



# TEORIA

## PROGRAMACIÓ CIENTÍFICA

---

T4A  
Taules i Cadenes



# TAULES

# Taules

## Què són?

- Què passa si necessitem guardar moltes variables del mateix tipus?
- Farem servir **taules**. Són estructures que guarden diverses variables del **mateix tipus**.

- **Exemple:** el número del DNI de cadascun dels estudiants

```
algorisme principal és
var
    dni1: enter;
    dni2: enter;
    dni3: enter;
    dni4: enter;
    dni5: enter;
    dni6: enter;
    dni7: enter;
    ...
fvar
inici
    ...
falgorisme
```



```
algorisme principal és
const
    NUM_ESTUDIANTS := 20;
fconst
var
    taula_dni: taula[NUM_ESTUDIANTS] d'enters;
    ...
fvar
inici
    ...
falgorisme
```



# Taules

## Què són?

- Què passa si necessitem guardar moltes variables del mateix tipus?
- Farem servir **taules**. Són estructures que guarden diverses variables del **mateix tipus**.
- Les taules poden tenir diverses **dimensions**
  - 1 dimensió: vector
  - 2 dimensions: matriu
  - n dimensions ...

Taula d'una dimensió, o vector

21	4	99	99	-7	...	37	9
----	---	----	----	----	-----	----	---

Taula de dues dimensions, o matrius

21	4	99
-10	34	17

Taules de >2 dimensions

21	4	99
21	4	99
-10	34	17

# Taules

## Definició de taules d'una dimensió (vectors)

Nom de  
la taula

Mida de  
la taula

Tipus que  
emmagatzema

\$ Definició d'un vector de 30 elements com a màxim

**var**

taula\_edats: taula[30] d'enters;

**fvar**

\$ Si declarem una constant, podem redefinir la mida de

\$ vàries taules només canviant el valor de la constant

**const** NUM\_PERSONES := 30; **fconst**

**var**

taula\_edats: taula[NUM\_PERSONES] d'enters;

taula\_pesos: taula[NUM\_PERSONES] de reals;

**fvar**



Recordeu !

Hard-coding NO

...

a = b + 100;

c = 100 + a;

...



**const**

MAX := 100;

**fconst**

...

a = b + MAX;

c = MAX + a;

...



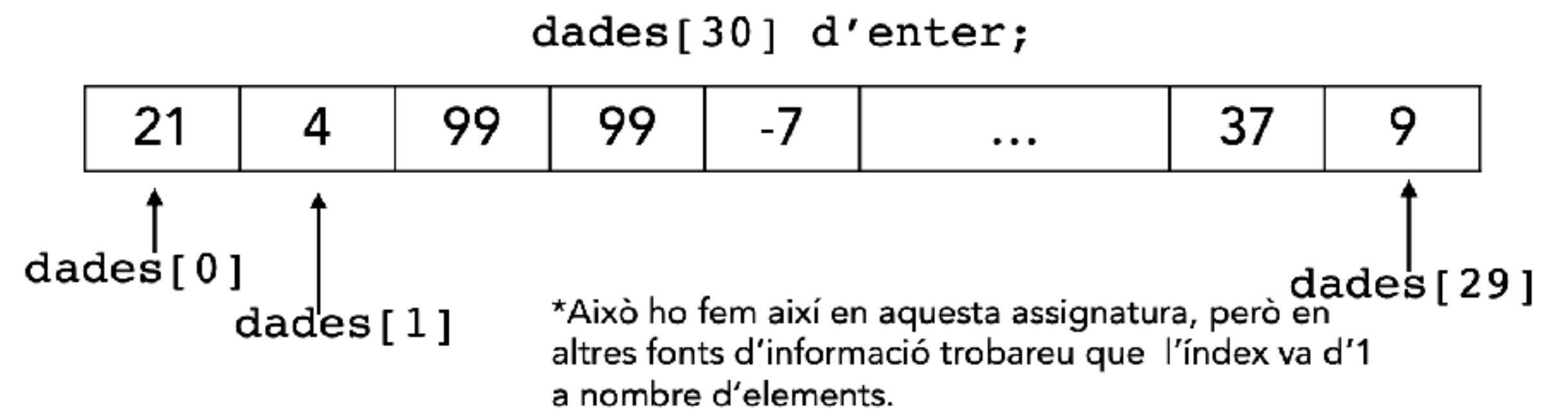
# Taules

## Accés a taules d'una dimensió (vectors)

- Per accedir a un element cal indicar el nom de la taula i l'índex o posició de l'element

## Definició

```
const
    MAX_NUMS := 30;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
fvar
```



## Escriure a taula

```

...
dades[0] := 21;
dades[1] := 4;
...
dades[MAX_NUMS-1] := 9;
...

```

## Llegir de taula

```
var
    numeret : enter;
fvar
    numeret := dades[0];
    $ numeret val 21
...

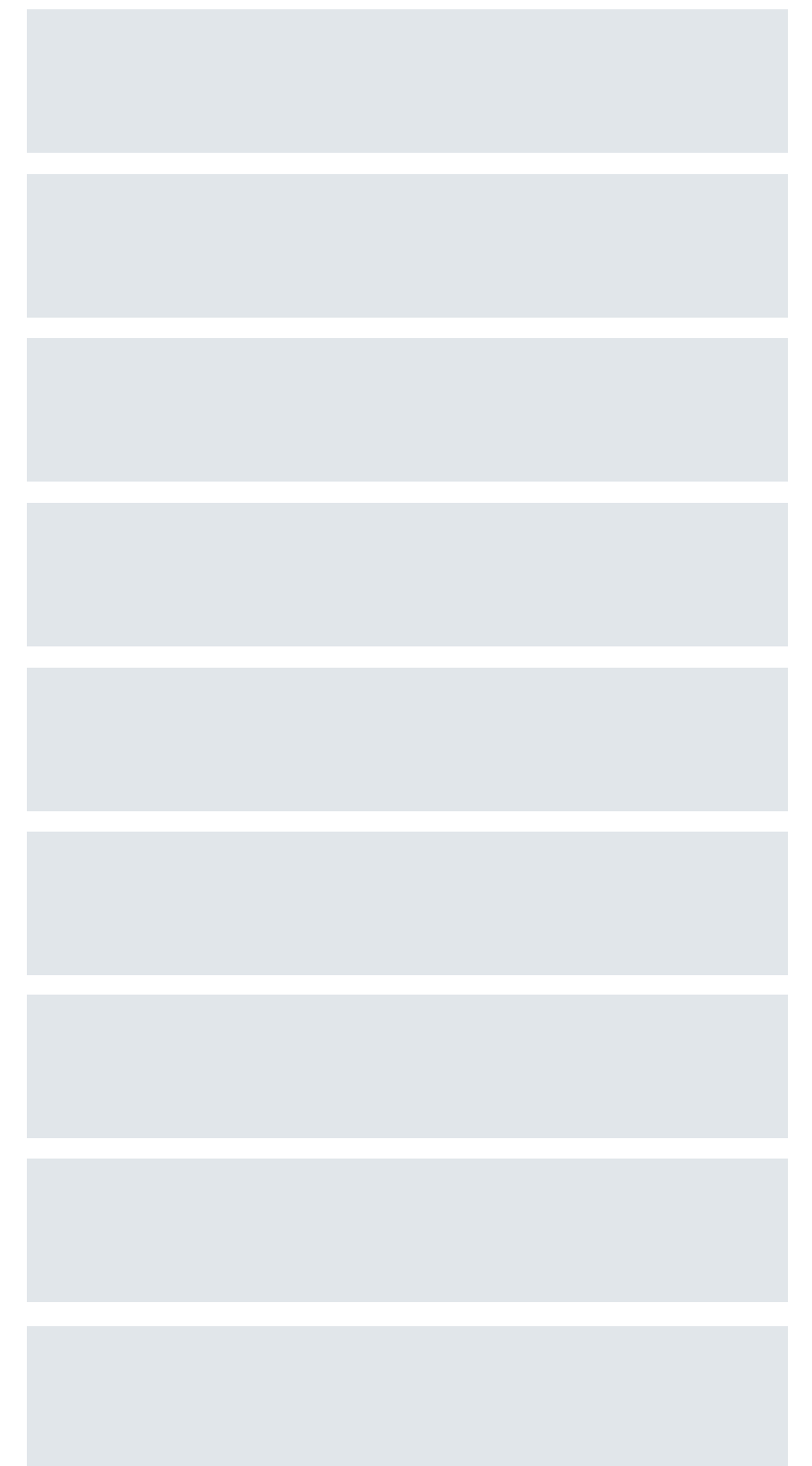
```

## Com funcionen les taules realment?

- Internament, les taules no són res més que **posicions de memòria contígues**
- Quan declarem una taula com aquesta:

```
const
    MAX_NUMS := 5;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
fvar
```

- Es reserva espai de memòria de mida NUM\_ELEMS \* mida d'un element
- $5 * \text{sizeof}(\text{int}) = 5 * 4 \text{ Bytes (32 bits)} = 20 \text{ Bytes}$



*Memòria de 32 bits*

## Com funcionen les taules realment?

- Internament, les taules no són res més que **posicions de memòria contígues**
- Quan declarem una taula com aquesta:

```
const
    MAX_NUMS := 5;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
fvar
```

- Es reserva espai de memòria de mida NUM\_ELEMS \* mida d'un element
  - $5 * \text{sizeof}(\text{int}) = 5 * 4 \text{ Bytes (32 bits)} = 20 \text{ Bytes}$
- Per accedir-hi, se suma l'adreça base + l'índex on volem accedir

```
dades[0] := 21;
```

$$@ 1000 + 0 = @ 1000$$

**@ 1000**

**21**

@ 1001

@ 1002

@ 1003

@ 1004

*Memòria de 32 bits*



## Com funcionen les taules realment?

- Internament, les taules no són res més que **posicions de memòria contígues**
- Quan declarem una taula com aquesta:

```
const
    MAX_NUMS := 5;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
fvar
```

- Es reserva espai de memòria de mida  $\text{NUM\_ELEMS} * \text{mida d'un element}$ 
  - $5 * \text{sizeof(int)} = 5 * 4 \text{ Bytes (32 bits)} = 20 \text{ Bytes}$
- Per accedir-hi, se suma l'adreça base + l'índex on volem accedir

```
dades[0] := 21;
dades[3] := 99;
```

@ 1000 + 3 =  
@ 1003

@ 1000

21

@ 1001

@ 1002

**@ 1003**

**99**

@ 1004

Memòria de 32 bits

## Com funcionen les taules realment?

- Internament, les taules no són res més que **posicions de memòria contígues**
- Quan declarem una taula com aquesta:

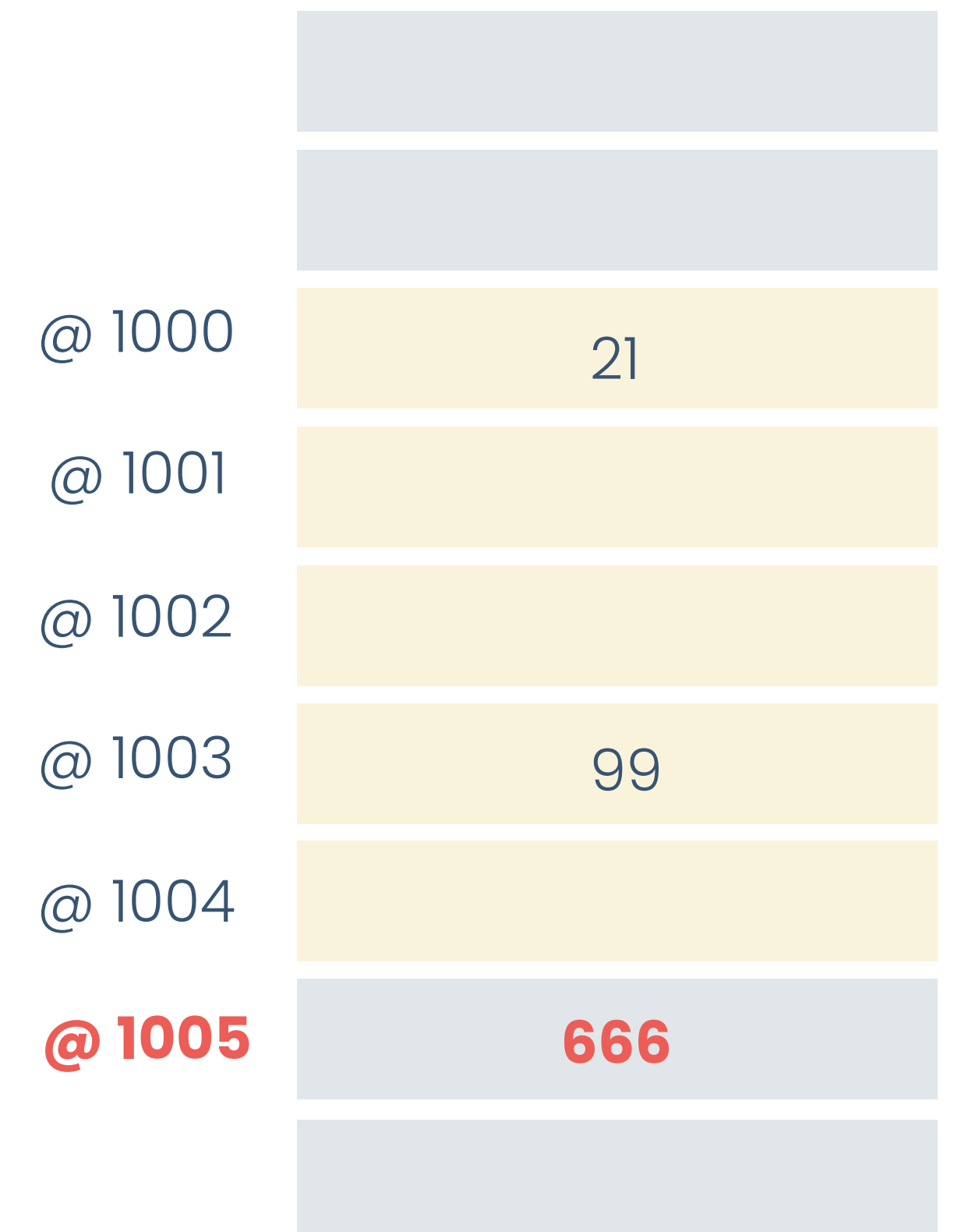
```
const
    MAX_NUMS := 5;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
fvar
```

- Es reserva espai de memòria de mida  $\text{NUM\_ELEMS} * \text{mida d'un element}$
- $5 * \text{sizeof(int)} = 5 * 4 \text{ Bytes (32 bits)} = 20 \text{ Bytes}$
- Per accedir-hi, se suma l'adreça base + l'índex on volem accedir

```
dades[0] := 21;
dades[3] := 99;
dades[5] := 666;
```

@ 1000 + 5 = @ 1005

*Acabem de sobre escriure  
memòria que es feia servir  
per altres coses! ERROR!*



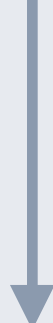
Memòria de 32 bits

**NOTA:** En C passa així, en altres llenguatges de programació senzillament hi ha un error de compilació.

# Taules

## Recorreguts de taules

- Ja hem vist que llegir/escriure fora de l'espai de memòria reservat provoca errors de compilació o comportaments indesitjats.
- Per tant, a l'hora de fer recorreguts, hem d'anar molt en compte amb no sortir-nos de la taula
  - Quan definim una taula, definim el nombre màxim d'elements d'aquella taula



```
const
    MAX_NUMS := 5;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
fvar
```

- Ens hem d'assegurar de recórrer des de la primera posició fins l'última, sense deixar-nos-en cap ni sortir-nos.

```
...
per (i:=0; i<MAX_NUMS; i++)
    dades[i] := 0;
fper
...
```

*Si ja sabem que tenim menys elements que MAX\_NUMS, només recorrerem fins el nombre d'elements que tenim*

# Taules

## Exemple

- Algorisme que declari una taula unidimensional de longitud = 10 i n'imprimeixi els continguts

```
algorisme imprimir_taula és
  const
    N := 10;
  fconst
  var
    i: enter;
    dades: taula[N] de real;
  fvar
  inici
    per (i:=0; i<N; i:=i+1) fer
      escriure("A la casella ", i, " hi ha: ", dades[i]);
    fper
falgorisme
```

*Quina sortida creieu que té aquest algorisme?*



## Inicialització de les taules

- Quan es reserva l'espai de memòria, se'ns assegura que tindrem aquell espai reservat però no sabem què hi ha dins

- Abans de reservar:

1243775
7575375
245747
7567575
-7447675
467657
756767
0
537839

Memòria de 32 bits

- Després de reservar:

1243775
7575375
245747
7567575
-7447675
467657
756767
0
537839

Memòria de 32 bits

```
const
    MAX_NUMS := 5;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
    a: enter;
fvar
...
    a := dades[0] + 1;    $ a = 245748
...
```

*Hem d'inicialitzar les taules per assegurar-  
nos que tenen el valor que toca*

# Inicialització de les taules

- Quan es reserva l'espai de memòria, se'ns assegura que tindrem aquell espai reservat però no sabem què hi ha dins

• Abans de reservar:

1243775
7575375
245747
7567575
-7447675
467657
756767
0
537839

Memòria de 32 bits

• Després de reservar:

1243775
7575375
0
0
0
0
0
0
537839

Memòria de 32 bits

```
const
    MAX_NUMS := 5;
fconst
var
    dades: taula[MAX_NUMS] d'enters;
fvar
...
per (i:=0; i<MAX_NUMS; i++)
    dades[i] := 0;
fper
...
```

```
...
a:= dades[0] + 1;    $ a = 1
...
```

# Taules

## Exemple

- Algorisme que desi a una taula els nombres introduïts per teclat. Preguntarem a l'usuari quants en vol introduir, els llegirem, i els desarem a una taula.



```
algorisme de_teclat_a_taula és
  const
    MAX_ELEMS := 100;
  fconst
  var
    num_elems, i: enter;
    nombres: taula[N] de real;
  fvar
  inici
    escriure("Quants elements vols desar a la taula?");
    llegir(num_elems);
    si(num_elems > MAX_ELEMS) llavors
      escriure("Massa elements! El màxim és ", MAX_ELEMS);
    sino
      per (i:=0; i<num_elems; i:=i+1) fer
        llegir(dades[i]);
      fper
    fsi
  falgorisme
```

# Taules

## Operacions amb taules

- Les taules d'elements no es poden comparar, ni assignar amb els operadors (:=, =, >, ...)
- El codi següent pretén comparar si dos vectors són iguals

```
var
    dades, càlculs: taula[MAX_EL] d'enters;
fvar
inici
...
    si (dades ≠ càlculs) llavors
        dades := càlculs;
fsi
```

NO!

Per fer aquestes operacions caldrà fer un algorisme/procediment que ho dugui a terme!

### Parèntesi:

### Per què no funciona fer-ho així?

- En C, "dades" i "càlculs" guarda l'adreça de memòria d'on comença cada vector

dades = @1000	7575375
	245747
	7567575
	-7447675
càlculs = @ 1004	575679
	756767
	467657
	467657
	1243775

Fer aquesta comparació: **dades = calculs ?**  
En realitat és fer aquesta comparació: **1000 = 1004 ?**



# Taules

**Exercici:** Escriure un algorisme que, donada una taula de nombres reals, en retorna la seva mitjana

```
algorisme mitjana és
  const

  fconst
  var

  fvar
  inici

    $ Suposem que la taula l'omplim de nombres

falgorisme
```

# Taules

## Matrius: taules de dues o més dimensions

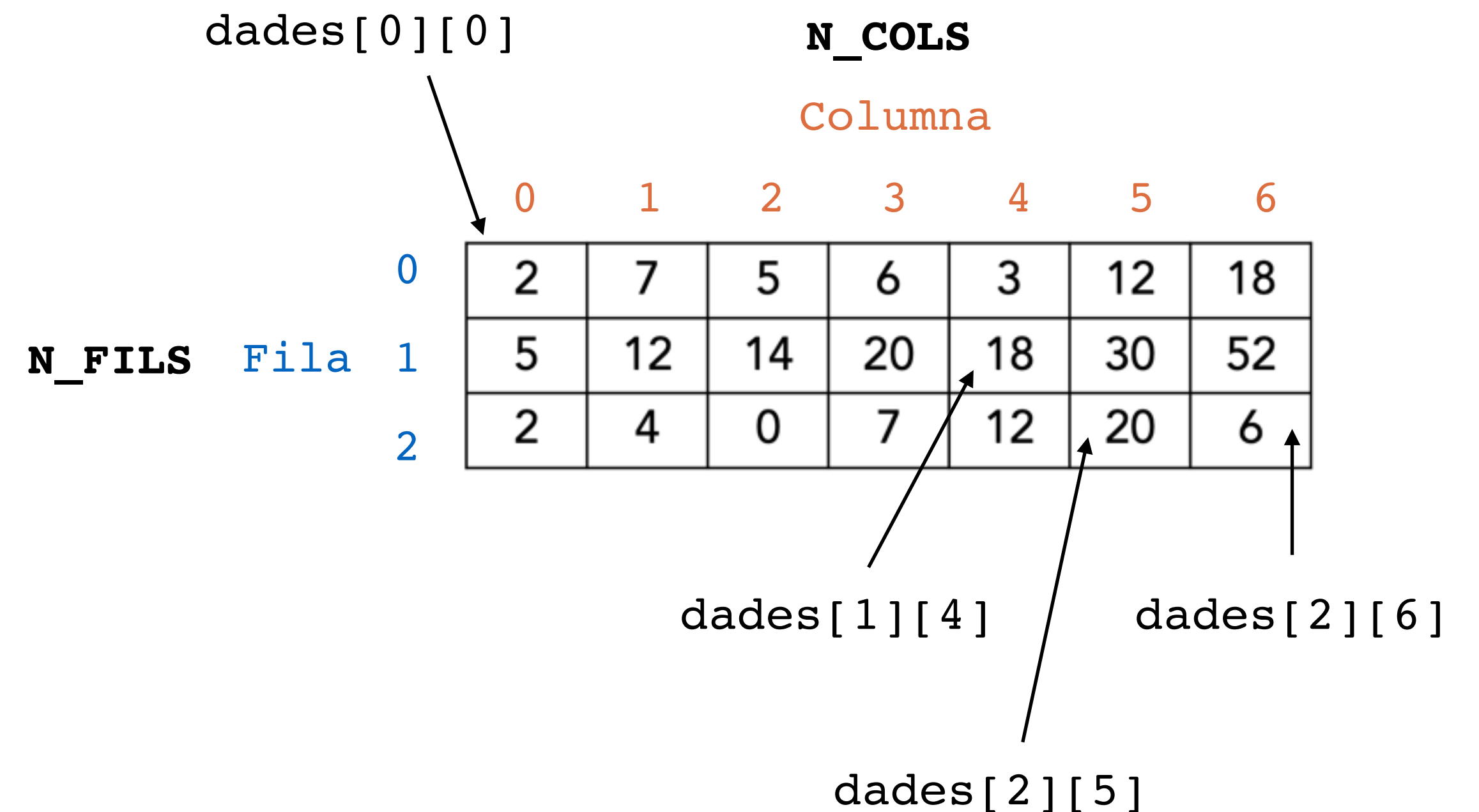
- Es defineixen i recorren igual que les d'una dimensió però tenint en compte **files** i **columnes**

### Definició

```
const
  N_FILS := 3;
  N_COLS := 7;
fconst
var
  dades: taula[N_FILS][N_COLS] d'enters;
fvar
```

### Escriure a taula / Llegir de taula

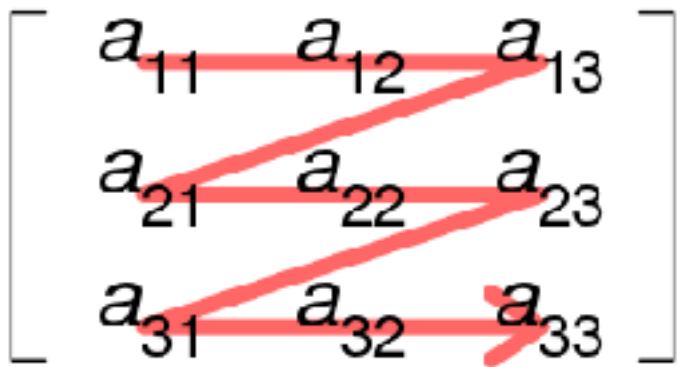
```
...
dades[0][0] := 2;
dades[1][4] := 18;
dades[2][6] := 6;
...
a := dades[2][5]; $ a = 20
...
```



# Parèntesi tècnic: Com s'emmagatzemen les matrius a memòria?

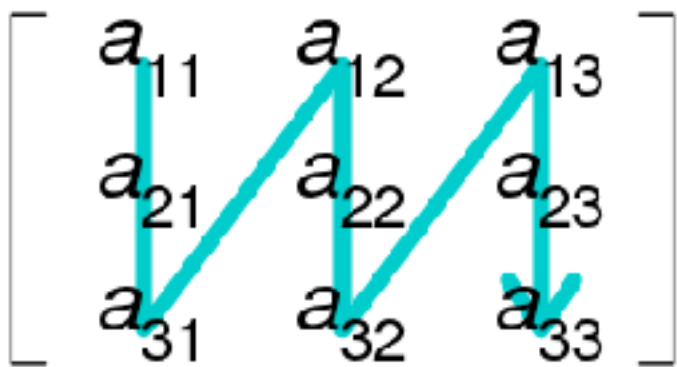
- Com s'emmagatzemen les matrius en memòria?
- Cada llenguatge de programació (o llibreria) té una manera particular d'emmagatzemar matrius en memòria
- És necessari saber com s'emmagatzema per poder fer recorreguts eficients.

## Row-major order

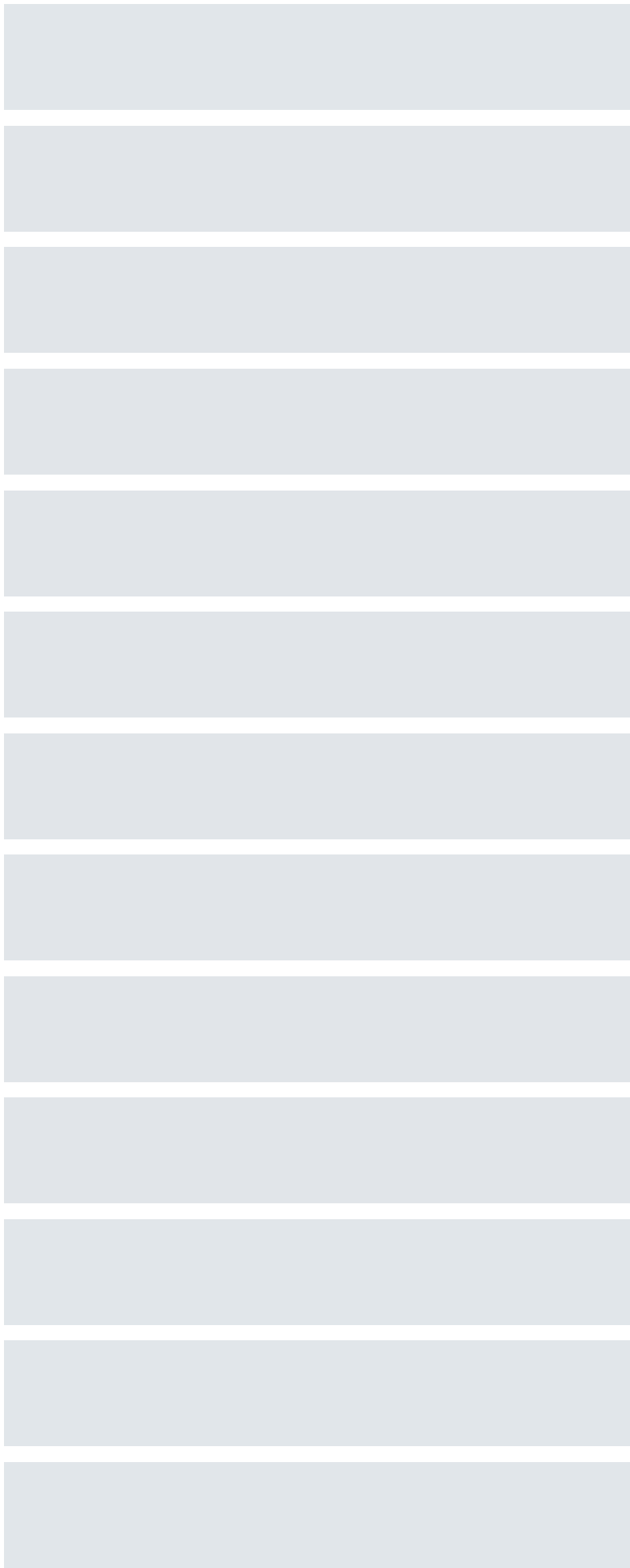


C / C++  
NumPy (Python)

## Column-major order



Fortran  
Matlab  
R  
Julia



Memòria de 32 bits

# Parèntesi tècnic: Com s'emmagatzemen les matrius a memòria?

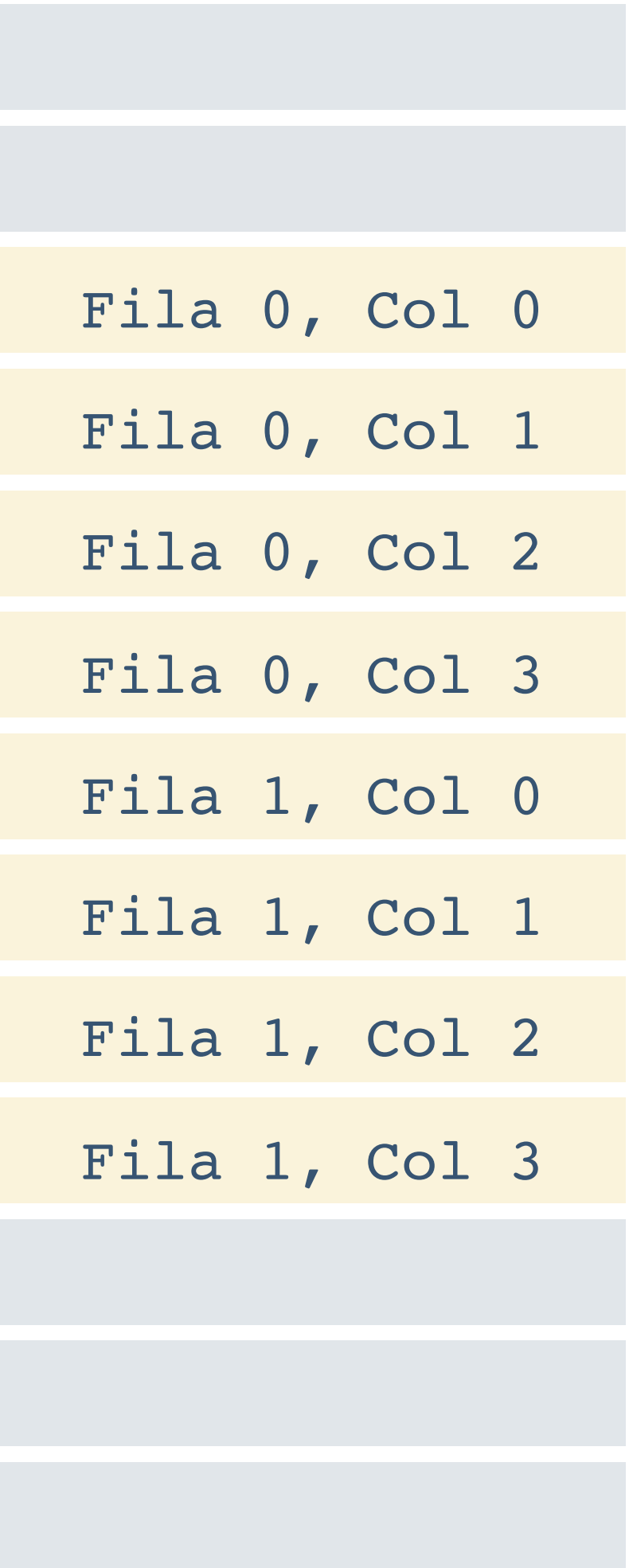
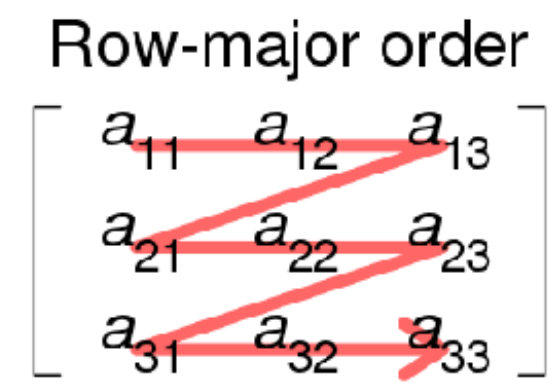
- En llenguatge C, quan definim una matriu:

```
const
    N_FILS := 2;
    N_COLS := 4;
fconst
var
    dades: taula[N_FILS][N_COLS] d'enters;
fvar
```

		Columna			
		0	1	2	3
Fila	0	2	7	5	6
	1	5	12	14	20

- Es reserva espai de memòria:
  - N\_FILS \* N\_COLS \* mida de l'element

- I aquest espai s'emmagatzema per files



Memòria de 32 bits



# Recorregut eficient de matrius

- A l'hora de recórrer, nosaltres podríem fer:
  - Un recorregut per files:

```
per (i:=0; i<N_FILS; i++)
  per (j:=0; j<N_COLS; j++)
    dades[i][j] := 0;
  fper
fper
```

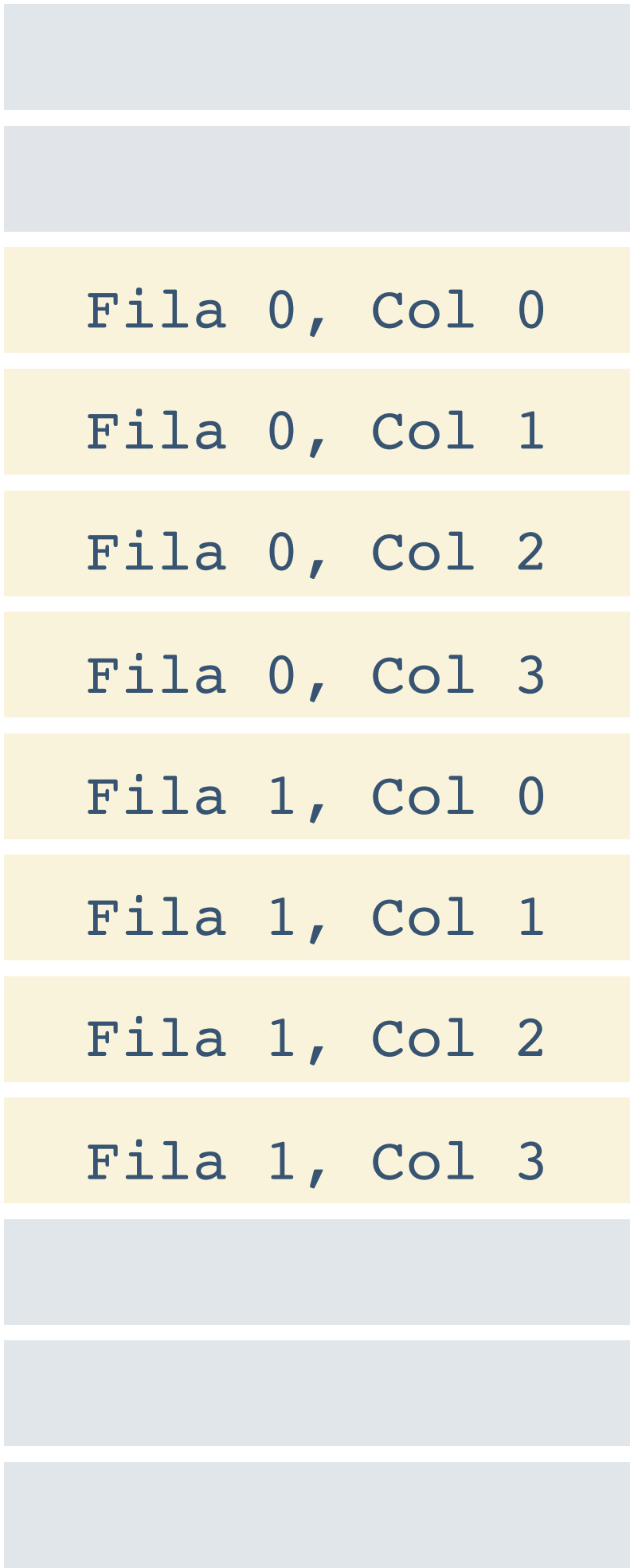
```
dades[0][0]
dades[0][1]
dades[0][2]
dades[0][3]
dades[1][0]
dades[1][1]
dades[1][2]
dades[1][3]
```

- O un recorregut per columnes

```
per (j:=0; j<N_COLS; j++)
  per (i:=0; i<N_FILS; i++)
    dades[i][j] := 0;
  fper
fper
```

```
dades[0][0]
dades[1][0]
dades[0][1]
dades[1][1]
dades[0][2]
dades[1][2]
dades[0][3]
dades[1][3]
```

- Els dos són correctes, però el recorregut per files és més eficient perquè minimitzem el nombre d'accessos a memòria.
  - Tècnicament, quan llegim de memòria llegim per blocs, per tant, fent un recorregut per files ja tenim el valor següent en el bloc carregat.



Memòria de 32 bits

# Taules

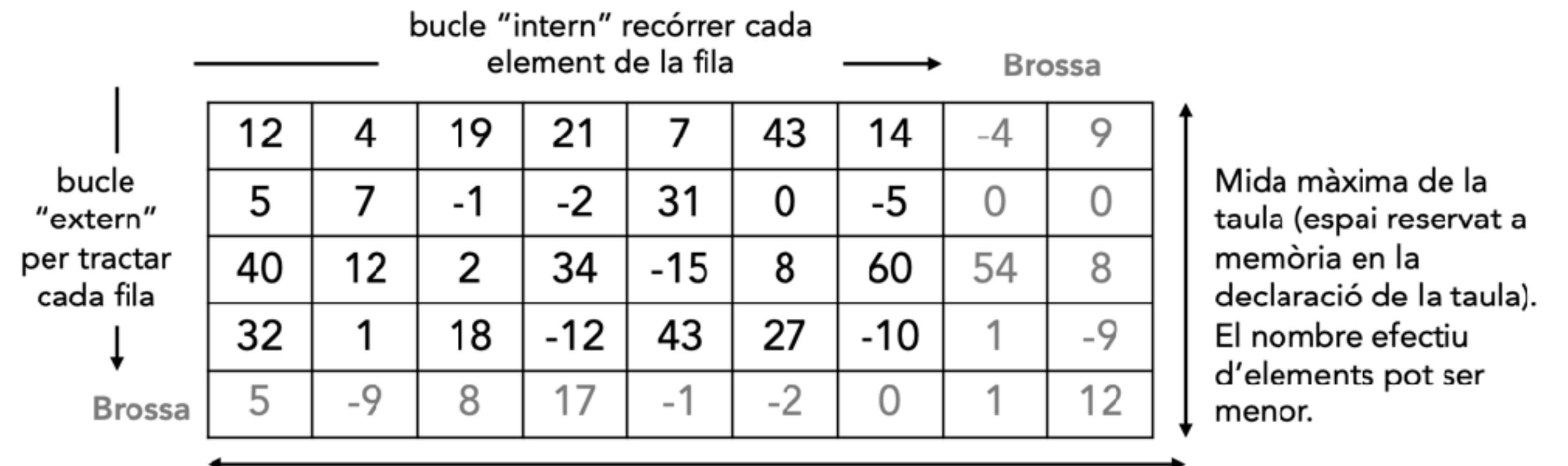
## Recorregut de matrius

- Hem de recórrer tant les files com les columnes

```
const
    N_FILS := 3;
    N_COLS := 7;
fconst
var
    dades: taula[N_FILS][N_COLS] d'enters;
fvar
```

```
per (i:=0; i<N_FILS; i++)
    per (j:=0; j<N_COLS; j++)
        dades[i][j] := 0;
    fper
fper
```

Necessitem dos bucles, un per tractar cada fila i un altre per tractar totes les dades de cada fila (tants elements/cel·les com columnes té la taula).



# Taules

## Exercici:

Inicialitzar una matriu de tal manera que el valor de la casella sigui la suma de les seves coordenades (i,j)

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

```
algorisme inicialitzar_matriu és
  const
    NUM_FILES := 10;
    NUM_COLS := 10;
  fconst
  var
    m: taula[NUM_FILES][NUM_COLS] de enter;
    i,j: enter;
  fvar
  inici
    per (i:=0; i<NUM_FILES; i:=i+1)
      per(j:=0; j<NUM_COLS; j:=j+1)
        m[i][j] = i + j;
      fper
    fper
falgorisme
```

# Taules

**Exercici:** Trobar el valor mínim d'una taula d'enters

```
algorisme trobar_minim és
  const
    NUM_FILES := 50;
    NUM_COLS := 20;
  fconst
  var
    min, i, j : enter;
    m: taula[NUM_FILES][NUM_COLS] de enter;
  fvar
  inici
    $ suposem que la matriu és plena de dades
  per (i:=0; i<NUM_FILES; i:=i+1)
    per(j:=0; j<NUM_COLS; j:=j+1)
      si (dades[i][j] < min) llavors
        min := m[i][j];
      fsi
    fper
  fper
  escriure("El mínim trobat és: ", min);
falgorisme
```



**CADENES**

# Cadenes de caràcters

## Què són?

- Són una taula de caràcters que permeten desar paraules i frases.
- Tenen la particularitat que acaben amb un sentinella ('\0') que indica el final de la cadena.
- Es desen en taules de caràcters, de mida determinada.
- Cada caràcter ocuparà una posició de la taula, i el sentinella també.

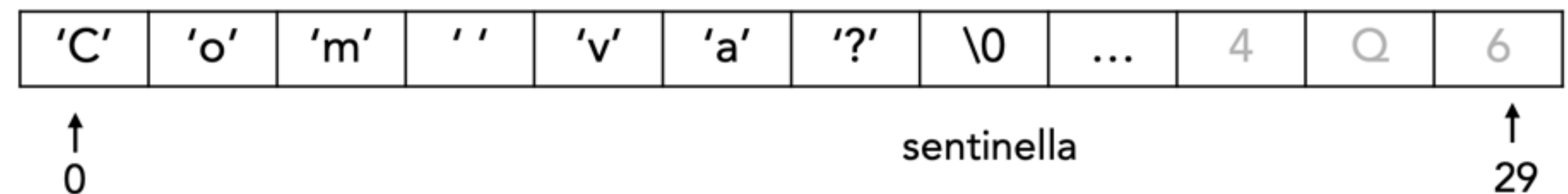
## Definició

```

algorisme test_caracters és
  const
    MAX_C := 100;
  fconst
  var
    frase: taula[MAX_C] de caràcters;
  fvar
  inici
    frase := "Com va?";
falgorisme

```

La cadena d'exemple  
es desa en una taula  
de 30 caràcters.



En aquesta assignatura, i per similitud al llenguatge C, el sentinella en les cadenes de caràcters és '\0', que indica que en la posició de la taula on hi ha el sentinella tots els bits són 0.