

# **PROGRAMACIÓN I**

## **AÑO 2025**

# Estructura de datos MATRIZ

- 1 Definición de tipo de dato Matriz
- 2 Operaciones frecuentes en el tipo Matriz
- 3 Ejercitación

# Arreglos bidimensionales: Matrices

- Una **matriz** es una estructura de datos compuesta que permite acceder a cada componente utilizando **dos índices** (fila y columna).
- Estos índices permiten ubicar un elemento dentro de la estructura. El primer índice indica la fila y el segundo índice indica la columna

Un tipo de dato Matriz es una colección indexada de elementos.

## **Características:**

- Homogénea
- Estática
- Acceso directo
- Lineal

# Matrices: Declaración

## En Pascal:

Matriz = **Array** [ fila, columna ] **of** tipo\_elementos;

Es posible indexar los elementos por un índice que corresponde a **cualquier tipo ordinal**:

- Entero
- Carácter
- Subrango

Los elementos de un arreglo pueden pertenecer a **cualquier tipo de datos**:

- Enteros
- Reales
- Caracteres
- Registros
- String
- Lógicos
- Otro arreglo

# Matrices: Declaración

En las matrices se deben tener en cuenta:

- Dimensión física de filas
- Dimensión lógica de filas
- Dimensión física de columnas
- Dimensión lógica de columnas

## **Const**

```
maxfil = ...;  
maxcol = ...;
```

## **Type**

```
rangof = 0..maxfil;  
rangoc = 0..maxcol;  
matriz = array [1..maxfil, 1..maxcol] of integer;  
matriz_dim = record  
    m: matriz;  
    DimFila: rangof;  
    DimCol: rangoc;  
end;
```

# Matrices: Carga de datos

## Const

maxfil = ...; maxcol = ...;

## Type

rangof = 0..maxfil;

rangoc = 0..maxcol;

matriz = **array** [1..maxfil, 1..maxcol] **of** integer;

matriz\_dim = **record**

M: matriz;

DimFila: rangof;

DimCol: rangoc;

**end;**

Si dimFila es 4, dimCol es 3 y se leen los siguientes valores:

1 3 2 4 7 6 8 9 15 10 11 5



1	3	2		
4	7	6		
8	9	15		
10	11	5		

**Procedure** Cargar ( var mD: matriz\_dim);

**Var** i: rangoF;

j: rangoc;

**Begin**

readln(mD.DimFila);

readln(mD.DimCol);

**if** Validar(mD.DimFila, mD.DimCol) **then**

**For** i:=1 **to** mD.DimFila **do**

**For** j:= 1 **to** mD.DimCol **do**

readln(mD.M[i,j]);

**End;**

Esta carga se realiza por filas, pero se podría cargar por columnas...

# Matrices: Imprimir

## Const

maxfil = ...; maxcol = ...;

## Type

rangof = 0..maxfil;

rangoc = 0..maxcol;

matriz = **array** [1..maxfil, 1..maxcol] **of** integer;

matriz\_dim = **record**

M: matriz;

DimFila: rangof;

DimCol: rangoc;

**end**;

Observar  
Parámetros

1	3	2		
4	7	6		
8	9	15		
10	11	5		

1  
3  
2  
4  
7  
6  
8  
9  
15  
10  
11  
5




El recorrido de la matriz está  
realizado por filas, también  
podría realizarse por  
columnas...

**Procedure** Imprimir (var mD: matriz\_dim);  
**var** i: rangof;  
      j: rangoc;  
**Begin**  
      **For** i:=1 **to** mD.DimFila **do**  
          **For** j:= 1 **to** mD.DimCol **do**  
              writeln(mD.M[i,j]);  
**End**;

# Matrices: Buscar un elemento

```
Function Buscar ( mD: matriz_dim): boolean;  
var  
    existe : boolean;  
    i: rangof;  
    j: rangoc;  
  
Begin  
    existe := false;  
    i:= 1;  
    while ( i<= mD.DimFila) and (not existe) do  
    begin  
        j := 1;  
        while ( j <= mD.DimCol) and (not existe) do  
            if (mD.M[i,j] = elem) then existe := true  
                else j := j+1;  
        if (existe = false) then i := i+1;  
    end;  
    Buscar := existe;  
End;
```

Buscar el elemento 8



1	3	2		
4	8	6		
8	9	15		
10	11	5		

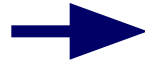
La búsqueda de la matriz está realizada por filas, también podría realizarse por columnas...



# Matrices: Eliminar una fila

Supongamos que necesitamos eliminar la fila 2 de la siguiente matriz.

Pasa a ser una fila no accesible



8	2	1	4	
9	6	4	4	
0	3	7	6	



DimF= 2

DimC= 4

# Matrices: Eliminar una fila

```
Procedure EliminarFila ( var mD: matriz_dim; fila_a_eliminar: rangoF;  
                        var exito:boolean);  
  
var  
    k: rangoC; f :rangoF;  
begin  
    exito:= false;  
    If ( fila_a_eliminar > 0 and fila_a_eliminar <= mD.DimFila ) { verifica que el nro. de fila  
                                                es válido, para efectuar la operación}  
    then begin  
        exito := true;  
        For f := fila_a_eliminar + 1 to mD.DimFila - 1 do  
            For k := 1 to mD.DimCol do  
                mD.M [f - 1, k] := mD.M [f, k];  
        mD.DimFila := mD.DimFila - 1  
    end  
end;
```

Qué pasa si la fila a eliminar  
es la última?

# Matrices: Suma

## Const

maxfil = ...; maxcol = ...;

## Type

rangof = 0..maxfil;

rangoc = 0..maxcol;

matriz = **array** [1..maxfil, 1..maxcol] **of** integer;

matriz\_dim = **record**

    M: matriz;

    DimFila: rangof;

    DimCol: rangoc;

**end**;

**Var** mD1, mD2, mD3: matriz\_dim;

## A tener en cuenta:

- Sólo se pueden sumar matrices de igual dimensión lógica de filas e igual dimensión lógica de columnas.

## Begin

Cargar(mD1);

Cargar(mD2);

**if** (mD1.DimFila = mD2.DimFila) and  
    (mD1.DimCol = mD2.DimCol)

**then Begin**

    Sumar(mD1, mD2, mD3);

    Imprimir(mD3);

**end**

**else** writeln ('No se pueden sumar');

**end.**

# Matrices: Suma

**M1**

1	3	2	
4	7	6	
8	9	15	
10	11	5	

+

**M2**

-1	1	2	
1	2	4	
2	3	-4	
5	5	5	

=

**M3**

0	4	4	
5	9	10	
10	12	11	
15	16	10	

```
Procedure Sumar (var mD1, mD3: matriz;  
                  var mD2: matriz);
```

```
var
```

```
  i: rangof;
```

```
  j: rangoc;
```

```
Begin
```

```
  mD3.DimFila:= mD1.DimFila;
```

```
  mD3.DimCol:= mD1.DimCol;
```

```
  For i:=1 to mD3.DimFila do
```

```
    For j:= 1 to mD3.DimCol do
```

```
      mD3.m[i,j] := mD1.m[i,j] + mD2.m[i,j];
```

```
end;
```

# Matrices: Producto

## Const

maxfil = ...; maxcol = ...;

## Type

rangof = 0..maxfil;

rangoc = 0..maxcol;

matriz = **array** [1..maxfil, 1..maxcol] **of** integer;

matriz\_dim = **record**

    M: matriz;

    DimFila: rangof;

    DimCol: rangoc;

**end**;

**Var** mD1, mD2, mD3: matriz\_dim;

## A tener en cuenta:

- Es posible hacer el producto si la cantidad de columnas de la primera matriz coincide con la cantidad de filas de la segunda matriz.
- La matriz producto tendrá la cantidad de filas de la primera matriz y la cantidad de columnas de la segunda matriz.

## Begin

Cargar(mD1);

Cargar(mD2);

**if** (mD1.DimCol = mD2.DimFila)

**then Begin**

    Producto(mD1, mD2, mD3);

    Imprimir(mD3);

**end**

**else** writeln('Estas matrices no se pueden multiplicar');

**end.**

# Matrices: Producto

```
Procedure Producto (Var mD1, mD2: matriz_dim; var mD3: matriz_dim);  
  var  
    fil, k : rangoF;  
    col : rangoC;  
  
begin  
  mD3.DimFila:= mD1.DimFila;  
  mD3.DimCol:= mD2.DimCol;  
  for fil := 1 to mD3.DimFila do  
    for col := 1 to mD3.DimCol do  
      begin  
        mD3.M[fil,col] := 0;  {inicializa valor a resolver}  
        for k := 1 to mD1.DimCol do  
          mD3.M[fil,col] := mD3.M[fil,col] + mD1.M [fil,k] * mD2.M [k,col];  
        end;  
      End;
```

# Matrices: Ejercitación

*Ejercicio 1:* Un teatro pone a la venta las entradas a un musical. La sala donde se realizará el musical dispone de 50 filas con 70 butacas por cada fila. Implementar un programa que permita la venta de entradas. Cada cliente indica la ubicación de su interés y si está ocupada, no se puede vender. En el caso que esté libre, se debe registrar la compra con el dni del cliente.

La venta de entradas finaliza cuando llega el cliente con dni -1 o cuando no quedan más entradas disponibles.

Nota: El cliente puede solicitar una entrada por vez.

*Ejercicio 2:* Implementar un módulo que reciba una matriz de productos y retorne el código de producto más caro. De cada producto se conoce el código, el stock y el precio unitario.

*Ejercicio 3:* Implementar un módulo que inserte una fila en una matriz. El módulo debe recibir la matriz, un número de fila y un vector que contenga los valores de la fila a insertar.