

# **Quadern de laboratori Estructura de Computadors**

Emilio Castillo  
José María Cela  
Montse Fernández  
David López  
Joan Manuel Parcerisa  
Angel Toribio  
Rubèn Tous  
Jordi Tubella  
Gladys Utrera  
FIBerHub

Departament d'Arquitectura de Computadors  
Facultat d'Informàtica de Barcelona  
Quadrimestre de Primavera - Curs 2014/15



Aquest document es troba sota una llicència Creative Commons

# Licencia Creative Commons

Esta obra está bajo una licencia Reconocimiento-No comercial-Compartir bajo la misma licencia 2.5 España de Creative Commons. Para ver una copia de esta licencia, visite

<http://creativecommons.org/licenses/by-nc-sa/2.5/es/>

o envíe una carta a

Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

Usted es libre de:

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

Bajo las condiciones siguientes:

- **Reconocimiento.** Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- **No comercial.** No puede utilizar esta obra para fines comerciales.
- **Compartir bajo la misma licencia.** Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.
- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Advertencia: Este resumen no es una licencia. Es simplemente una referencia práctica para entender el Texto Legal (la licencia completa).

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.

# Sessió 3: Tipus de dades estructurats

**Objectiu:** Entendre com s'emmagatzemen en memòria les matrius declarades en el llenguatge C, i aprendre i posar en pràctica l'accés a variables de tipus matriu.

## Lectura prèvia

### Matrius

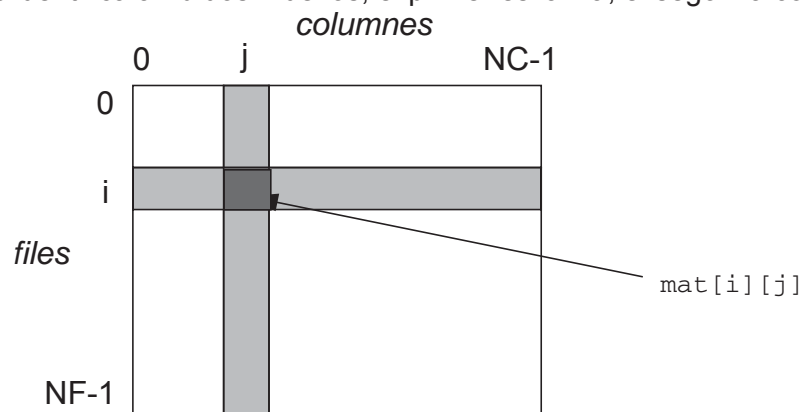
Una matriu és una agrupació multidimensional d'elements de tipus homogenis, els quals s'identifiquen per un índex en cada dimensió. Una cas particular són les matrius de dues dimensions: un conjunt bidimensional de  $NF \times NC$  elements, on  $NF$  és el nombre de files i  $NC$  el de columnes. La declaració en C d'una matriu d'enters és:

```
int mat[NF][NC]; /* NF i NC han de ser constants */
```

i hi podem inicialitzar els elements, com per exemple:

```
int mat[2][3] = {{-1, 2, 4}, {0, 5, -2}};
```

Cada element s'identifica amb dos índexos, el primer és la fila, el segon la columna:



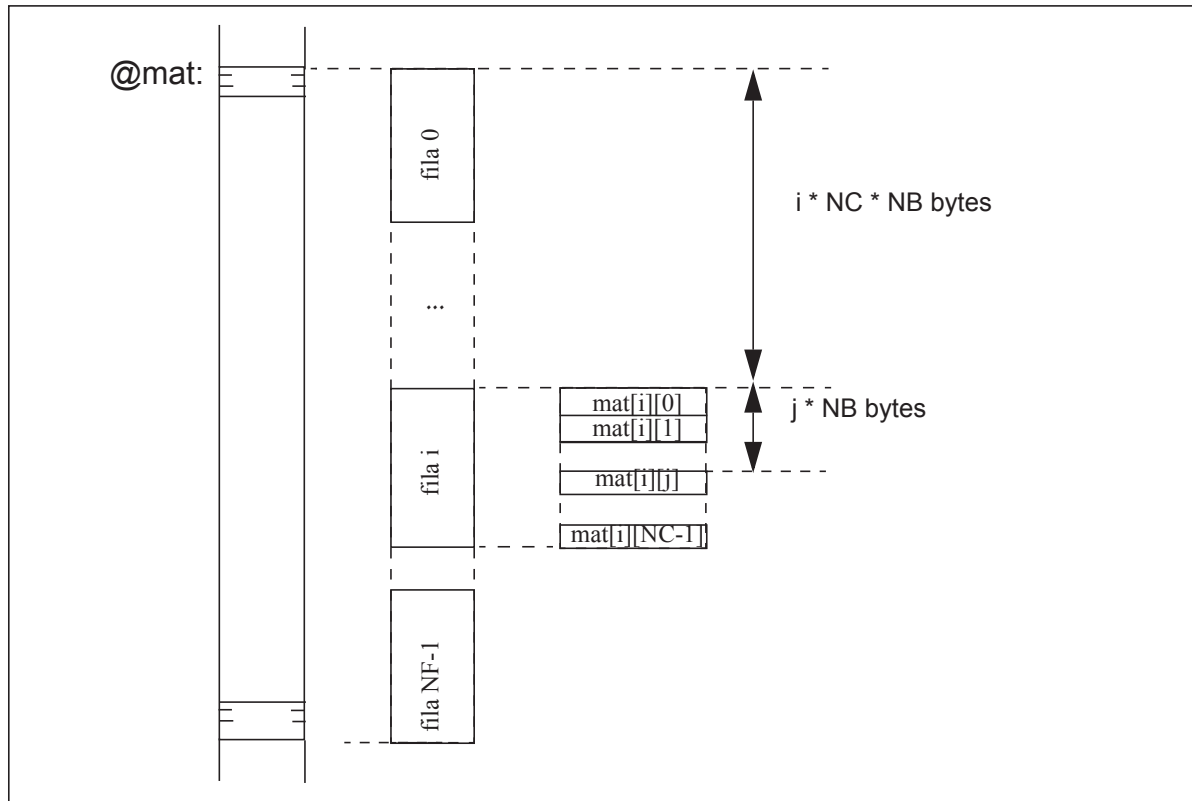
En C els elements s'emmagatzemen en memòria en posicions consecutives per files, i en cada fila ordenats per columnes.

En MIPS els elements també es guarden en posicions consecutives a partir de l'adreça *mat*, respectant les regles d'alineació dels elements (*mat* es guardarà doncs en una adreça múltiple de 4). En MIPS serà:

```
.data
.align 2
mat:  .space NF*NC*4
o bé:
mat:  .word -1,2,4,0,5,-2
```

Suposant que `mat` és una matriu de `NF` files i `NC` columnes d'elements de mida `NB` bytes, per accedir a un element qualsevol `mat[i][j]` ho farem saltant `i` files completes, i després `j` elements, a partir de l'adreça inicial `mat` (figura 3.1). L'adreça es calcularà de la següent manera (`NB` = núm bytes per element):

$$\text{@mat}[i][j] = \text{@mat} + (i * \text{NC} + j) * \text{NB} = \text{@mat} + i * \text{NC} * \text{NB} + j * \text{NB}$$



**Figura 3.1:** : Emmagatzematge de matrius per files, i accés a l'element `mat[i][j]`

## Enunciats de la sessió

### Activitat 3.A: Declaració de matrius

Completa el següent exercici:

**Exercici 3.1:** Donada la següent declaració de variables globals en C, tradueix-la a ensamblador MIPS. Recorda que en C els elements d'una matriu es guarden per files. Utilitza la directiva `.align` per garantir l'alineació dels elements, allà on calgui.

```
int mat1[5][6];
char mat2[3][5];
long long mat3[2][2];
int mat4[2][3] = {{2, 3, 1}, {2, 4, 3}};
```

```
data:

    .align 2
mat1: .space 120
mat2: .space 15
    .align 3
mat3: .space 32
mat4: .word 2, 3, 1, 2, 4, 3
```

### Activitat 3.B: Accés als elements d'una matriu

Completa el següent exercici:

**Exercici 3.2:** Calcula l'adreça de memòria de cada un dels següents accessos a les matrius declarades en l'apartat anterior. Pots deixar la resposta en funció de les adreces *mat1*, *mat2*, *mat3* i *mat4*.

@mat1[4][3] =	@mat1 + (4*6+3)*4
@mat2[2][4] =	@mat2 + (2*5+4)
@mat3[1][0] =	@mat3 + (2*1)*8
@mat4[0][2] =	@mat4 + (2)*4

Aquesta activitat consistirà a traduir el següent programa (Figura 3.2) a ensamblador MIPS. No fa res d'especial interès, però ens permetrà practicar l'accés a elements d'una matriu.

```
int mat1[5][6];
int mat4[2][3] = {{2, 3, 1}, {2, 4, 3}};
int col = 2;

main()
{
    mat1[4][3] = subr(mat4, mat4[0][2], col);
    mat1[0][0] = subr(mat4, 1, 1);
}

int subr(int x[][3], int i, int j)
{
    mat1[j][5] = x[i][j];
    return i;
}
```

**Figura 3.2:** Programa que fa diversos accessos aleatoris a matrius

Abans de continuar, completa els exercicis 3.3 i 3.4:

**Exercici 3.3:** Tradueix a llenguatge ensamblador MIPS les declaracions de variables globals i la funció *main* de la Figura 3.2. Recorda que aquesta s'ha de programar com una subrutina, seguint totes les regles estudiades (p. ex., s'ha de preservar el registre *\$ra* si el *main* crida altres subrutines). Observa que la declaració de les variables globals *mat1* i *mat4* són les mateixes que en l'exercici 3.1, i que les adreces dels elements *mat1[4][3]* i *mat4[0][2]* són les que ja has calculat a l'exercici 3.2.

```
.data
mat1: .space 120
mat4: .word 2, 3, 1, 2, 4, 3
col:  .word 2
```

```
.text
.globl main
```

```
main:
    addiu $sp, $sp, -4
    sw $ra, 0($sp)

    la $a0, mat4
    lw $a1, 8($a0)
    la $a2, col
    lw $a2, 0($a2)
    jal subr
```

```
la $t0, mat1
sw $v0, 108($t0)
la $a0, mat4
li $a1, 1
li $a2, 1
jal subr

la $t0, mat1
sw $v0, 0($t0)

lw $ra, 0($sp)
addiu $sp, $sp, 4
jr $ra
```

**Exercici 3.4:** Tradueix a assembleador MIPS la subrutina *subr* de la Figura 3.2

```

subr:
    la $t0, mat1
    li $t1, 24
    multu $a2, $t1
    mflo $t1
    addu $t0, $t0, $t1
    li $t1, 6
    mult $a1, $t1
    mflo $t1
    addu $t1, $t1, $a2
    sll $t1, $t1, 2
    addu $t1, $a0, $t1
    lw $t1, -12($t1)
    sw $t1, 20($t0)
    move $v0, $a1

    jr $ra

```

**Comprovació pràctica**

Copieu el codi dels exercicis 3.4 i 3.3 al fitxer *s3b.s*. Executeu-lo amb el MARS i comproveu en finalitzar el programa que l'element `mat1[0][0]=1`, que `mat1[1][5]=4`, que `mat1[2][5]=3` i que `mat1[4][3]=1` (per localitzar-los a la vista de dades del MARS haureu de calcular a mà la seva adreça tenint en compte que la variable *mat1* s'emmagatzema a partir de la posició 0x10010000 de memòria).

A continuació, utilitzant el depurador de MARS, poseu un breakpoint a l'inici de la subrutina *subr*, executeu el programa fins a aquest punt, i digueu quin és el valor (en hexadecimal) dels següents registres:

\$a0 =	0x10010078
\$a1 =	0x00000001
\$a2 =	0x00000002
\$ra =	0x00400030

## Activitat 3.C: Accés seqüencial a la columna d'una matriu

Completa el següent exercici abans de continuar:

**Exercici 3.5:** Donada la següent declaració de la matriu *mat*, de 4 files per 6 columnes, de nombres enters:

```
int mat[4][6];
```

Escriu la fórmula per calcular l'adreça de l'element *mat[i][2]*, en funció de l'adreça de *mat* i del valor enter *i*:

@mat[i][2] =	@mat + (6i+2)*4
--------------	-----------------

Fent servir aquesta fórmula, calcula la distància en bytes (stride) entre les adreces de dos elements consecutius d'una mateixa columna:

@mat[i+1][2] - @mat[i][2] =	$  \begin{aligned}  & (@mat + (6*(i+1)+2)*4) - (@mat + (6i+2)*4) = \\  & = (6i+8)*4 - 24i - 8 = \\  & = 24i + 32 - 24i - 8 = 24 \text{ bytes}  \end{aligned}  $
-----------------------------	---

Veient les respostes de l'exercici 3.5, observeu que, donats dos elements consecutius d'una columna, l'adreça del segon element s'obté sumant una quantitat constant a l'adreça de l'element anterior. En general, la distància entre dos elements consecutius d'un vector o d'una fila d'una matriu també és constant. A aquest valor constant se l'acostuma a anomenar *stride*, i és el resultat que has calculat a l'exercici anterior. Quan recorrem vectors o matrius utilitzant aquesta propietat diem que estem fent un "accés seqüencial" als seus elements, i seguim els següents 3 passos:

1. Al principi, inicialitzar un punter (un registre) amb l'adreça del primer element a recórrer.
2. Accedir a cada element fent servir sempre aquest punter.
3. Just a continuació de cada accés, incrementar el punter tants bytes com hi hagi de diferència entre un element i el següent (l'*stride*, el mateix que has calculat a l'exercici 3.5).



A continuació considerarem el següent programa en C, on la funció *suma\_col* calcula la suma dels elements de la columna 2 de la matriu *m* que rep com a paràmetre:

```
int mat[4][6] = { {0, 0, 2, 0, 0, 0},
                  {0, 0, 4, 0, 0, 0},
                  {0, 0, 6, 0, 0, 0},
                  {0, 0, 8, 0, 0, 0} };

int resultat;

main()
{
    resultat = suma_col(mat);
}

int suma_col(int m[][6])
{
    int i, suma = 0;

    for(i = 0; i < 4; i++)
        suma += m[i][2];

    return suma;
}
```

**Figura 3.3:** Suma de la columna 2 de la matriu amb accés aleatori.

Podem reescriure en C la funció *suma\_col* aplicant la tècnica d'accés seqüencial:

```
int suma_col(int m[][6])
{
    int i, suma = 0;
    int *p;

    p = &m[0][2];                /* inicialitzar el punter */
    for(i = 0; i < 4; i++) {
        suma += *p;              /* accés a m usant el punter */
        p += 6;                  /* incrementar el punter */
    }

    return suma;
}
```

**Figura 3.4:** Suma de la columna 2 de la matriu amb accés seqüencial.

La matriu *m* de la funció *suma\_col* és igual a la matriu *mat* que has vist a l'exercici 3.5. Fixa't però en la diferència que hi ha entre el valor de l'*stride* que has calculat a l'exercici 3.5 i el valor amb què s'incrementa el punter *p* en la Figura 3.4. Això és degut a l'aritmètica de punters en C: un increment d'una unitat en un punter suposa sumar-li, en assembleador, la mida (en bytes) de l'element al qual apunta. En el nostre cas, la mida de l'element és 4 bytes, ja que *p* apunta a un element de tipus *int*. Per tant, el valor que apareix en el codi en C s'haurà de multiplicar per 4 en la traducció a assembleador.

Completa l'exercici 3.6 abans de continuar:

**Exercici 3.6:** Tradueix a MIPS el programa de la Figura 3.3 usant la tècnica d'accés seqüencial per recórrer la matriu (versió que apareix a la Figura 3.4). Recorda que la funció *main* s'ha de programar com una subrutina, seguint les regles pertinents.

<pre> .data mat: .word 0,0,2,0,0,0      .word 0,0,4,0,0,0      .word 0,0,6,0,0,0      .word 0,0,8,0,0,0  resultat: .word 0  .text .globl main main:     addiu \$sp, \$sp, -4     sw \$ra, 0(\$sp)      la \$a0, mat     jal suma_col     la \$t0, resultat     sw \$v0, 0(\$t0)      lw \$ra, 0(\$sp)     addiu \$sp, \$sp, 4     jr \$ra </pre>	<pre> suma_col:     li \$t0, 0     move \$t1, \$a0     addiu \$t1, \$t1, 8      li \$t2, 0     li \$t3, 4  for: beq \$t2, \$t3, end     lw \$t4, 0(\$t1)     addu \$t0, \$t0, \$t4     addiu \$t1, \$t1, 24      addiu \$t2, \$t2, 1     b for  end: move \$v0, \$t0     jr \$ra </pre>
--	---

### Comprovació pràctica

Copieu el codi de l'exercici 3.6 anterior en el fitxer *s3c.s*. Verifiqueu que després d'executar el programa, la variable global *resultat* val 20. Feu també la següent comprovació: reinicieu el programa (tecla F12); poseu un punt d'aturada a la subrutina *suma\_col*, a la instrucció següent del *lw* que accedeix a la matriu; i observeu com, a cada pulsació de la tecla F5 (Go), el *lw* va carregant al registre destí els successius elements de la columna 2 de *mat*: 2, 4, 6, 8.