



UNIVERSIDAD
NACIONAL DE
SAN MARTÍN

Programación UNSAM Punteros

David López

Memoria y direcciones

- Las variables están almacenadas en memoria RAM
 - La RAM se divide en celdas (casilleros) de 1 byte de tamaño
 - Cada celda tiene una dirección única
 - Una variable ocupa 1 o más celdas
 - Todas las variables tienen asociada una dirección, que es la dirección de la primer celda que ocupan
-

Ejemplo: variable simple

```
int x;
```

```
x = 2;
```

Memoria

1000	
1001	
1002	
1003	02
1004	00
1005	00
1006	00
1007	
1008	

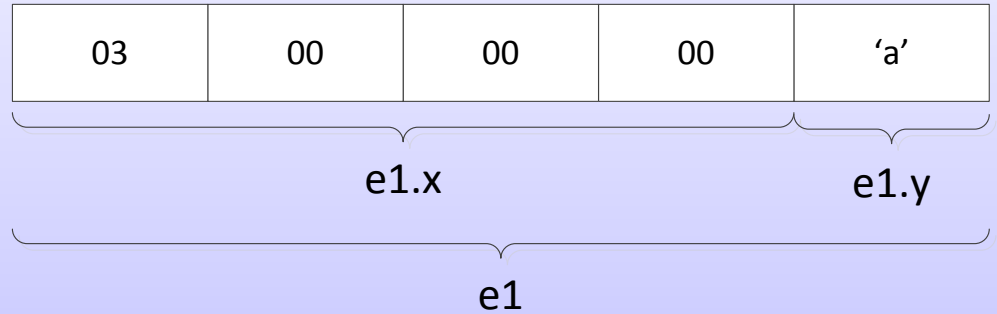
4 bytes

Ejemplo 2: estructura

```
struct estructura {
    int x;
    char y;
};

struct estructura e1;

e1.x = 3;
e1.y = 'a';
```



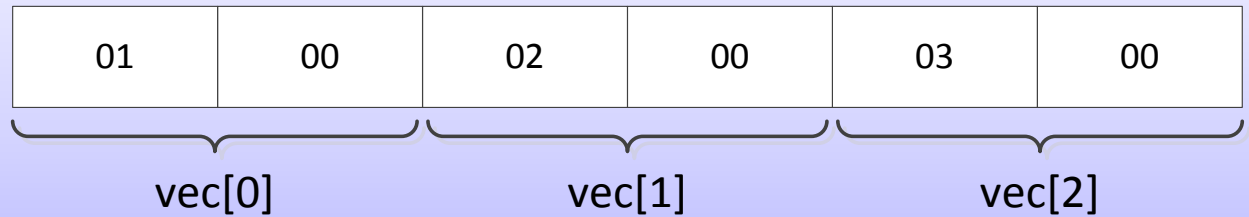
Ejemplo 3: vector

```
short vec [3];
```

```
vec[0] = 1;
```

```
vec[1] = 2;
```

```
vec[2] = 3;
```



Ejemplo 4: matriz

```
#include <stdio.h>
```

```
int main(void) {  
    char m[3][2];
```

```
    m[0][0] = 'a';
```

```
    m[0][1] = 'b';
```

```
    m[1][0] = 'c';
```

```
    m[1][1] = 'd';
```

```
    m[2][0] = 'e';
```

```
    m[2][1] = 'f';
```

```
    return 0;
```

```
}
```

'a'	'b'
'c'	'd'
'e'	'f'

'a'	'b'	'c'	'd'	'e'	'f'						
m[0][0]		m[0][1]		m[1][0]		m[1][1]		m[2][0]		m[2][1]	

Operador dirección (&)

- El operador & devuelve la dirección de memoria de una variable
-

Ejemplo 5

Memoria	1000		}	x
	1001			
	1002			
	1003	02		
	1004	00		
	1005	00		
	1006	00		
	1007			
	1008			

```
int x = 2;
```

```
printf ("%d", x); /* imprime 2 */
```

```
printf ("%p", &x); /* imprime la dirección, ej. 1003 */
```

Formato %p para mostrar un dir. de memoria

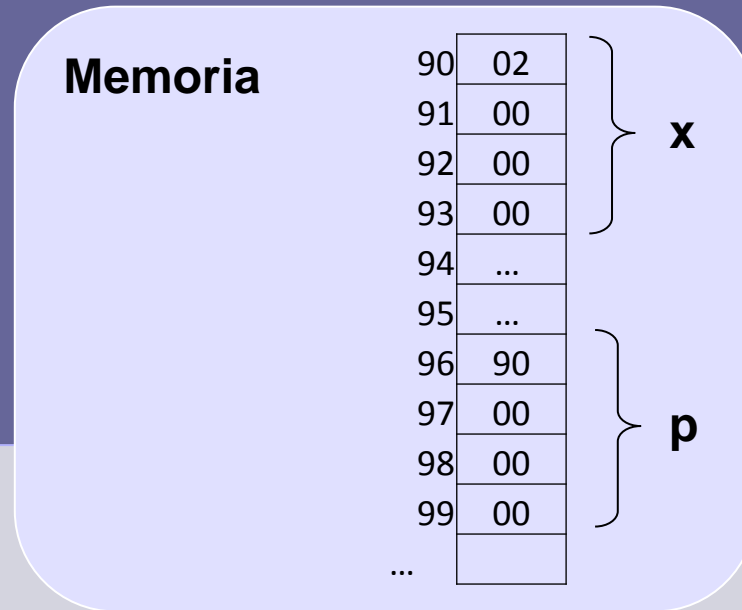
Punteros

● Puntero:

- Tipo especial de variable que sirve para almacenar una dirección de memoria
- Se dice que apunta a una variable si su contenido es la dirección de esa variable
- Se debe declarar indicando a qué tipo de datos va a apuntar (int *, float * , struct xx *, void *. etc.). Por ejemplo:

```
int *p1;  
float *p2;  
struct producto *p3;
```

Ejemplo 6



```
int x;  
int *p;
```

```
x = 2;  
p = &x;
```

```
printf ("La direccion de x es %p y el valor es %d", &x, x);
```

```
printf ("La direccion de p es %p y el valor es %p", &p, p);
```

90

2

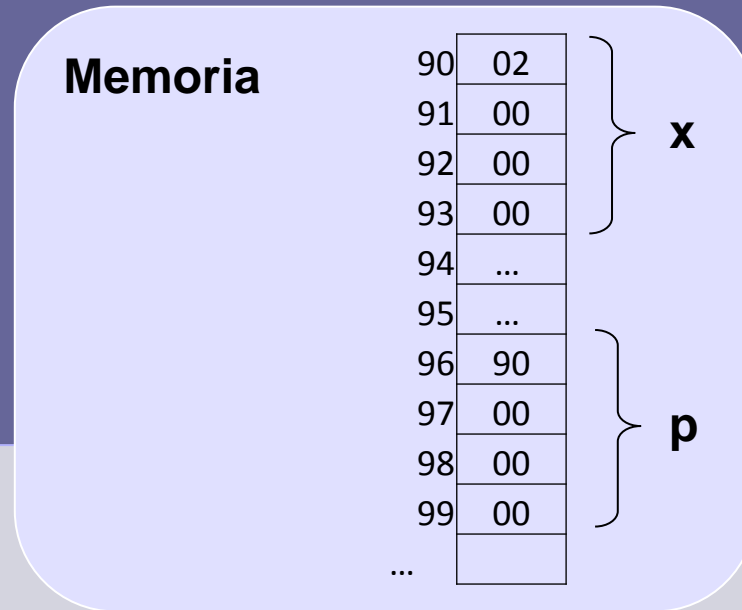
96

90

Indirección

- El operador `*` permite obtener la variable apuntada por un puntero
 - Siempre y cuando el puntero contenga una dirección válida
-

Ejemplo 7



```
int x;  
int *p;
```

```
x = 2;  
p = &x;
```

```
printf ("La direccion de x es %p y el valor es %d", p, *p);  
(*p)++;  
printf ("La direccion de x es %p y el valor es %d", p, *p);
```

90

2

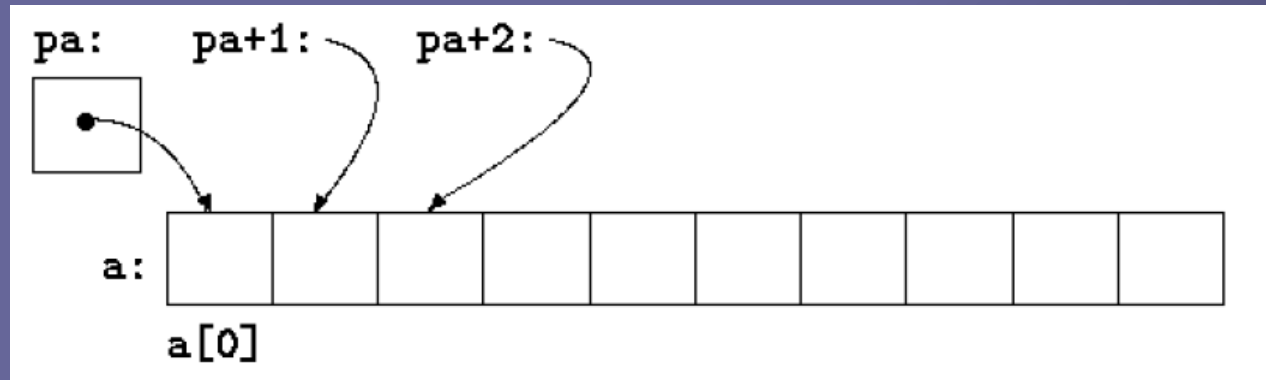
90

3

Relación entre punteros y vectores

- Los vectores son punteros
- Su nombre apunta al primer elemento
- Se puede usar indistintamente los operadores * y [] para acceder a un elemento en un vector
- Ejemplo:

```
int a[10];  
int *pa;  
  
pa = &a[0];
```



Relación entre punteros y vectores

● Ejemplo 8:

Memoria

1000		
1001	15	v[0]
1002	?	v[1]
1003	?	v[2]
1004	?	v[3]
1005	32	v[4]
1006	?	v[5]
1007	?	v[6]
1008
...		

```
char v[10];
```

```
v[0] = 15;
```

```
v[4] = 32;
```

```
printf ("%p", v); /*imprime dir. de comienzo del vec. ej. 1001*/
```

```
printf ("%hhd", *v); /* imprime 15 */
```

```
printf ("%hhd", *(v+4)); /* imprime 32 */
```

Formato %hhd para mostrar char como enteros de un byte de tamaño

Relación entre punteros y vectores

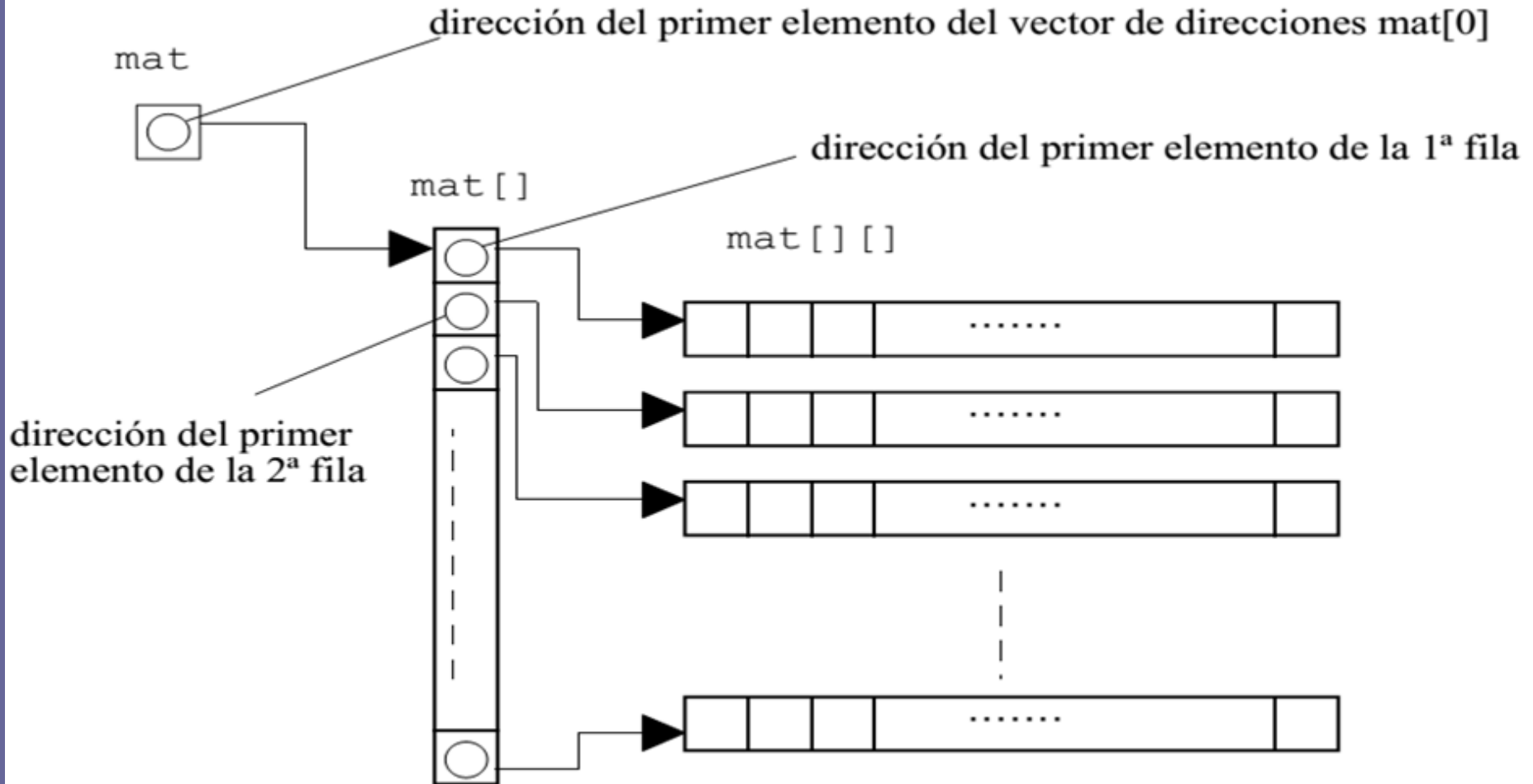
● Conclusión:

- Si v es un vector entonces $v[i]$ es lo mismo que $*(v+i)$

Relación entre punteros y matrices

- Una matriz se puede ver como un vector de vectores (filas)
 - El nombre de la matriz actúa como un vector de punteros, cada uno de los cuales es el vector fila.
-

Relación entre punteros y matrices



Relación entre matrices y punteros.

Ejemplo 9: matriz

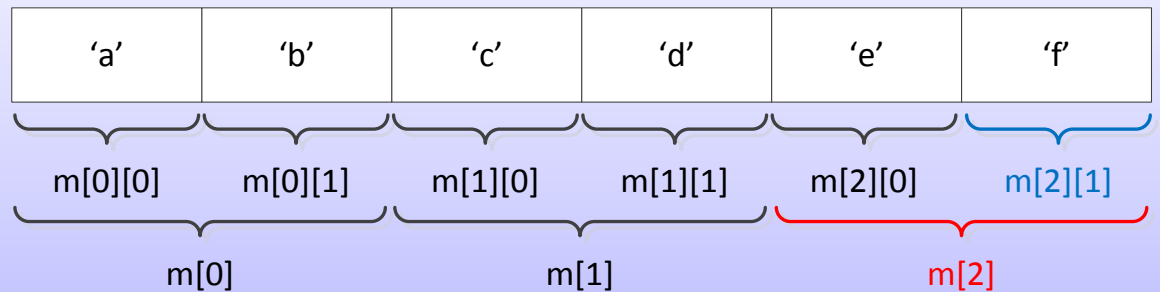
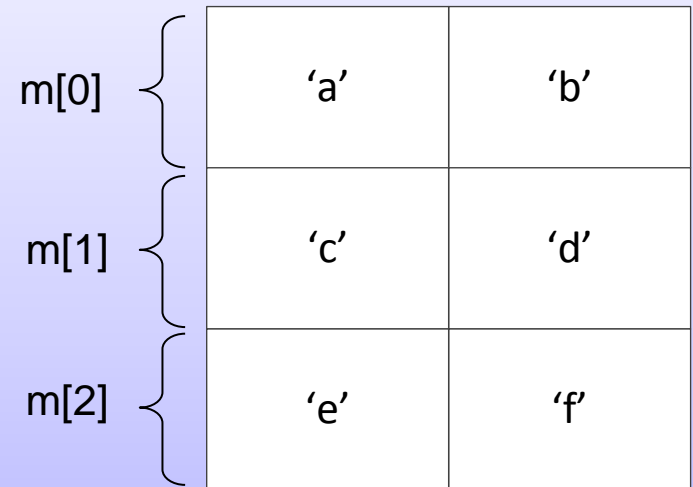
```
#include <stdio.h>
```

```
int main(void) {  
    char m[3][2];  
    char *p;
```

```
    m[0][0] = 'a';  
    m[0][1] = 'b';  
    m[1][0] = 'c';  
    m[1][1] = 'd';  
    m[2][0] = 'e';  
    m[2][1] = 'f';
```

```
    p = m[2]; /* Vector "ultima fila" */  
    printf ("%c\n", p[1]); /* Imprime 'f' */  
    return 0;
```

```
}
```



Aritmética de punteros

- A los punteros se les puede **sumar o restar un entero** o **se pueden restar punteros** para saber la distancia en unidades del dato al que apuntan.
- Los nombres de vectores y matrices estáticos⁽¹⁾ y estructuras **son punteros constantes** por lo cual **no se pueden modificar mediante aritmética pero sí se pueden usar en aritmética**.
- Ver ejemplos en las sig. diapositivas

⁽¹⁾ Son los declarados con []. Ejemplo `int v[100];`

Ejemplo 10

```
#include <stdio.h>


int main () {
    int i, v[4] = {10,20,30,40};
    int *p;

    p = v; /*Igual que p = &v[0];*/
    for (i = 0; i < 4; i++) {
        printf ("Direccion: %p, valor: %d \n", p, *p);
        p++;
    }
    return 0;
}
```

- Al incrementar un puntero de tipo T se avanzan (**sizeof T**) bytes, es decir un elemento de tipo T en un vector

Ejemplo 11

```
#include <stdio.h>

int main () {
    int v[4] = {10,20,30,40};
    int *p; 

    p = v;
    /* v++; Error de compilación*/
    p++; /* OK. Avanza 4 bytes */
    printf ("%d \n", *p); /* Imprime 20 */
    printf ("%d \n", *(v + 1)); /* OK. Imprime 20 */
    printf ("%d \n", *(p + 1)); /* OK. Imprime 30 */
    printf ("%d \n", *(p + 3)); /* Que pasa en este caso????? */
    return 0;
}
```

Intenta acceder más allá
del final del vector

Aritmética de punteros

- Si v es un vector entonces:

Esto	Es lo mismo que
v	$\&v[0]$
$v+i$	$\&v[i]$
$*(v+i)$	$v[i]$
$*v$	$v[0]$

Pasaje de argumentos por valor y por referencia

- En C, los argumentos normalmente pasan *por valor*, es decir se pasa una copia. Ejemplo 12:

```
#include <stdio.h>

void incrementa (short x, short *y) {
    x++;
    (*y)++;
}

int main () {
    short a = 1, b = 2, *p;

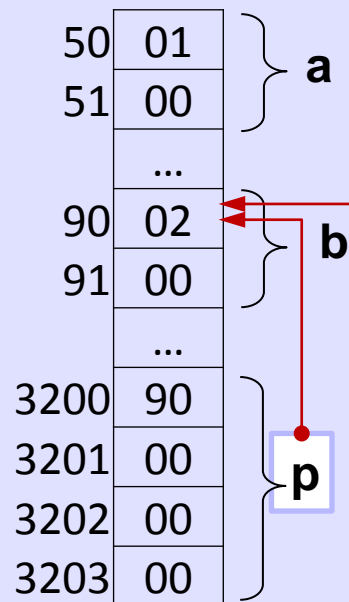
    p = &b;
    printf ("Antes de invocar a incrementa: a:%d b:%d\n", a, b); /*Imprime 1 y 2*/
    incrementa (a, p);
    printf ("Despues de invocar a incrementa: a:%d b:%d\n", a, b); /*Que imprime?*/
    return 0;
}
```

Imprime 1 y 3

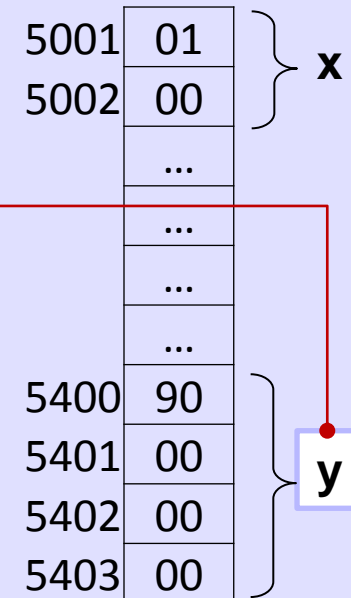
Ejemplo 12

Memoria

main()



incrementa()



Aunque **p** e **y** son dos copias distintas de la dirección de **b**, ambas apuntan al mismo lugar. Por eso la modificación ocurre sobre el valor original (único).

Entonces es una forma de pasar el argumento **b** *por referencia*. Como cuando se usa el **&** para mandar un argumento a una función. Ej. `scanf ("%d", &x);`