercambie los valores referidos sin modificar los valores de p y q .
- (b) escribí otro algoritmo que intercambie los valores de los punteros.

Sea un tercer puntero r : **pointer to int** que inicialmente es igual a p , y asumiendo que inicialmente $*p = 5$ y $*q = -4$ ¿cuáles serían los valores de $*p$, $*q$ y $*r$ luego de ejecutar el algoritmo en cada uno de los dos casos?

4a)

```
proc intercambiar(in/out p, q : pointer to int)
  var temp : pointer to int
  temp = q
  q = p
  p = temp
end proc
```

4b)

```
proc intercambiarDireccionados(in/out p, q : pointer to int)
  var temp : int
  temp = *q
  *q = *p
  *p = temp
end proc
```

5)

5. Dados dos arreglos $a, b : \text{array}[1..n] \text{ of nat}$ se dice que a es “lexicográficamente menor” que b sii existe $k \in \{1 \dots n\}$ tal que $a[k] < b[k]$, y para todo $i \in \{1 \dots k - 1\}$ se cumple $a[i] = b[i]$. En otras palabras, si en la primera posición en que a y b difieren, el valor de a es menor que el de b . También se dice que a es “lexicográficamente menor o igual” a b sii a es lexicográficamente menor que b o a es igual a b .

- (a) Escribir un algoritmo `lex_less` que recibe ambos arreglos y determina si a es lexicográficamente menor que b .
- (b) Escribir un algoritmo `lex_less_or_equal` que recibe ambos arreglos y determina si a es lexicográficamente menor o igual a b .

- (c) Dado el tipo enumerado

```
type ord = enumerate
  igual
  menor
  mayor
end enumerate
```

Escribir un algoritmo `lex_compare` que recibe ambos arreglos y devuelve valores en el tipo `ord`.
¿Cuál es el interés de escribir este algoritmo?

5a)

Versión recursiva:

```

fun lex_less(a, b : array[1..n] of nat) out res : bool
    res = n < 1 ∨ a[1] < b[1] ∨ (a[1] = b[1] ∧ lex_less (a !!! (2, n)))
end fun

```

Versión no recursiva:

```

fun lex_less(a, b : array[1..n] of nat) out res : bool
    var j : nat
    j = 1
    while a[j] = b[j] ∧ j < n do
        j = j+1
    od
    res = a[j] < b[j]
end fun

```

5b)

```

fun lex_less_or_equal(a, b : array[1..n] of nat) out res : bool
    var j : nat
    j = 1
    while a[j] = b[j] ∧ j ≤ n do
        j = j+1
    od
    res = j > n ∨ a[j] < b[j]
end fun

```

5c)

```

fun order(a, b : nat) out res : ord
    if a < b →
        res = menor
    else if a = b →
        res = igual
    else
        res = mayor
    fi
fi
end fun

```

```

fun lex_compare(a, b : array[1..n] of nat) out res : ord
    var j : nat
    j = 1
    while a[j] = b[j] ∧ j < n do
        j = j+1
    od
    res = order(a[j], b[j])
end fun

```

6)

6. Escribir un algoritmo que dadas dos matrices a, b: **array[1..n,1..m] of nat** devuelva su suma.

```
fun suma(a, b : array[1..n, 1..m]) out res : array[1..n, 1..m]
  for j = 1 to n do
    for k = 1 to m do
      res[j, k] = a[j, k] + b[j, k]
    end for
  end for
end fun
```

7)

7. Escribir un algoritmo que dadas dos matrices a: **array[1..n,1..m] of nat** y b: **array[1..m,1..p] of nat** devuelva su producto.

```
fun producto(a : array[1..n, 1..m], b : array[1..m, 1..p])out res : array[1..n, 1..p]
  for j = 1 to n do
    for k = 1 to p do
      res[j, k] = 0
      for i = 0 to m do
        res[j, k] = res[j, k] + a[j, i] * b[i, k]
      end for
    end for
  end for
end fun
```