

$p(4k-2)^2$  ✓

$i=0$

$(4k-3)^2 \quad p(4k-4)^2 < (4k-8)^2 \quad p(4k-7)^2$

$(4k-6)^2 \quad p(4k-5)^2 < (4k-9)^2 \quad p(4k-10)^2$

$\dots$

$(4k-1)^2 \quad p(4k-2)^2 < (4k-3)^2 \quad p(4k-4)^2$

$= P(4k)^2_k$

left  $< \circ < \circ <$

$i+j$

$(-1)$

$3^2$

$2^2$

$1^2$

$5^2$

bq

all c

# FONAMENTS DE PROGRAMACIÓ II

## GRAUS: GEI, GEI-Biotec

UNIVERSITAT ROVIRA I VIRGILI

# **TAULES I CADENES**

# Taules

## Per què taules?

- 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

## Per què taules?

- 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 / array
  - 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: Recordatori de pseudocodi

## Declaració

```
const
  N := 30;
  M := 10;
fconst
var
  vector: taula[M] d'enters;
  matriu: taula[M][N] de real;
fvar
```

 M és files, N és columnes

## Escriure a taula

```
...
vector[0] := 21;
vector[1] := 4;
...
vector[N-1] := 9;
...
matriu[0][0] = 3.4;
matriu[M-1][N-1] = 5;
...
```

## Llegir de taula

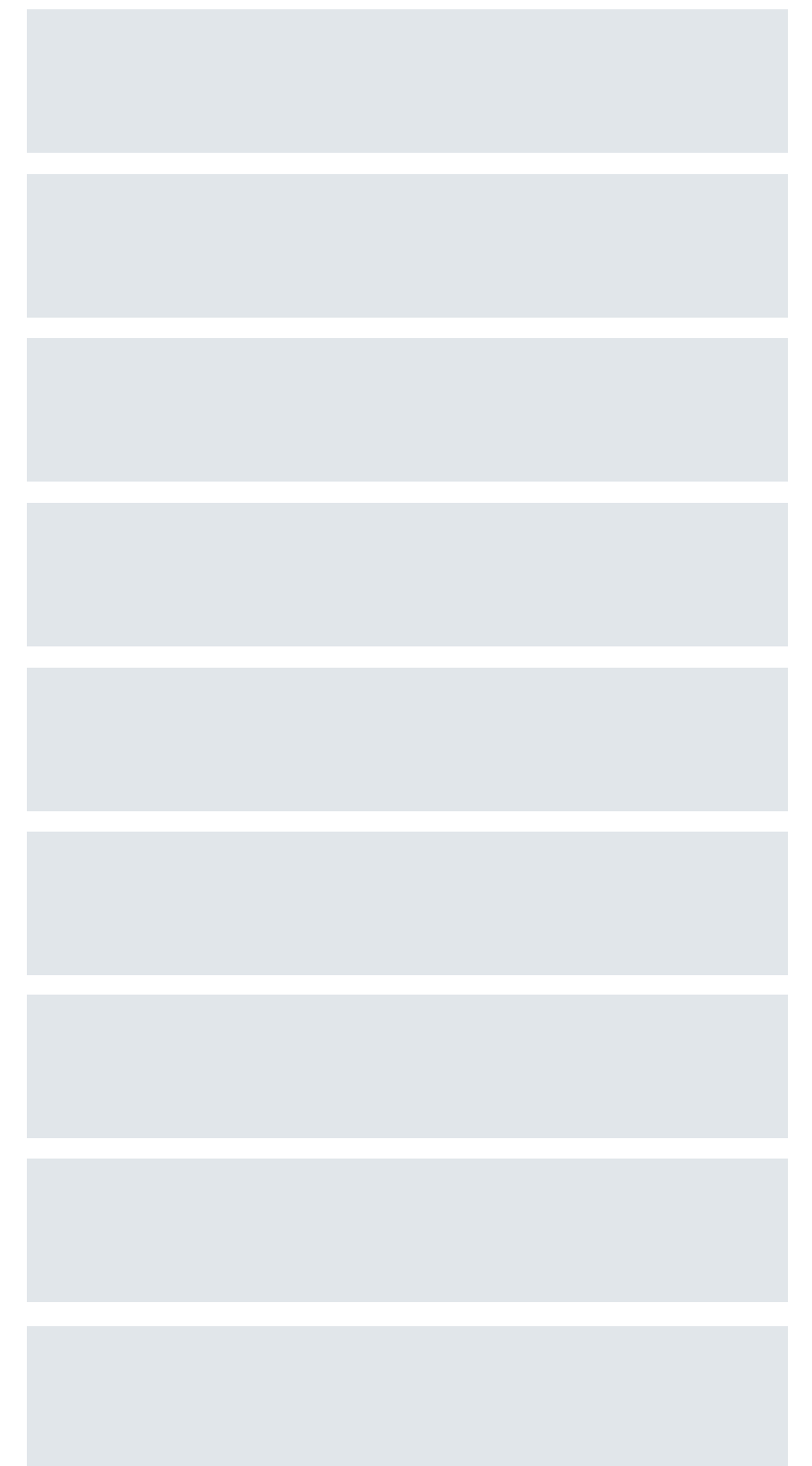
```
var
  numeret : enter;
  altre_numeret : real;
fvar
  numeret := dades[0];
  $ numeret val 21
...
  altre_numeret := matriu[0][0];
  $ altre_numeret val 3.4
...
```

## 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**

@ 1001

@ 1002

@ 1003

@ 1004

**21**

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

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

@ 1000 + 5 = @ 1005

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

**Compte! Això és una  
simplificació, els índexos  
no van d'1 en 1**

@ 1000

21

@ 1001

@ 1002

@ 1003

99

@ 1004

**@ 1005**

**666**

Memòria de 32 bits

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

# Taules

## Operacions amb taules

- Ja sabeu que les taules d'elements no es poden comparar, ni assignar amb els operadors ( $:=$ ,  $=$ ,  $>$ , ...). Sabeu per què?

*El codi següent pretén comparar, erròniament, 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!

## Per què no funciona fer-ho així?

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

<b>dades = @1000</b>	7575375
	245747
	7567575
	-7447675
<b>càlculs = @ 1004</b>	575679
	756767
	467657
	467657
	1243775

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

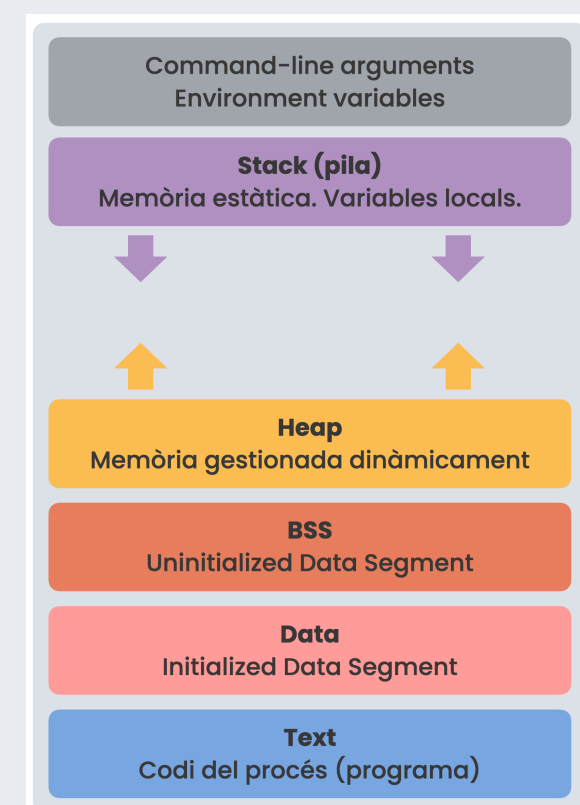
# Taules



On es guarden les taules que defineixo així?

```
#define N 100
int main(){
    int dades[N];
    return 0;
}
```

- Es guarden a la pila



Quin valor inicial tenen les posicions de la taula definida així?

- Tenen valor arbitrari, "brossa". Per això sempre les hem d'inicialitzar.

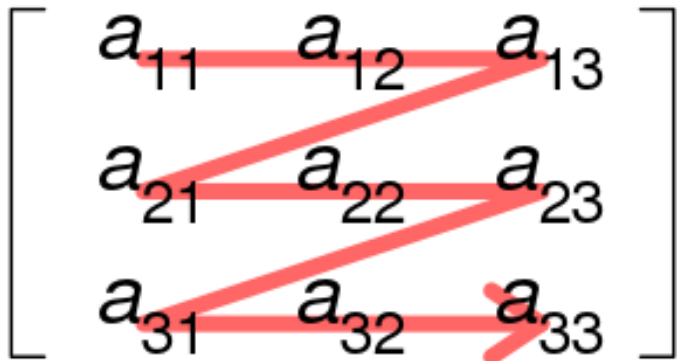
Quin problema pot comportar que la taula es guardi a la pila?

- Si la taula és massa gran, hi pot haver un **desbordament de pila**. La pila és petita!
  - En Windows, el límit predeterminat de la pila és d'**1 MB** per fils creats per l'usuari.
    - La mida màxima de la taula d'enters (4 bytes) que pots emmagatzemar a la pila de 1 MB és de 262.144 elements, si tinguessis tota la pila disponible.
  - En UNIX / MacOS, el límit predeterminat de la pila és de **8MB**.

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

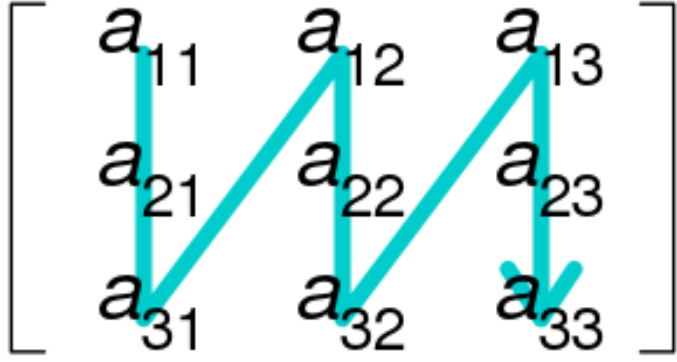
- Com s'emmagatzemen les matrius en memòria?
- Cada llenguatge de programació (o llibreria) té una manera particular d'emmagatzemar matrius en memòria
- És necessari saber com s'emmagatzema per poder fer recorreguts eficients.

Row-major order

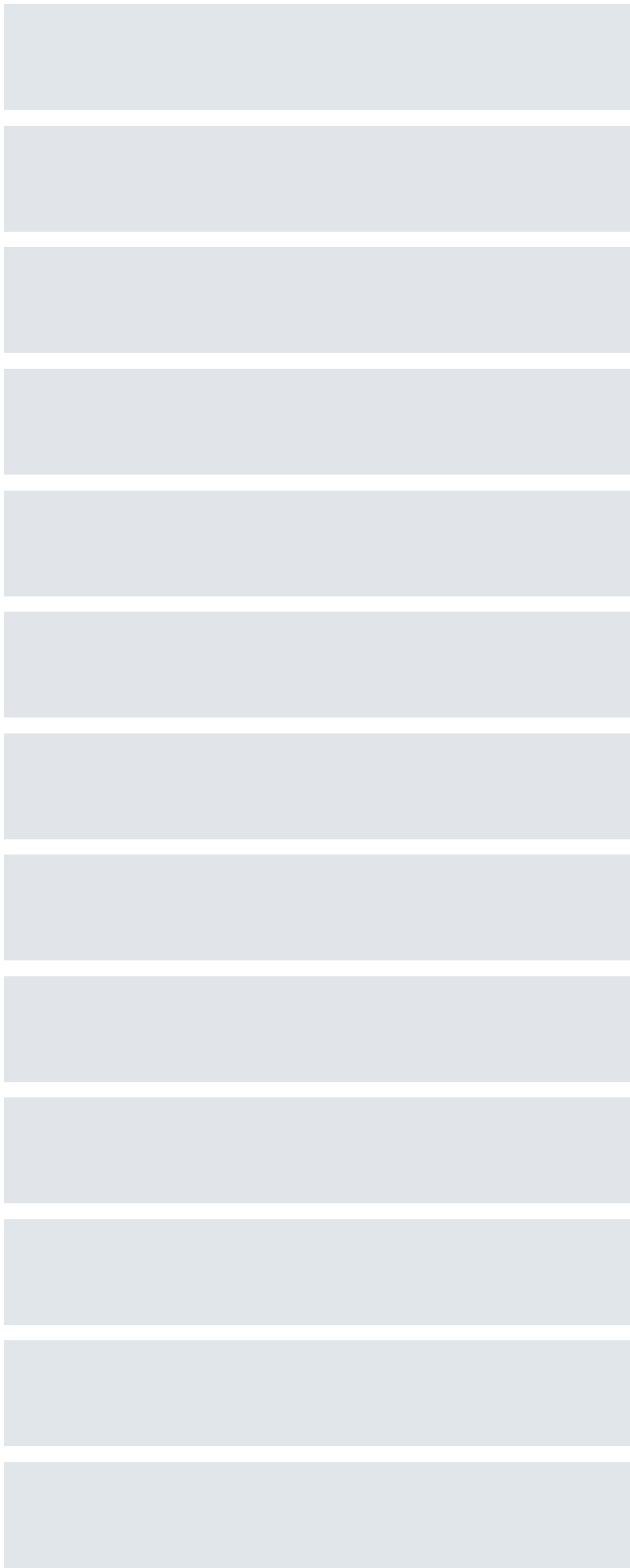


C / C++  
NumPy (Python)

Column-major order



Fortran  
Matlab  
R  
Julia



Memòria de 32 bits

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

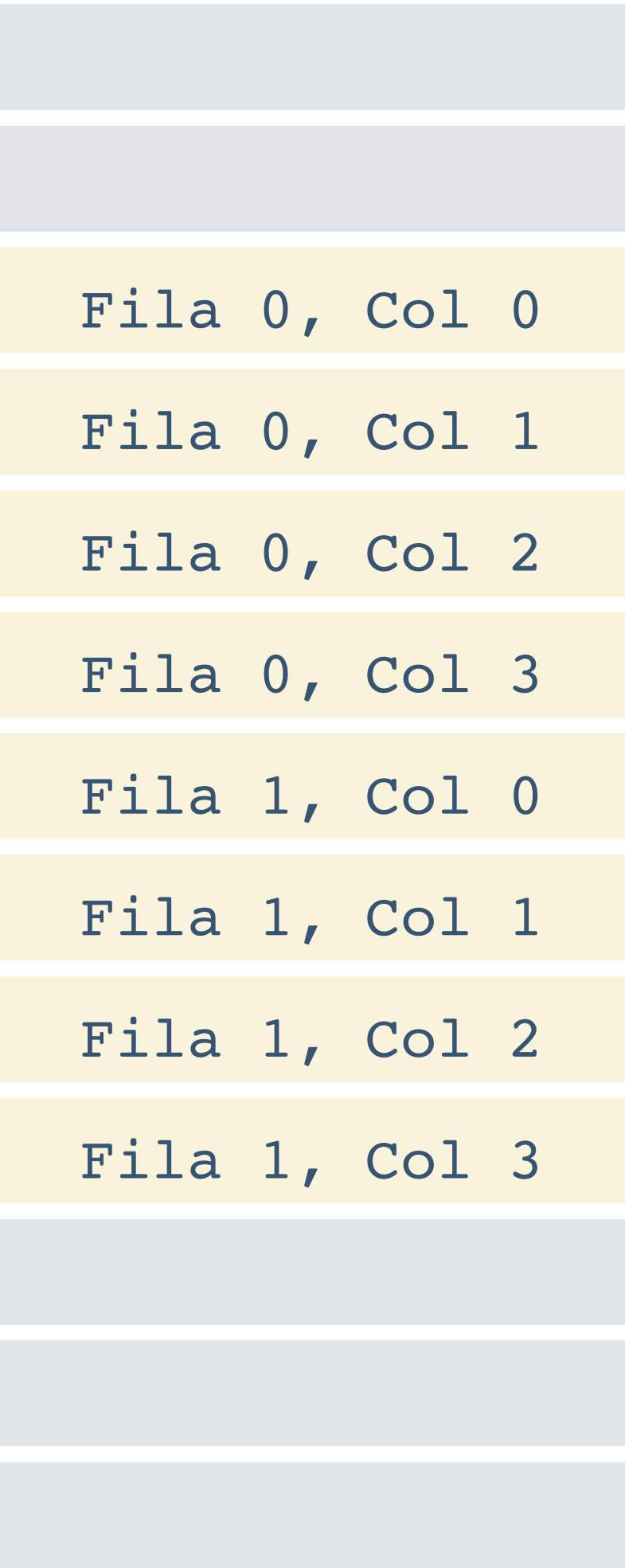
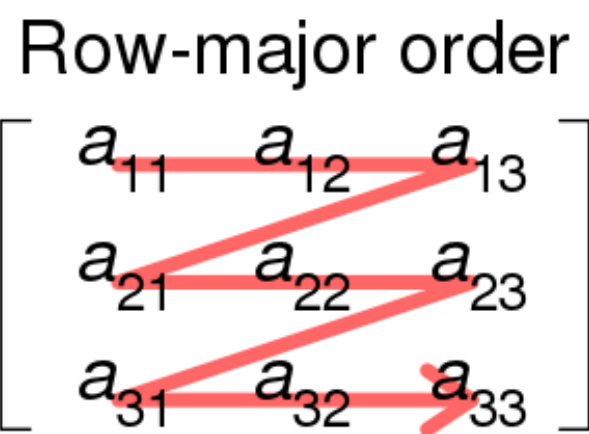
- En llenguatge C, quan definim una matriu:

```
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]
```

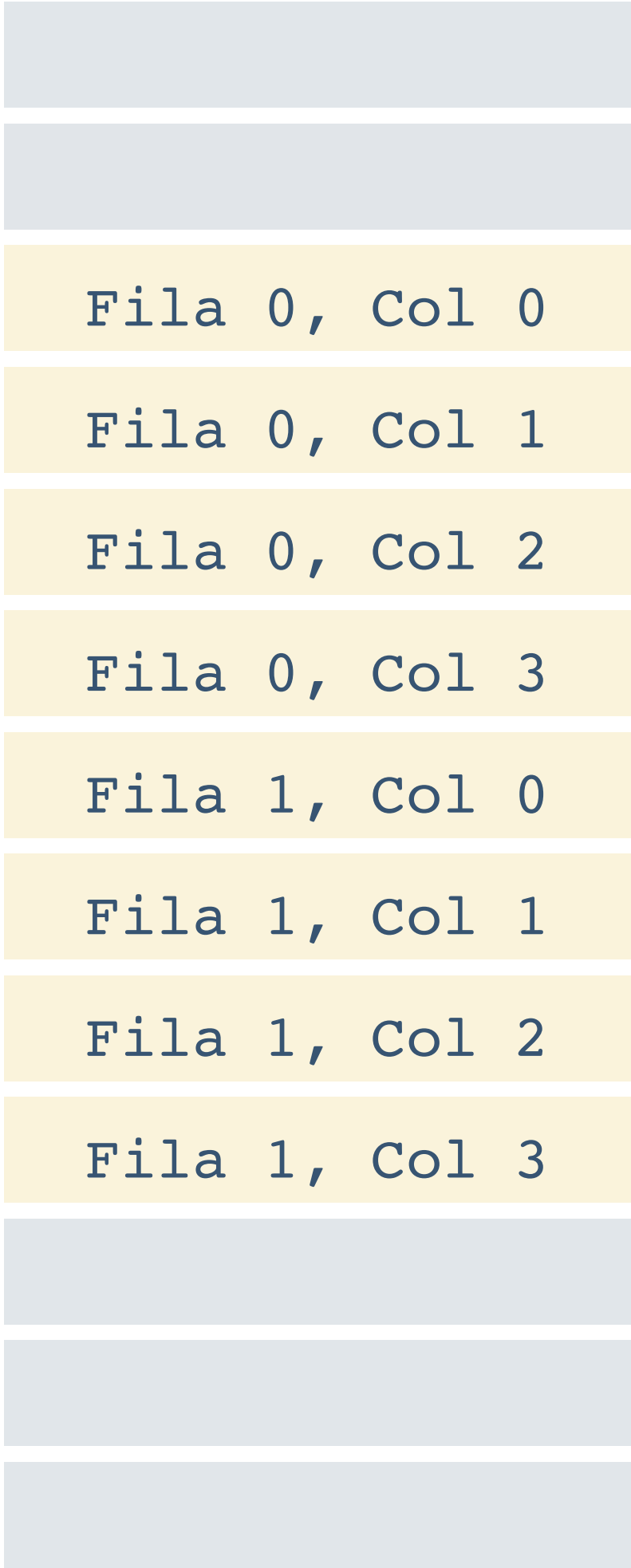
- O un recorregut per columnes

```
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.

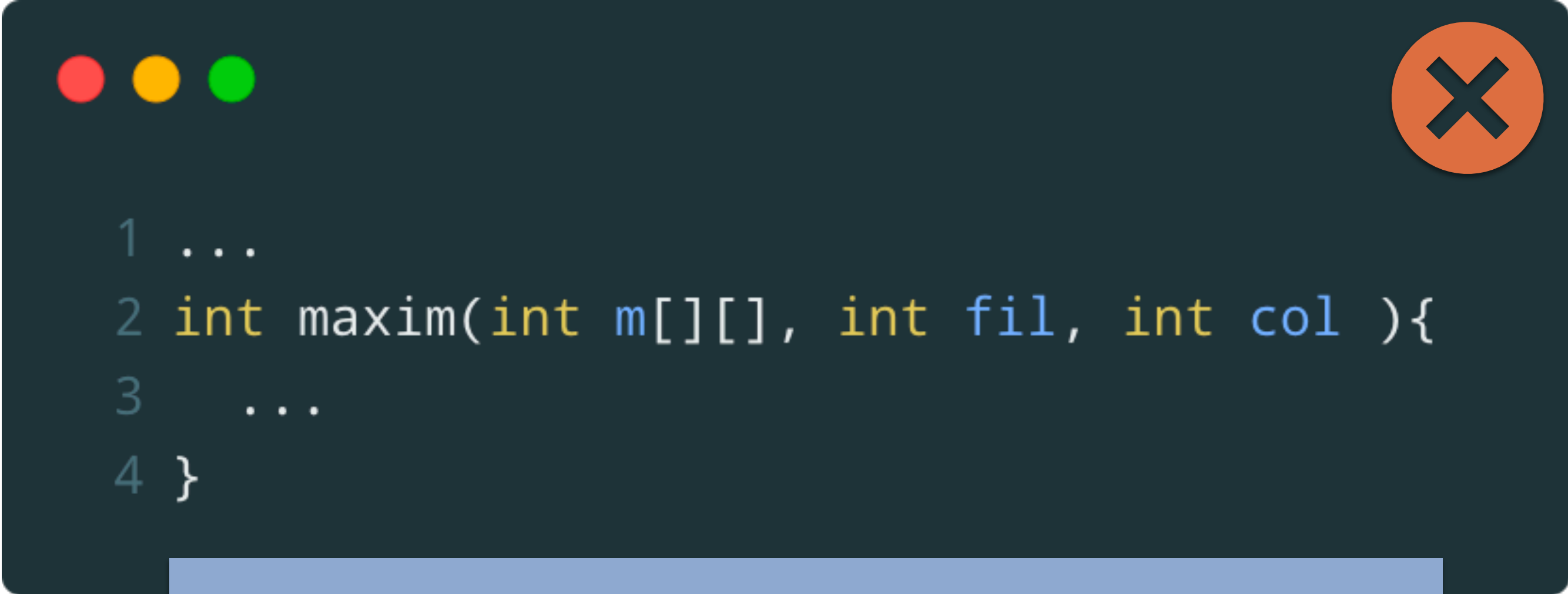
*Tècnicament, quan llegim de memòria llegim per blocs, per tant, fent un recorregut per files ja tenim el valor següent en el bloc carregat.*



Memòria de 32 bits



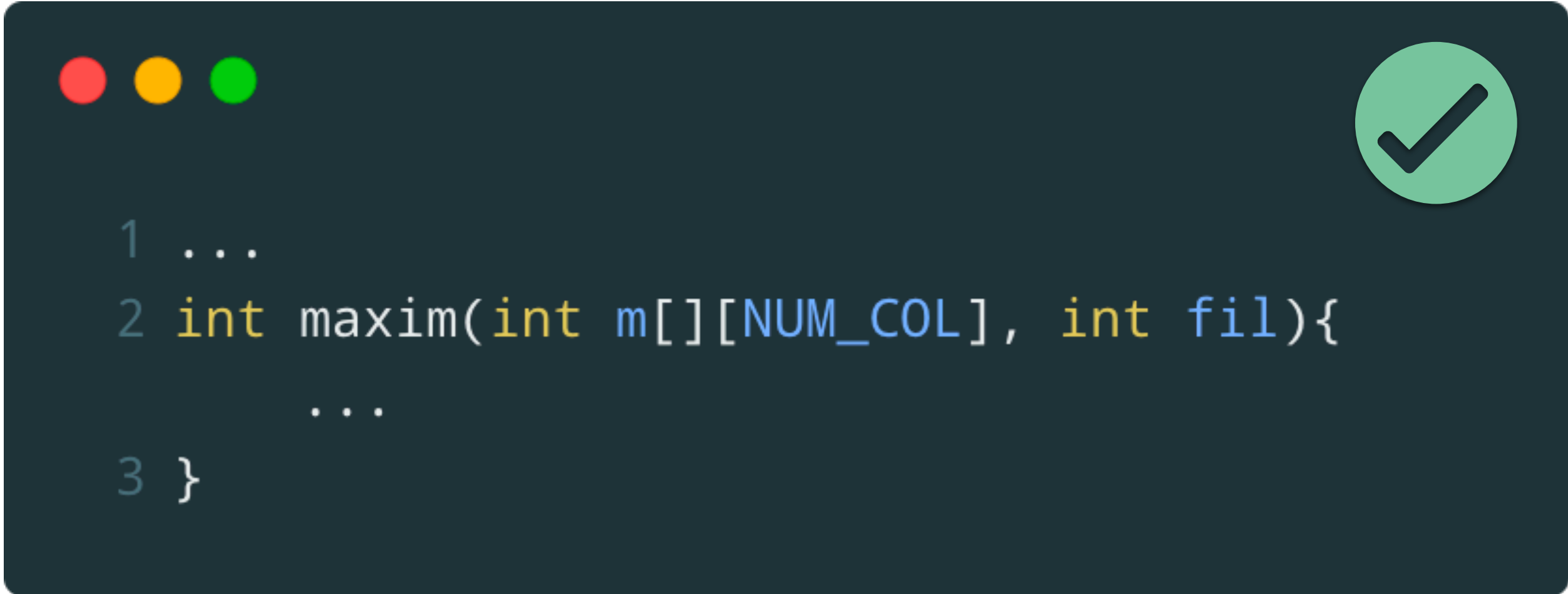
## Per què s'ha d'especificar el nombre de columnes quan passes una taula bidimensional per paràmetre?



```
1 ...  
2 int maxim(int m[], int fil, int col ){  
3     ...  
4 }
```

**error:** array has incomplete element type 'int []'

- Això fa referència a la capçalera de la funció. Quan *cridem* la funció, no s'ha de passar la mida, ni en el cas bidimensional ni en cap altre.

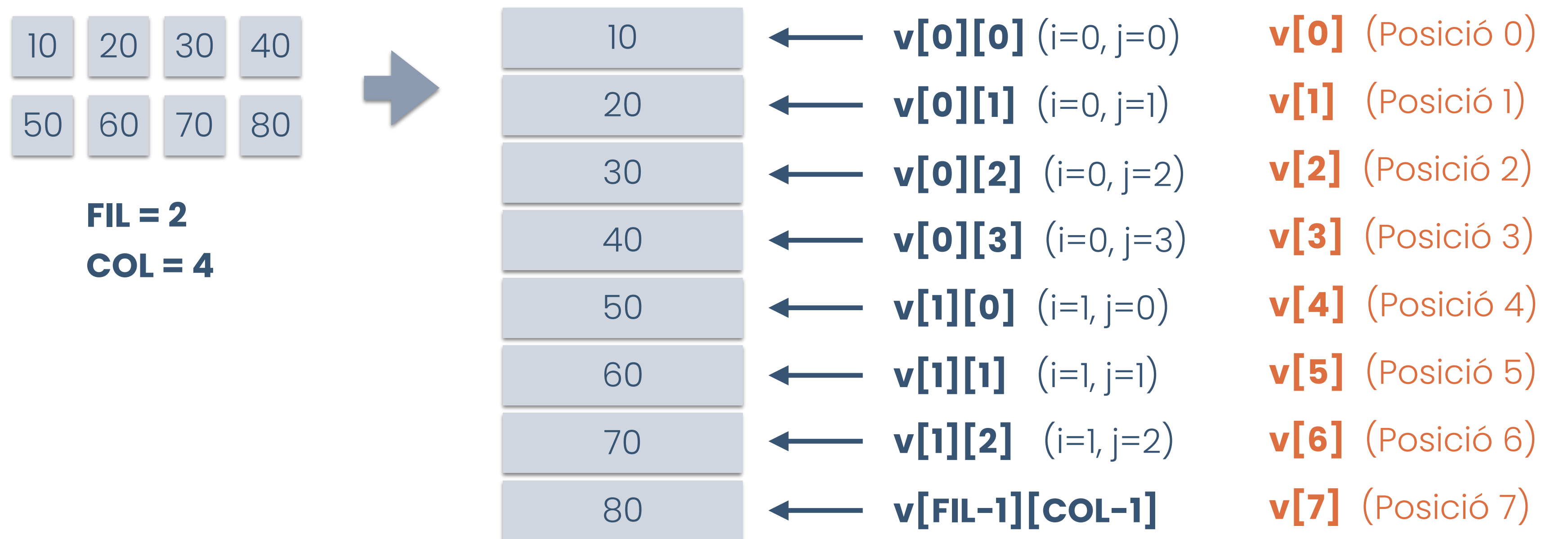


```
1 ...  
2 int maxim(int m[][NUM_COL], int fil){  
3     ...  
4 }
```



# Per què s'ha d'especificar el nombre de columnes quan passes una taula bidimensional per paràmetre?

- Acabem de veure com s'emmagatzema una matriu en C.



- Com hi podríem accedir si consideréssim que la nostra matriu és un vector?

**v[i][j]** → **v[ COL\*i + j ]**

Per fer aquest càlcul, necessitaríem saber quin valor té "COL".  
Per tant, el compilador necessita saber-ho també.  
Per això hem de especificar com a mínim el nombre de columnes.



## Potser algú de vosaltres s'està preguntant...

- "A mi em van dir que si especifico la mida d'un vector a la capçalera d'una funció, el compilador ho ignora". (És cert!)
- És a dir, que aquestes dues declaracions de la funció `processa_taula` són equivalents:

```
int processa_taula (int taula[]) {  
    ...  
}
```

```
int processa_taula (int taula[100]) {  
    ...  
}
```

- Aquestes declaracions no volen dir que la primera serveixi per a qualsevol taula, i la segona serveixi només per a taules de 100 posicions. El "100" és ignorat pel compilador.
- Tanmateix, la segona dimensió **no és ignorada pel compilador**, i l'hem d'incloure obligatòriament per a taules estàtiques de dues dimensions.

```
int processa_taula (int taula[][200]) {  
    ...  
}
```

*Això només s'aplica a taules estàtiques (que són les que hem vist fins ara). No aplica a les taules multidimensionals alocatades amb gestió dinàmica de la memòria (que veurem al tema 4)*

- Si la taula té tres o més dimensions, també hem d'especificar les dimensions tercera, quarta, etc...

# Cadenes: taules de caràcters

## Què són?

- No són res més que 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.
- Cada caràcter ocuparà una posició de la taula, i el sentinella també.



### Per què les cadenes han d'acabar amb el caràcter sentinella?

- Per saber on acaba la cadena, ja que no es guarda la longitud de la cadena (per motius d'eficiència)

### Què passa si em deixo el caràcter sentinella?

- Les funcions que treballen amb strings, de la llibreria *string.h* no funcionaran, perquè precisament esperen que un string acabi en \0. No en detectarien el final.

### Per què ' \0 ' i no qualsevol altre caràcter?

- El caràcter '\0' és "invisible" (non-printable)
- El codi ASCII del caràcter '\0' és 0. Per tant:
  - Ocupa menys
  - En llenguatge màquina, comprovar si una variable és zero és menys costós que comprovar si és un altre valor

### He de reservar espai per al caràcter sentinella?

- Òbviament. Un zero també ocupa espai.

# REGISTRES I TIPUS D'USUARI

# Registres

## Recordem la necessitat dels registres

- Fins ara, amb les taules, hem estat capaços d'emmagatzemar diverses variables del mateix tipus. A més, hi accedíem a través de la seva posició (índex).
- A vegades necessitarem emmagatzemar dades de **tipus diferents**
- Com ho hauríem hagut de fer fins ara?

- Per exemple, si vull guardar informació sobre un **llibre**:
  - Títol: taula de caràcters
  - Autor: taula de caràcters
  - Any de publicació: enter

- Puc crear un **registre** "llibre", on cadascuna de les variables en serà un **camp** (o **membre**)

L'accés a un registre es fa usant el nom del camp

Variable de tipus registre que es diu "llibre"

```
algorisme test_registres és
  const
    N := 50;
  fconst
  var
    llibre: registre
      autor: taula[N] de caràcter;
      títol: taula[N] de caràcter;
      any: enter;
    fregistre

  fvar
  inici
    llibre.autor := "J.K. Rowling";
    llibre.títol := "Harry Potter i la Pedra Filosofal";
    llibre.any := 1997;
  falgorisme
```



# Tipus d'usuari

## Per què volem crear un tipus?

- Què he de fer si necessito guardar informació de més d'un llibre? En l'exemple anterior, hauria de definir una nova variable "llibre2" amb el mateix tipus de camps

```
algorisme test_registres és
  const
    N := 50;
  fconst
  var
    llibre: registre
      autor: taula[N] de caràcter;
      títol: taula[N] de caràcter;
      any: enter;
    fregistre
    llibre2: registre
      autor: taula[N] de caràcter;
      títol: taula[N] de caràcter;
      any: enter;
    fregistre

  fvar
    ...
falgorisme
```



# Tipus d'usuari

## Per què volem crear un tipus?

- Què he de fer si necessito guardar informació de més d'un llibre? En l'exemple anterior, hauria de definir una nova variable "llibre2" amb el mateix tipus de camps
- Puc definir un nou tipus (**tipus d'usuari**)
- L'objectiu és poder fer crear diverses variables d'aquest nou tipus igual que ho puc fer amb els tipus bàsics:

```
var
  a, b, c: enter;
  c1, c2: caràcter;
  b1, b2: llibre;
fvar
```

Defineixo el  
registre llibre dins  
la secció "tipus",  
abans de  
l'algorisme i els  
procediments

- Com definir un tipus d'usuari:

```
tipus
  llibre: registre
    autor: taula[N] de caràcter;
    títol: taula[N] de caràcter;
    any: enter;
fregistre

ftipus
...
algorisme testllibres és
  var
    b1, b2: llibre;
  fvar
...
```

Després puc declarar tantes variables  
d'aquest tipus com vulgui



# Sobre els noms dels tipus

- Una confusió freqüent quan es comencen a fer servir tipus és no saber distingir el tipus de la variable:

```
tipus
  llibre: registre
        autor: taula[N] de caràcter;
        títol: taula[N] de caràcter;
        any: enter;
fregistre

ftipus
...
algorisme testllibres és
  var
    b1, b2: llibre;
  fvar
...

```

La paraula **llibre** és el nom del tipus

b1 i b2 fan referència a les variables de tipus **llibre**

- Ho sabeu distingir? Digues quina de les dues maneres és la correcta per accedir al camp `any` del llibre:
  - a) `b1.any := 1999;`
  - b) `llibre.any := 1999;`

- A *Fonaments de Programació I* se us va proposar la nomenclatura `*_t` per distingir els tipus de les variables.
- El sufix `_t` pretén indicar que la paraula fa referència a un **tipus**

```
tipus
  llibre_t: registre
        autor: taula[N] de caràcter;
        títol: taula[N] de caràcter;
        any: enter;
fregistre

ftipus
...
algorisme testllibres és
  var
    llibre: llibre_t;
  fvar
...

```

Variable

Tipus

## Sobre els noms dels tipus

- A *Fonaments de Programació I* se us va proposar la nomenclatura `*_t` per distingir els tipus de les variables.
- El sufix `_t` pretén indicar que la paraula fa referència a un **tipus**

```
tipus
  llibre_t: registre
    autor: taula[N] de caràcter;
    títol: taula[N] de caràcter;
    any: enter;
fregistre

ftipus
...
algorisme test_llibres és
  var
    llibre: llibre_t;
  fvar
  ...
```

A diagram with two blue arrows pointing downwards. The first arrow starts from the word 'llibre' in the line 'llibre: llibre\_t;' and points to the word 'Variable' in blue. The second arrow starts from the suffix '\_t' in 'llibre\_t' and points to the word 'Tipus' in teal.

- A aquesta assignatura, podeu trobar noms amb el sufix `_t` o bé els noms de tipus amb la primera lletra en majúscula. (Cosa que innegablement és una influència de la programació en Java)

```
tipus
  Llibre: registre
    autor: taula[N] de caràcter;
    títol: taula[N] de caràcter;
    any: enter;
fregistre

ftipus
...
algorisme test_llibres és
  var
    llibre: Llibre;
  fvar
  ...
```

A diagram with two teal arrows pointing downwards. The first arrow starts from the word 'llibre' in the line 'llibre: Llibre;' and points to the word 'Variable' in blue. The second arrow starts from the word 'Llibre' in the same line and points to the word 'Tipus' in teal.

- Heu de procurar entendre la diferència entre el tipus i les seves variables independentment dels noms, però usar una d'aquestes dues convencions us pot ajudar al començament.

# Tipus d'usuari

## Accés als registres

- Als registres s'hi accedeix amb el punt '.' i el nom del seu camp, no per cap índex. Es tracten com a variables normals.

...

```
b1.autor := "J.K. Rowling";  
b1.titol := "Harry Potter i la Pedra  
Filosofal";  
b1.any := 1997;
```

```
b2.autor := "J.K. Rowling";  
b2.titol := "Harry Potter i la Cambra  
Secreta";  
b2.any := 1998;
```

...

# Tipus d'usuari

## Comparacions entre registres

- Tal com passava amb les taules, no podem fer comparacions entre registres.

```
$ No es poden comparar registres
b1 = b2;
b1 < b2;
$ Sí que es poden comparar els seus camps
(si són tipus simples)
iguals := b1.any == b2.any;
```

- Per fer comparacions haurem de crear procediments



## El motiu pel qual no podem comparar registres és el mateix pel qual no podem comparar taules?

- No, no és el mateix motiu.
- En les taules, es pot fer servir l'operador "==", simplement el que passa és que en compara les adreces de memòria, i no ens serveix per a comparar-ne continguts
- En canvi, als registres, l'operador "==" no està definit segons l'estàndard de C. Això és perquè:
  - Els registres s'emmagatzemen a memòria deixant "forats" entre els camps. Per tant, una comparació de tots els continguts de la memòria podria donar resultats diferents fins i tot amb dos registres que tenen els mateixos camps i els mateixos valors.
  - Per aquest motiu, es va decidir no implementar els operadors de comparació amb registres.

# Tipus d'usuari

## Comparacions entre registres

- Tal com passava amb les taules, **no podem fer comparacions entre registres**.

```
$ No es poden comparar registres
b1 = b2;
b1 < b2;
$ Sí que es poden comparar els seus camps
(si són tipus simples)
iguals := b1.any == b2.any;
```

- Per fer comparacions haurem de crear procediments

## Assignacions entre registres

- En canvi, al contrari de les taules, **sí que es poden assignar registres**, i se'n copien els continguts dels seus camps.

```
$ Sí que es poden assignar registres
b1 := b2;
$ Òbviament també es pot fer de manera
manual, camp a camp
b1.any := b2.any;
```

# Taules de registres

## Com guardo molta informació?

- De la mateixa manera que es pot crear una taula d'enters, reals..., podem crear una taula del nostre tipus de registres:

```
tipus
    llibre_t: registre
        autor: taula[N] de caràcter;
        títol: taula[N] de caràcter;
        any: enter;
    fregistre

ftipus

...

const
    NUM_LLIB := 7;
fconst

var
    saga: taula[NUM_LLIB] de llibre_t;
fvar

...
```

```
...
inici

    saga[0].autor := "J.K. Rowling";
    saga[0].titol := "Harry Potter i la Pedra
Filosofal";
    saga[0].any := 1997;

    saga[1].autor := "J.K. Rowling";
    saga[1].titol := "Harry Potter i la Cambra
Secreta";
    saga[1].any := 1998;
...
```



# Exercicis

## 1. **Saber si hi ha un zero** (nivell: molt fàcil)

Donada una taula d'enters de 100 posicions, que pressuposarem que la podem omplir de contingut amb la crida de l'acció `omplir_taula(t: taula d'enter)`, escriure un codi que determini si a la taula hi ha algun zero, i n'imprimeixi per pantalla aquest resultat.

## 2. **En una seqüència, comptar enters** (nivell: molt fàcil)

En una seqüència d'enters positius llegida de teclat, acabada en `-1`, escriure un programa que compti quants nombres parells ens han introduït

## 3. **(Amb taules) Xifres repetides** (nivell: normal)

Escriure un programa que demani un nombre a l'usuari i imprimeixi un missatge dient si aquest té xifres repetides o no. Per exemple, 1234 no té xifres repetides, en canvi 1232 sí.

Compte que el nombre que t'introdueix l'usuari és un nombre enter, no pas una taula!

## 4. **Xifres repetides (versió 2)** (nivell: normal)

Modifica el programa anterior de manera que, a més de dir si el nombre té xifres repetides, et digui **quines** són aquestes xifres. A més, fes que el teu programa només et digui cada xifra que es repeteix una sola vegada.

E.g. amb el número 12342424, hauria de dir:

"La xifra 2 està repetida"

"La xifra 4 està repetida" (un sol cop cadascuna)

## 5. **Valors no repetits** (nivell: una miiiica més alt)

Donada una seqüència de nombres introduïda per l'usuari i finalitzada amb un `-1`, guarda el valors en una taula, però sense duplicats.

Per exemple, si l'entrada és:

4 52 99 21 6 3 99 4 2 -1 ,

La taula hauria de guardar:

4 52 99 21 6 3 2.