Práctico 9 - Arreglos y Subrangos

Programación 1
InCo - Facultad de Ingeniería, Udelar

1. Dadas las siguientes declaraciones:

```
const
   N = ... ; {valor mayor estricto a 1}
var
   arreglo: array [1..N] of integer;
   i, menor, indice: integer;
```

Indique cuál o cuáles de los siguientes fragmentos de programa encontrarán el valor más pequeño de este arreglo y almacenarán en la variable `indice` el **menor índice** de la celda donde está guardado dicho valor.

I)
```
indice := 1;
for i := 2 to N do
    if arreglo[i] < indice then
        indice := i
```

II)
```
indice := 1;
for i := 2 to N do
    if arreglo[i] < arreglo[indice] then
        indice := i
```

III)
```
menor := arreglo[1];
indice := 1;
for i := 2 to N do
    if arreglo[i] <= menor then
    begin
        menor := arreglo[i];
        indice := i;
    end
```

IV)
```
indice := N;
for i := N-1 downto 1 do
    if arreglo[i] <= arreglo[indice] then
        indice := i
```

> Solución: II y IV

2. Dadas las siguientes declaraciones:

```
const
   N = ...; {valor mayor estricto a 1}
var
   cadena: array [1..N] of char;
   k: integer;
   temp: char;
```

Indique cuál o cuáles de los siguientes fragmentos de programa invertirán el arreglo cadena.

I)
```
for k := 1 to N do
begin
    temp := cadena[k];
    cadena[k] := cadena[(N+1) - k];
    cadena[(N+1) - k] := temp
end
```

II)
```
for k := 1 to (N DIV 2) do
begin
    temp := cadena[k];
    cadena[k] := cadena[(N+1) - k];
    cadena[(N+1) - k] := temp
end
```

III)
```
for k := 1 to (N DIV 2) do
begin
    temp := cadena[k];
    cadena[k] := cadena[(N+1) - k];
    cadena[k] := temp
end
```

IV)
```
for k := 1 to (N DIV 2) do
begin
    temp := cadena[k];
    cadena[k] := cadena[(N DIV 2) - k];
    cadena [(N DIV 2) - k] := temp
end
```

Solución: II

3. Dadas las siguientes declaraciones:

```
const
   N = ...; {valor mayor estricto a 1}
type
   CadenaN = array [1..N] of integer;
```

(a) Escriba la función llamada *cantMayores* que, dados un arreglo de enteros y un entero `num`, devuelve la cantidad de valores almacenados en el arreglo que son mayores que `num`.

`function cantMayores(cadn:CadenaN; num:integer) : integer;`

```
function cantMayores(cadn:CadenaN; num:integer) : integer;
var cant,i : integer;
begin
    cant := 0;
    for i := 1 to N do
        if cadn[i] > num then
            cant := cant + 1;
    cantMayores := cant
end;
```

(b) Escriba la función llamada *ordenado* que, dado un arreglo de enteros, devuelve `true` si el arreglo está ordenado en forma ascendente y `false` en caso contrario.

`function ordenado(cadn:CadenaN) : boolean;`

```
function ordenado(cadn:CadenaN) : boolean;
var i : integer;
begin
    i := 1;
    while (i < N) and (cadn[i] <= cadn[i+1]) do
        i := i + 1;
    ordenado := i = N
end;
```

(c) Escriba el procedimiento llamado *maxValorPos* que, dado un arreglo de enteros, devuelve el valor más grande y el primer índice en que éste ocurre.

```
procedure maxValorPos(cadn:CadenaN; VAR valor, pos:integer);
```

```
procedure maxValorPos(cadn:CadenaN; VAR valor, pos:integer);
var i : integer;
begin
    pos := 1;
    for i := 2 to N do
        if cadn[i] > cadn[pos]then
            pos := i;

    valor := cadn[pos]
end;
```

4. (a) Defina un tipo `TipoMatriz` (arreglo bidimensional) de enteros de diez filas y diez columnas.

   Ver solución en la última parte.

   (b) Escriba un procedimiento que cargue una variable de tipo `TipoMatriz` con valores leídos desde la entrada.

   Ver solución en la última parte.

   (c) Escriba un procedimiento que dada una matriz de tipo `TipoMatriz` y dos variables enteras `m` y `n`, intercambie las filas `m` y `n` de la matriz. Es decir, todos los valores de la fila `m` deben quedar en la fila `n` y viceversa.

   Ver solución en la última parte.

   (d) Escriba un programa principal que lea de la entrada una matriz (usando la parte b) y dos valores `m` y `n` y despliegue el resultado de intercambiar las filas `m` y `n` (usando la parte c). En caso de que `m` y/o `n` no correspondan a números de fila válidos, se debe emitir un mensaje de error.

```
program pr9ej4;
const K = 10;
type TipoMatriz = array [1..K, 1..K] of integer;  {Parte a}
     TipoFila = array [1..K] of integer;
var mat : TipoMatriz;
    m,n : integer;

{Parte b}
procedure carga (var matriz: TipoMatriz);
var i,j : integer;
begin
    writeLn('Ingrese una matriz de ', K:2,' x ',K:2,' valores enteros: ');
    {cargo la matriz por fila}
    for i := 1 to K do
    begin
        for j := 1 to K do
            read(matriz[i,j]);
        readln
    end
end;

{Parte c}
procedure intercambioFilas (var matriz: TipoMatriz; m,n:integer);
var aux : TipoFila;
begin
    aux := matriz[m];
    matriz[m] := matriz[n];
    matriz[n] := aux
end;

procedure imprimirMatriz (matriz: TipoMatriz);
var f,c : integer;
begin
     for f := 1 to K do
    begin
        for c := 1 to K do
            write(matriz[f,c]);
        writeLn;
    end
end;

begin
    carga(mat);

    write('Ingrese dos filas a intercambiar: ');
    readln(m,n);

    if (m > N) or (m < 1) or (n > N) or (n < 1) then
        writeLn('Error: El o los valores ingresados no corresponden a
                                                un numero de fila valido.')
    else
    begin
        intercambioFilas(mat,m,n);
        imprimirMatriz(mat)
    end
end.
```

5. Dadas las siguientes declaraciones para representar cadenas de caracteres de largo M y N :

```
const M = ...; {valor mayor estricto a 1}
```

```
const N = ...; {valor mayor estricto a M}
type
    CadenaM = array [1..M] of char;
    CadenaN = array [1..N] of char;
```

(a) Escriba una función llamada *indiceSubCadena* que dadas dos cadenas (de largo `M` y `N` respectivamente) determine si la primera cadena ocurre al menos una vez como parte de la segunda cadena. En caso afirmativo, se devuelve la posición (índice) en que comienza la primera coincidencia. En caso contrario, se devuelve 0.

Ver solución en la última parte.

(b) Escriba dos procedimientos llamados *leerCadenaM* y *leerCadenaN* que carguen una cadena de largo `M` y `N` respectivamente con caracteres leídos desde la entrada.

Ver solución en la última parte.

(c) Escriba un programa principal que lea dos cadenas de largo `M` y `N` y despliegue si la primer cadena ocurre como parte de la segunda. Si la cadena ocurre, se debe mostrar la posición de la primera coincidencia.

Ejemplos:

Sea M = 3, N = 10:
 	
`Ingrese los 3 caracteres de la cadenaM: tor`
`Ingrese los 10 caracteres de la cadenaN: totoratora`
`La cadena se encuentra a partir de la posición 3`

Sea M = 5, N = 6:
 	
`Ingrese los 5 caracteres de la cadenaM: tosco`
`Ingrese los 6 caracteres de la cadenaN: totora`
`La cadena no se encuentra`

```
program pr9ej5;
const M = ...; {valor mayor estricto a 1}
const N = ...; {valor mayor estricto a M}
type
    CadenaM = array [1..M] of char;
    CadenaN = array [1..N] of char;
var cm:CadenaM;
    cn:CadenaN;
    indice:integer;


function indiceSubCadena(cadn:CadenaN; cadm:CadenaM):integer;
var indn: integer;

    function  esSubCadenaEn(cadn:CadenaN; cadm:CadenaM; i:integer):boolean;
    var idx : integer;
    begin
        idx := 1;
        while (idx <= M) and (cadn[i + idx - 1] = cadm[idx]) do
            idx := idx + 1;

        esSubCadenaEn := (idx = M +1)
    end;

begin
    indn := 1;
    {Mientras no encuentra la subcadena y hay lugar para encontrarla}
    while ((indn + M - 1) <= N) and (not esSubCadenaEn(cadn,cadm,indn)) do
        indn := indn + 1;

    if (indn + M - 1) <= N then
        indiceSubCadena := indn
    else
        indiceSubCadena := 0

end;

procedure leerCadenaM(var cadm:CadenaM);
var i : integer;
begin
    write('Ingrese los ',M:1,' caracteres de la cadenaM: ');
    for i:=1 to M do
        read(cadm[i]);
    readln
end;

procedure leerCadenaN(var cadn:CadenaN);
var i : integer;
begin
    write('Ingrese los ',N:1,' caracteres de la cadenaN: ');
    for i:=1 to N do
        read(cadn[i]);
    readln
end;

begin
    leerCadenaM(cm);
    leerCadenaN(cn);
    indice := indiceSubCadena(cn,cm);
```

```
    if indice = 0 then
        writeln('La cadena no se encuentra.')
    else
        writeln('La cadena se encuentra a partir de la posición ', indice)
end.
```

6. La transpuesta de una matriz cuadrada `a` es una matriz `b` del mismo tipo cuyas celdas satisfacen la relación `b[i,j] = a[j,i]` para todos los valores posibles de `i` y `j`. Considere las siguientes declaraciones:

```
const N = ...; {valor mayor estricto a 1}
type
    Matriz = array [1..N, 1..N] of integer;
```

   (a) Escriba el procedimiento *transpuestaMatrizAB* que, dada una matriz cuadrada `a` calcule su matriz transpuesta `b`.

   ```
   procedure transpuestaMatrizAB(a:Matriz; VAR b:Matriz);
   ```

```
procedure transpuestaMatrizAB(a:Matriz; VAR b:Matriz);
var i,j:integer;
begin
    {Para cada fila}
    for i:=1 to N do
        {Para cada columna}
        for j:=1 to N do
            b[i,j] := a[j,i]
end;
```

   (b) Escriba un procedimiento que calcule nuevamente la transpuesta, pero ahora escribiendo el resultado sobre la misma matriz `a` (sin usar una segunda matriz).

   ```
   procedure transpuestaMatrizA(VAR a:Matriz);
   ```

```
procedure transpuestaMatrizA(VAR a:Matriz);
var i,j,aux:integer;
begin
    {Para cada fila}
    for i:=1 to N do
        {Para cada elemento del triangulo superir de la matriz }
        for j:= i + 1 to N do
        begin
            aux :=  a[i,j];
            a[i,j] := a[j,i];
            a[j,i] := aux
        end
end;
```

7. Dadas las siguientes declaraciones:

```
const
    M = ...; {valor mayor estricto a 1}
    N = ...; {valor mayor estricto a 1}
    P = ...; {valor mayor estricto a 1}
type
    RangoM = 1..M;
    RangoN = 1..N;
    RangoP = 1..P;
    MatrizMN = array [RangoM, RangoN] of integer;
    MatrizNP = array [RangoN, RangoP] of integer;
    MatrizMP = array [RangoM, RangoP] of integer;
```

Escriba el procedimiento *productoMatrices* que dadas una matriz `a` (de dimensiones MxN) y una matriz `b` (de dimensiones NxP), calcule su producto almacenando el resultado en una matriz `c` (de dimensiones MxP).

```
procedure productoMatrices(a:MatrizMN; b:MatrizNP; VAR c:MatrizMP);
```

```
procedure productoMatrices(a:MatrizMN; b:MatrizNP; VAR c:MatrizMP);
var i, j, k: integer;
begin
  for i := 1 to M do
    for j := 1 to P do
    begin
      c[i,j] := 0;
      for k := 1 to N do
        c[i,j] := c[i,j] + a[i,k]*b[k,j]
    end
end;
```

8. Dadas las siguientes declaraciones:

```
const
   N = ...; {valor mayor estricto a 1}
type
   Digito = '0'..'9';
   Digitos = array [1..N] of Digito;
```

(a) Escriba el procedimiento llamado *leerArreglo* que cargue un arreglo de dígitos con valores leídos desde la entrada.

```
procedure leerArreglo(VAR a:Digitos);
```

Ver solución en la última parte.

(b) Escriba la función llamada *conversion* que, dado un arreglo de dígitos, convierta la secuencia de dígitos al entero equivalente. Por ejemplo, si `N=5` y el arreglo es `['0','0','1','2','3']`, entonces el entero resultante será 123.

```
function conversion(a:Digitos) : integer;
```

Ver solución en la última parte.

(c) Escriba un programa principal que lea un arreglo de dígitos (usando parte (a)) e imprima el entero equivalente (usando parte b).

```
program ej8;
const
  N = ...; {valor mayor estricto a 1}
type
  Digito = '0'..'9';
  Digitos = array [1..N] of Digito;
var a: Digitos;

procedure leerArreglo (var a: Digitos);
var i: integer;
begin
  for i := 1 to N do
    read(a[i])
end;

function conversion (a: Digitos): integer;
var i, res: integer;
begin
  res := 0;
  for i := 1 to N do
    res := res*10 + ord(a[i]) - ord('0');
  conversion := res
end;

begin
  leerArreglo(a);
  writeln(conversion(a):0)
end.
```

9. Se dice que una matriz cuadrada `a` de dimensiones `NxN` es simétrica cuando `a[i,j] = a[j,i]` para todos los valores posibles de `i` y `j`. Cuando esto ocurre, es posible representar solamente el triángulo superior (también puede hacerse con el inferior) a efectos de almacenar una sola vez los elementos duplicados. Para ello, se utiliza un arreglo unidimensional para "compactar" los elementos del triángulo superior, del siguiente modo: los valores de la fila 1 se almacenan al comienzo del arreglo. A continuación, se agregan los valores de fila 2 (sin el primer elemento). Luego, los valores de la fila 3 (sin los primeros dos elementos), y así sucesivamente.

   Ejemplo para N = 4:

   ```
   Matriz simétrica: 72  50  48  26
                     50  91  10  64
                     48  10  55  30
                     26  64  30  87

   Arreglo unidimensional: [72, 50, 48, 26, 91, 10, 64, 55, 30, 87]
   ```

   (a) Sea `N` una constante con algún valor entero mayor que cero. Declare el tipo `MatrizN` correspondiente a una matriz de enteros con `N` filas y `N` columnas. Calcule en función de `N`, el largo (cantidad de celdas) que debe tener el arreglo unidimensional. Declare el tipo `ArregloUni` correspondiente a dicho arreglo.

   ```
   type
     MatrizN = array [1..N, 1..N] of integer;
     ArregloUni = array [1..(N+1)*N div 2] of integer;
   ```

   (b) Escriba el procedimiento llamado *matriz2arreglo* que dada una matriz simétrica, almacena en el arreglo unidimensional los valores del triangulo superior de la matriz.

   ```
   procedure matriz2arreglo(a:MatrizN; VAR arreglo:ArregloUni);
   ```

```
procedure matriz2arreglo(a:MatrizN; VAR arreglo:ArregloUni);
var i, j, k: integer;
begin
  k := 1;
  for i := 1 to N do
    for j := i to N do
    begin
      arreglo[k] := a[i, j];
      k := k+1
    end
end;
```

(c) Escriba el procedimiento llamado *arreglo2matriz* que dado un arreglo unidimensional carga una matriz simétrica.

`procedure arreglo2matriz(arreglo:ArregloUni; VAR a:MatrizN);`

```
procedure arreglo2matriz(arreglo:ArregloUni; VAR a:MatrizN);
var i, j, k: integer;
begin
  k := 1;
  for i := 1 to N do
  begin
    a[i, i] := arreglo[k];
    k := k+1;
    for j := i+1 to N do
    begin
      a[i, j] := arreglo[k];
      a[j, i] := arreglo[k];
      k := k+1
    end
  end
end;
```

(d) Piense una expresión (en función de `i` y `j`) para calcular cuál es el índice del arreglo unidimensional el cual se corresponde con el valor almacenado en las coordenadas `[i, j]` de la matriz (le será de utilidad para el siguiente subprograma).

Si calculamos el lugar dentro del arreglo como si se guardaran las filas completas de la matriz tendríamos: `n*(i-1) + j`

Si a ese resultado lerestamos la cantida de celdas suprimidas (del triángulo inferior):

`(i-1)*i/2`

Asi obtenemos que la expresión matemática para obtener el índice del arreglo, dado los valores i, j del triángulo superior, es:

`n*(i-1) + j - (i-1)*i/2`

(e) Escriba la función llamada *obtSim* que, dados un arreglo unidimensional `a` y las coordenadas `i` y `j`, devuelve el valor almacenado en la celda del arreglo `a` correspondiente a las coordenadas i y j. Por ejemplo, `obtSim(a,2,4)` debe devolver el valor `64` (almacenado en celda 7 del arreglo).

`function obtSim(a:ArregloUni; i,j:integer) : integer;`

Ver solución en la última parte.

(f) Escriba un programa principal que lea los valores del triangulo superior de la matriz y los almacene directamente en el arreglo unidimensional. Luego, debe leer una secuencia de parejas de coordenadas e invocar a `obtSim` por cada una de ellas para exhibir el valor almacenado en el arreglo unidimensional correspondiente a dichas coordenadas. La secuencia de datos de entrada finaliza con el valor -1.

Ejemplo para N = 4:

```
Ingrese valor para 1 1: 72
Ingrese valor para 1 2: 50
Ingrese valor para 1 3: 48
Ingrese valor para 1 4: 26
Ingrese valor para 2 2: 91
Ingrese valor para 2 3: 10
Ingrese valor para 2 4: 64
Ingrese valor para 3 3: 55
Ingrese valor para 3 4: 30
Ingrese valor para 4 4: 87
Ingrese parejas de coordendas: 3 2 1 3 4 4 -1
Coordenadas: 3 2
Valor almacenado: 10
Coordenadas: 1 3
Valor almacenado: 48
Coordenadas: 4 4
Valor almacenado: 87
```

```
program pr9ej9;
const
  N = ...; {valor mayor estricto a 1}
  M = (N+1)*N div 2;
type
  ArregloUni = array [1..M] of integer;
var
  i, j, k: integer;
  a: ArregloUni;

function obtSim(a:ArregloUni; i,j:integer) : integer;
var aux : integer;
begin
    {Intercambia las coordenadas en caso de no ser del triangulo superior}
    if (i > j) then
    begin
        aux := i;
        i := j;
        j := aux;
    end;
    obtSim := a[n*(i-1) - (i-1)*i div 2 + j]
end;

begin
  k := 1;
  for i := 1 to N do
    for j := i to N do
    begin
      write('Ingrese valor para ', i:0, ' ', j:0, ': ');
      readln(a[k]);
      k := k+1
    end;
  write('Ingrese parejas de coordendas: ');
  read(i);
  while i <> -1 do
  begin
    read(j);
    writeln('Coordenadas: ', i:0, ' ', j:0);
    writeln('Valor almacenado: ', obtSim(a, i, j):0);
    read(i)
  end
end.
```