

Memoria Estática

Punteros, Vectores y Matrices

Iván Arcuschin Moreno
(diapos de David González Márquez)

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

20-08-2019

Repaso de punteros

- ▶ Es una variable que referencia una posición de la memoria. (ejemplo: una variable cuyo valor es una dirección de memoria)
- ▶ Tiene un tipo y un nombre.
- ▶ Almacena una dirección de memoria.
- ▶ Sirve para referenciar una posición de memoria.
- ▶ Operadores:
 - & → Da como resultado la dirección de memoria de una variable.
 - * → Da como resultado el valor apuntado por un puntero. (Además de ser el indicador del tipo puntero)

Repaso de punteros

Ejemplos:

- ▶ `int *pepe`

Repaso de punteros

Ejemplos:

- ▶ `int *pepe`

Declara un puntero de tipo entero con nombre *pepe*.

Repaso de punteros

Ejemplos:

- ▶ `int *pepe`

Declara un puntero de tipo entero con nombre *pepe*.

- ▶ `int x = 5`
`pepe = &x`

Repaso de punteros

Ejemplos:

- ▶ `int *pepe`

Declara un puntero de tipo entero con nombre *pepe*.

- ▶ `int x = 5`

`pepe = &x`

Guarda en el puntero *pepe* la dirección de *x*.

Se dice que *pepe* apunta a *x*.

Repaso de punteros

Ejemplos:

- ▶ `int *pepe`

Declara un puntero de tipo entero con nombre *pepe*.

- ▶ `int x = 5`

`pepe = &x`

Guarda en el puntero *pepe* la dirección de *x*.

Se dice que *pepe* apunta a *x*.

- ▶ `*pepe = 8`

Repaso de punteros

Ejemplos:

- ▶ `int *pepe`

Declara un puntero de tipo entero con nombre *pepe*.

- ▶ `int x = 5`

`pepe = &x`

Guarda en el puntero *pepe* la dirección de *x*.

Se dice que *pepe* apunta a *x*.

- ▶ `*pepe = 8`

Guarda 8 en la posición apuntada por el puntero *pepe*.

Repaso de punteros

Ejemplos:

- ▶ `int *pepe`

Declara un puntero de tipo entero con nombre *pepe*.

- ▶ `int x = 5`

`pepe = &x`

Guarda en el puntero *pepe* la dirección de *x*.

Se dice que *pepe* apunta a *x*.

- ▶ `*pepe = 8`

Guarda 8 en la posición apuntada por el puntero *pepe*.

- ▶ `int y`

`y = *pepe`

Guarda en *y* el valor apuntado por *pepe*.

Repaso de punteros

Ejemplos:

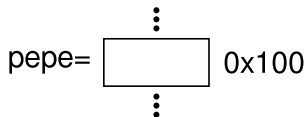
1. `int *pepe`

1.

2. `int x = 5`
`pepe = &x`

3. `*pepe = 8`

4. `int y`
`y = *pepe`



Repaso de punteros

Ejemplos:

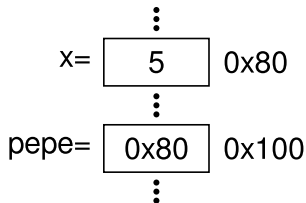
1. `int *pepe`

2. `int x = 5`
`pepe = &x`

3. `*pepe = 8`

4. `int y`
`y = *pepe`

2.



Repaso de punteros

Ejemplos:

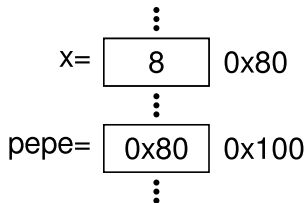
1. `int *pepe`

2. `int x = 5`
`pepe = &x`

3. `*pepe = 8`

4. `int y`
`y = *pepe`

3.



Repaso de punteros

Ejemplos:

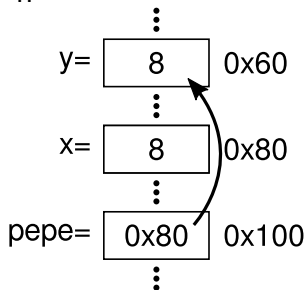
1. `int *pepe`

2. `int x = 5`
`pepe = &x`

3. `*pepe = 8`

4. `int y`
`y = *pepe`

4.



Vectores

Un vector o arreglo es una secuencia ordenada de elementos consecutivos en memoria de un tamaño fijo.

A_0	A_1	A_2	\dots	A_{n-3}	A_{n-2}	A_{n-1}
-------	-------	-------	---------	-----------	-----------	-----------

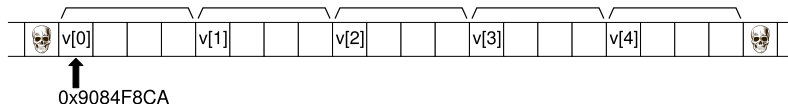
Vectores

Declaremos un vector v en C:

```
int v[5];
```

¿Cómo está guardado en memoria?

Como 5 enteros (*doublewords* / 4 bytes) **consecutivos**:

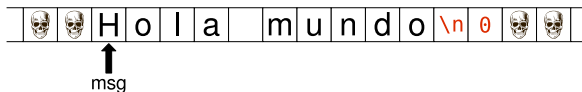


Vectores

Si rememoramos el ejemplo de la primera clase:

```
section .rodata:  
msg: DB 'Hola mundo', 10, 0  
largo: EQU $-msg-1
```

msg es una etiqueta que, vista como un puntero, es un vector de caracteres almacenados de la siguiente manera:



msg es un `char*`, es como si en C hiciéramos:

```
char msg[11] = "Hola mundo\n";
```


Vectores

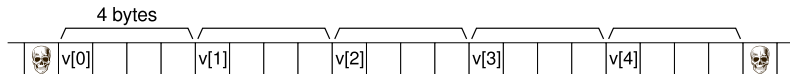
Dado:

```
int v[5];
```

Si suponemos que el primer elemento se encuentra almacenado en la dirección 0x200 de memoria

¿cómo se realiza la siguiente asignación?

```
v[2] = 8;
```



Vectores

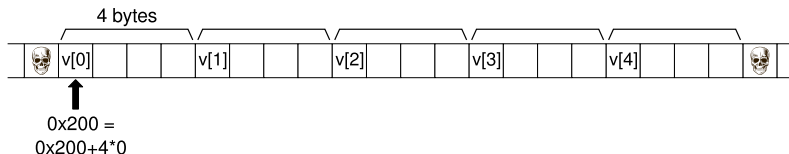
Dado:

```
int v[5];
```

Si suponemos que el primer elemento se encuentra almacenado en la dirección 0x200 de memoria

¿cómo se realiza la siguiente asignación?

```
v[2] = 8;
```



Vectores

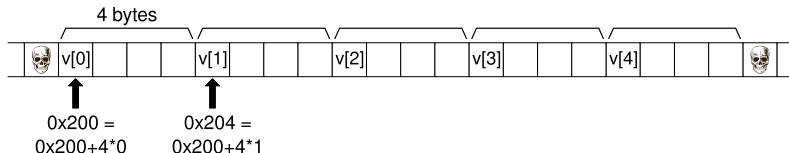
Dado:

```
int v[5];
```

Si suponemos que el primer elemento se encuentra almacenado en la dirección 0x200 de memoria

¿cómo se realiza la siguiente asignación?

```
v[2] = 8;
```



Vectores

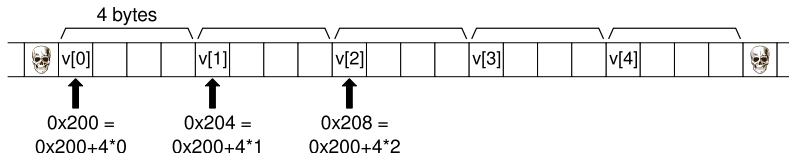
Dado:

```
int v[5];
```

Si suponemos que el primer elemento se encuentra almacenado en la dirección 0x200 de memoria

¿cómo se realiza la siguiente asignación?

```
v[2] = 8;
```



Vectores

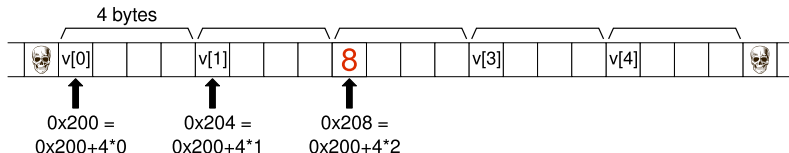
Dado:

```
int v[5];
```

Si suponemos que el primer elemento se encuentra almacenado en la dirección 0x200 de memoria

¿cómo se realiza la siguiente asignación?

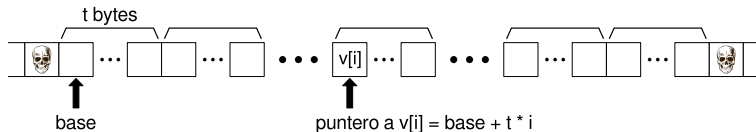
```
v[2] = 8;
```



Vectores

En general:

puntero al inicio + tamaño del dato * índice del elemento



En Intel:

Si el tamaño de los elementos es un valor válido como escala,

[<base> + <índice> * <escala>] $\xrightarrow{\text{ejemplo}}$ [rax+rbx*4]

Vectores y Punteros

Si tenemos:

```
int v[5];
```

`v` es un puntero al primer elemento del vector.

Luego, vale en C:

```
int *p_v = v; ← tomo el puntero al vector
```

ó

```
int *p_v = &v[0]; ← tomo la dirección del primer elemento
```

Matrices

- ▶ Se representan en memoria como un vector de vectores.
- ▶ Si la matriz tiene dimensión $M \times N$ entonces sabemos que está formada por M vectores de N elementos cada uno.
- ▶ Las matrices se almacenan por filas.

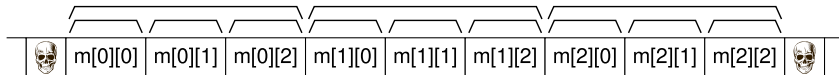
Matrices

Ejemplo

Si tenemos M , una matriz de enteros de 3×3 :

$m[0][0]$	$m[0][1]$	$m[0][2]$
$m[1][0]$	$m[1][1]$	$m[1][2]$
$m[2][0]$	$m[2][1]$	$m[2][2]$

En memoria se representa:



Matrices

El primer elemento de M se almacena en la dirección 0x300.
Queremos asignar un valor 7 en $M[2, 1]$, luego en C:

```
int m[3][3];  
m[2][1] = 7;
```

¿Y en ensamblador?

4 bytes

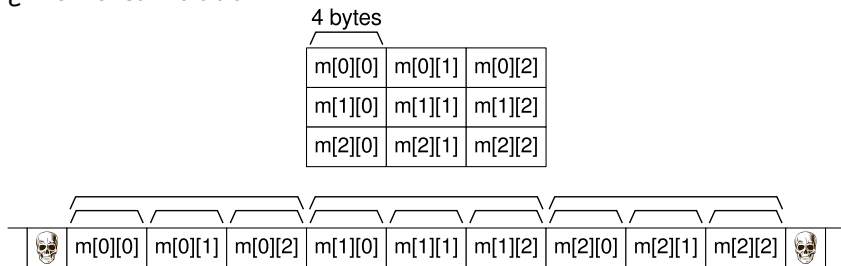
m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]
m[2][0]	m[2][1]	m[2][2]

Matrices

El primer elemento de M se almacena en la dirección 0x300.
Queremos asignar un valor 7 en $M[2, 1]$, luego en C:

```
int m[3][3];  
m[2][1] = 7;
```

¿Y en ensamblador?



Matrices

El primer elemento de M se almacena en la dirección $0x300$.
Queremos asignar un valor 7 en $M[2, 1]$, luego en C:

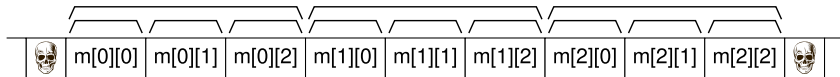
```
int m[3][3];  
m[2][1] = 7;
```

¿Y en ensamblador?

$$0x300 + 4 \cdot 3 \cdot 0 + 4 \cdot 0 = 0x300$$

4 bytes

m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]
m[2][0]	m[2][1]	m[2][2]



$$0x300 =$$

$$0x300 + 4 \cdot 3 \cdot 0 + 4 \cdot 0$$

Matrices

El primer elemento de M se almacena en la dirección $0x300$. Queremos asignar un valor 7 en $M[2, 1]$, luego en C:

```
int m[3][3];  
m[2][1] = 7;
```

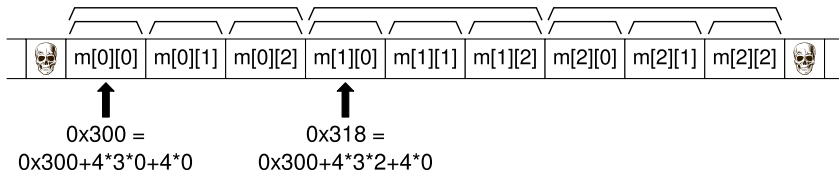
¿Y en ensamblador?

$$0x300 + 4 \cdot 3 \cdot 0 + 4 \cdot 0 = 0x300$$

$$0x300 + 4 \cdot 3 \cdot 1 + 4 \cdot 0 = 0x30C$$

4 bytes

m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]
m[2][0]	m[2][1]	m[2][2]

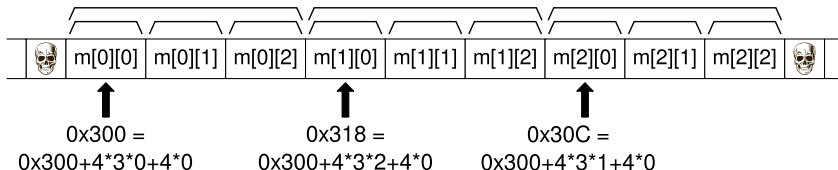
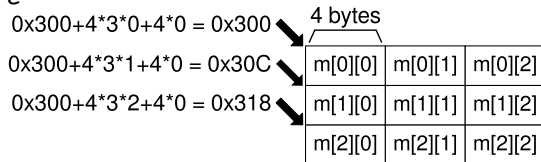


Matrices

El primer elemento de M se almacena en la dirección $0x300$.
Queremos asignar un valor 7 en $M[2, 1]$, luego en C:

```
int m[3][3];  
m[2][1] = 7;
```

¿Y en ensamblador?

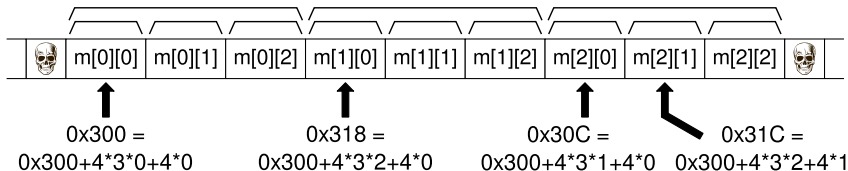
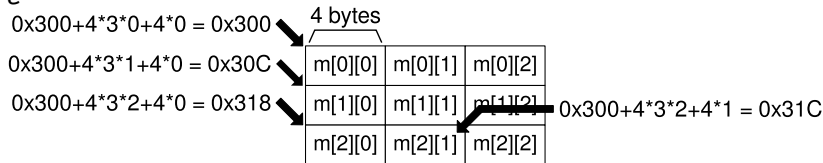


Matrices

El primer elemento de M se almacena en la dirección $0x300$.
Queremos asignar un valor 7 en $M[2, 1]$, luego en C:

```
int m[3][3];  
m[2][1] = 7;
```

¿Y en ensamblador?



Matrices

En general, para una matriz $M[i,j]$:

puntero al inicio	+	tamaño dato	*	índice fila	*	tamaño fila
	+	tamaño dato	*	índice columna		

En Intel:

Si el tamaño de los elementos es un valor válido como escala,

Primero obtengo el offset de la fila

`<índiceFila> * <tamañoDato*tamañoFila>`

ejemplo → `mul rdx ; ojo! modifica rdx también`

Segundo el offset dentro de la fila

`[<índiceFila*tamañoDato*tamañoFila> + <índiceColumna> * <tamañoDato>]`

ejemplo → `lea rsi, [rax+rcx*2]; rsi <= rax+rcx*2`

Tercero, accedo al dato

`[<base> + <índiceFila*tamañoDato*tamañoFila+índiceColumna*tamañoDato>]`

ejemplo → `mov rdx, [rbx+rsi]`

Matrices y Punteros

Si tenemos:

```
int m[3][4];
```

m es un puntero al primer elemento de la matriz.

Luego, vale en C:

```
int *p_m = (int*)m; ← tomo el puntero a la matriz
```

ó

```
int *p_m = &m[0][0]; ← tomo la dirección del primer dato
```

Notación en C

Declaración y *cast* de un puntero a un puntero a matriz:

```
int (*matrix)[rowSize] = (int (*)(rowSize)) p;
```

Se declara la variable `matrix` como un **puntero a una matriz**.

`p` es un **puntero a memoria** que se transforma a matriz.

`rowSize` es la cantidad de datos en una fila (columnas)

Como es un puntero, no se declara la cantidad total de filas.

Esta sintaxis se puede utilizar para declarar matrices de N dimensiones.

```
t (*m)[a]...[z] = (t (*)(a)...[z]) p;
```

Ejercicios

Ejercicio 1

```
short suma(short* vector, short n);
```

Dado un vector de n enteros de 16 bits, devolver la suma de los elementos.

Ejercicio 2

```
void diagonal(short* matriz, short n, short* vector);
```

Dada una matriz de $n \times n$ enteros de 16 bits, devolver los elementos de la diagonal en el vector pasado por parámetro.

Ejercicio 3

```
int* primerMaximo(int (*matriz)[sizeC], int* f, int* c);
```

Dada una matriz de $f \times c$ enteros de 32 bits, encontrar el primer máximo buscando en el orden de la memoria. Devuelve un puntero a este valor y sus coordenadas en f y c

Ejercicios

Ejercicio 1 - Solución

suma:

```
; RDI = vector  
; SI = n
```

```
push rbp  
mov rbp, rsp  
push r12
```

```
xor r12, r12  
xor rcx, rcx  
mov cx, si
```

.cicloSuma:

```
add r12w, [rdi]  
lea rdi, [rdi+2]  
loop .cicloSuma
```

```
mov rax, r12
```

.fin:

```
pop r12  
pop rbp  
ret
```

Ejercicios

Ejercicio 2 - Solución

diagonal:

```
; rdi <-- matriz  
; si <-- filas/columnas  
; rdx <-- vector
```

```
shl rsi, 48
```

```
shr rsi, 48
```

```
lea r8, [rsi*2 + 2]
```

```
mov rcx, 0
```

.ciclo:

```
mov ax, [rdi]  
mov [rdx], ax
```

```
add rdi, r8
```

```
add rdx, 2
```

```
inc rcx
```

```
cmp rcx, rsi
```

```
jne .ciclo
```

```
ret
```

¡Gracias!

¿Preguntas?