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 minArray2D(a: array[1..n, 1..m] of int) ret r: int
    r := a[1][1]

    for i := 1 to n do
        for j := 1 to m do
            if (a[i][j] < r) then
                r := a[i][j]
            fi
        od
    od

end fun

fun minsPerRowArray2D(a: array[1..n, 1..m] of int) ret r: array[1..n]
    var maxValuePos: nat

    for i := 1 to n do
        r[i] := a[i][1]

        for j := 2 to m do
            if (a[i][j] < r[i]) then
                r[i] := a[i][j]
            fi
        od

    od

end proc
```

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 $med : array[1980..2016, enero..diciembre, 1..28, Temp..Prec]\ of\ int$ es un arreglo multidimensional que contiene todas las mediciones estadísticas del clima para la ciudad de Córdoba desde el $1/1/1980$ hasta el $28/12/2016$.

- Ej: $med[2014, febrero, 3, Pres]$ 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.

```
FST_YEAR := 1980
LST_YEAR := 2016
DAYS := 28

fun minTempRegistered(a: array[1980..2016, enero..diciembre, 1..28, Temp..Prec]
of int) ret r: int
    r := a[FST_YEAR][enero][1][TempMin]

    for year := FST_YEAR to LST_YEAR do
        for month := enero to diciembre do
            for day := 1 to DAYS do
                if (a[year][month][day][TempMin] < r) then
                    r := a[year][month][day][TempMin]
                fi
            od
        od
    od

end fun
```

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.

```
FST_YEAR := 1980
LST_YEAR := 2016
DAYS := 28

fun maxTempRegisteredPerYear(a: array[1980..2016, enero..diciembre, 1..28,
Temp..Prec] of int) ret r: array[1980..2016] of int

    for year := FST_YEAR to LST_YEAR do
        r[year] := a[year][enero][1][TempMax]

        for month := enero to diciembre do
```

```
            for day := 1 to DAYS do
                if (a[year][month][day][TempMax] > r[year]) then
                    r[year] := a[year][month][day][TempMax]
                fi
            od
        od
    od

end fun
```

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)

```
FST_YEAR := 1980
LST_YEAR := 2016
DAYS := 28

fun maxTempRegisteredPerYear(
    a: array[1980..2016, enero..diciembre, 1..28, Temp..Prec] of int
) ret r: array[1980..2016] of mes
    var qPrec, qPrecMax : nat

    for year := FST_YEAR to LST_YEAR do
        r[year] := enero
        qPrecMax := 0

        for month := enero to diciembre do
            qPrec := 0

            for day := 1 to DAYS do
                qPrec := qPrec + a[year, month, day, Prec]
            od

            if (qPrec > qPrecMax) then
                r[year] := month
                qPrecMAx := qPrec
            fi
        od
    od

end fun
```

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).

```
FST_YEAR := 1980
LST_YEAR := 2016
DAYS := 28
```

```
fun yearOfLowerMaxPrecRegistered(
    a: array[1980..2016, enero..diciembre, 1..28, Temp..Prec] of int
) ret r: int
    var qPrec, qPrecMin : nat

    b := maxTempRegisteredPerYear(a)

    qPrecMin := 0
    month := b[FST_YEAR]
    r := FST_YEAR

    for day := 1 to DAYS do
        qPrecMin := qPrecMin + a[FST_YEAR, month, day, Prec]
    od

    for year := FST_YEAR + 1 to LST_YEAR do
        month := b[year]

        qPrec := 0

        for day := 1 to DAYS do
            qPrec := qPrec + a[year, month, day, Prec]
        od

        if (qPrec < qPrecMin) then
            r := year
            qPrecMin := qPrec
        fi
    od

end fun
```

e

Dar un algoritmo que obtenga el mismo resultado sin utilizar el del inciso (c)

```
FST_YEAR := 1980
LST_YEAR := 2016
DAYS := 28

proc aux(
    in a: array[1980..2016, enero..diciembre, 1..28, Temp..Prec] of int,
    in year: int,
    in/out qPrecMonthlyMax: nat
) ret r :nat
    r := qPrecMonthlyMax

    for month := enero to diciembre do
        qPrecMonthly := 0

        for day := 1 to DAYS do
            qPrecMonthly := qPrecMonthly + a[year, month, day, Prec]
        od

        if (qPrecMonthly > r) then
```

```
            r := qPrecMonthly
        fi
    od
end proc

fun yearOfLowerMaxPrecRegisteredQuick(
    a: array[1980..2016, enero..diciembre, 1..28, Temp..Prec] of int
) ret r: int
    var qPrecMonthly, qPrecMonthlyMax, qPrecLowerMax : nat

    qPrecLowerMax := 0
    r := FST_YEAR

    aux(a, FST_YEAR, qPrecLowerMax)

    for year := FST_YEAR to LST_YEAR do
        qPrecMonthlyMax := 0

        aux(a, year, qPrecMonthlyMax)

        if (qPrecLowerMax > qPrecMonthlyMax) then
            qPrecLowerMax := qPrecMonthlyMax
            r := year
        fi
    od

end fun
```

3

Dado el tipo

```
type person =   tuple
                    name: string
                    age: nat
                    weight: nat
                end tuple
```

a

Escribí un algoritmo que calcule la edad y peso promedio de un arreglo $a : array[1..n]\ of\ person$.

```
fun averageAgeAndWeight(
    a : array[1..n] of person
) ret r: person
    r.name := "Averages"

    for i := 1 to n do
        r.age := r.age + a[i].age
        r.weight := r.weight + a[i].weight
    od

    r.age := r.age / n
```

```
    r.weight := r.weight / n

end fun
```

b

Escribí un algoritmo que ordene alfabéticamente dicho arreglo.

```
fun goesBeforeAux(
    x: array[1..n] of char,
    y: array[1..m] of char
) ret r: bool
    var i : nat

    r := true
    i := 1
    while (i <= n && r) do
        r := (i <= m) && (x[i] <= y[i])

        i := i + 1
    od

end fun

fun goesBefore(
    a: array[1..n] of person,
    i: nat,
    j: nat
) ret r: bool
    r := goesBeforeAux(a[i].name, a[j].name)

end fun

proc partition(in/out a: array[1..n] of person, in lft, rgt: nat, out ppiv: nat)
    var i,j: nat

    ppiv := lft
    i := lft + 1
    j := rgt

    while (i <= j) do
        if (goesBefore(a, i, ppiv)) then
            i := i + 1
        else if (goesBefore(a, ppiv, j)) then
            j := j - 1
        else if (goesBefore(a, ppiv, i) && goesBefore(a, j, ppiv)) then
            swap(a, i, j)
            i := i + 1
            j := j - 1
        fi
    od

    swap(a, ppiv, j)
    ppiv:= j
```

```
end proc

proc quick_sort_rec(in/out a: array[1..n] of person, in lft,rgt: nat)
    var ppiv: nat
    if rgt > lft then
        partition(a, lft, rgt, ppiv)

        quick_sort_rec(a, lft, ppiv - 1)
        quick_sort_rec(a, ppiv + 1, rgt)
    fi
end proc

proc sortABC(
    a : array[1..n] of person
)
    quick_sort_rec(a, 1, n)
end proc
```

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.

```
proc swap1(p, q : pointer to int)
    var tempP : pointer to int
    var tempI : int

    tempP := p
    tempI := *p

    *tempP := *q
    tempP = q
    *tempP := tempI

end proc
```

b

```
proc swap2(p, q : pointer to int)
    var tempP : pointer to int

    tempP := p
    p := q
    q := tempP

end proc
```

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?

```
.................

r := p
*p := 5
*q := -4
// *r := 5
// *p := 5
// *q := -4

swap1(p, q)
// *r := -4
// *p := -4
// *q := 5

//  figure out shit before doesnt happen
swap2(p, q)
// *r := 5
// *p := -4
// *q := 5
```

5

Dados dos arreglos $a, b : array[1..n]\ of\ nat$ se dice que a es ***"lexicográficamente menor"*** que b si existe $k \in 1...n$ tal que $a[k] < b[k]$, y para todo $i \in 1...k-1$ 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 si 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.

```
fun lex_less(
    a,b : array[1..n] of nat
) r: bool
    var i: nat

    i := 1
    while (i < n && a[i] = b[i]) do
        i := i + 1
    od

    r := a[i] < b[i]

end fun
```

b

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

```
fun lex_less_or_equal(
```

```
    a,b : array[1..n] of nat
) r: bool
    var i: nat

    i := 1
    while (i < n && a[i] = b[i]) do
        i := i + 1
    od

    if (i = n)

    r := a[i] < b[i] || ((i = n) && a[i] = b[i])

end fun
```

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?

```
fun lex_compare(
    a,b : array[1..n] of nat
) r: ord

    if (lex_less_or_equal(a, b)) then
        if (a[n] = b[n]) then
            r := igual
        else
            r := menor
        fi
    else
        r := mayor
    fi

end fun
```

- Vuelve más legible el código, lo hace más performante y lo acerca más al programador / cliente.

6

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

```
fun sumArray2D(
    a,b : array[1..n, 1..m] of nat
) r: nat
    r := 0

    for i := 1 to n do
        for j := 1 to m do
            r := r + a[i][j] + b[i][j]
        od
    od

end fun
```

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 sumArray2D(
    a : array[1..n, 1..m] of nat
    b : array[1..m, 1..p] of nat
) r: array[1..n, 1..p] of nat

    for i := 1 to n do
        for k := 1 to p do
            r[i][k] := 0

            for j := 1 to m do
                r[i][k] := r[i][k] + a[i][j] * b[j][k]
            od
        od
    od

end fun
```