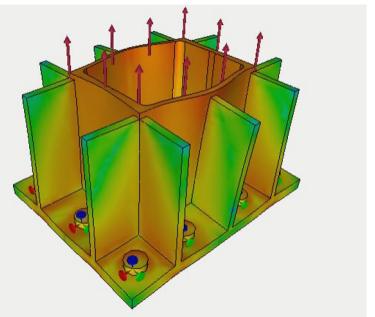
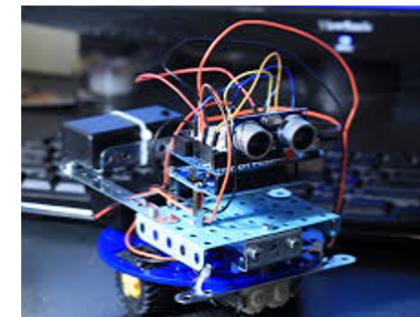
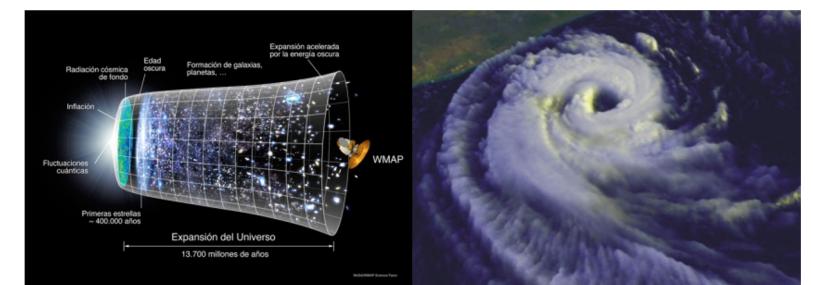
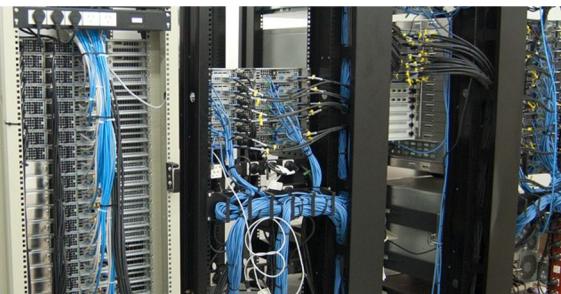




UNIVERSIDAD NACIONAL DE ASUNCIÓN
**FACULTAD DE
INGENIERÍA**

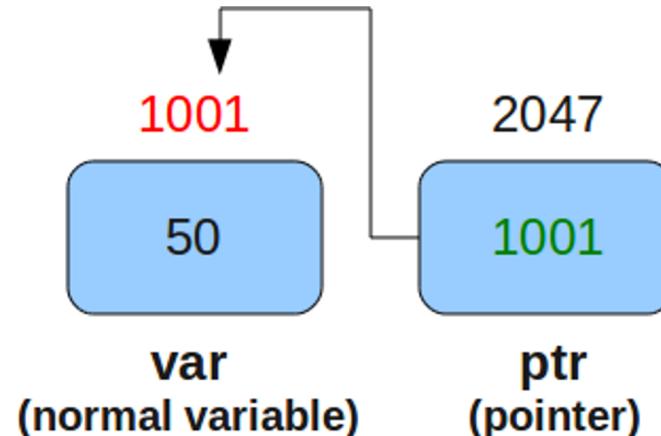
Cursos Básicos

COMPUTACIÓN

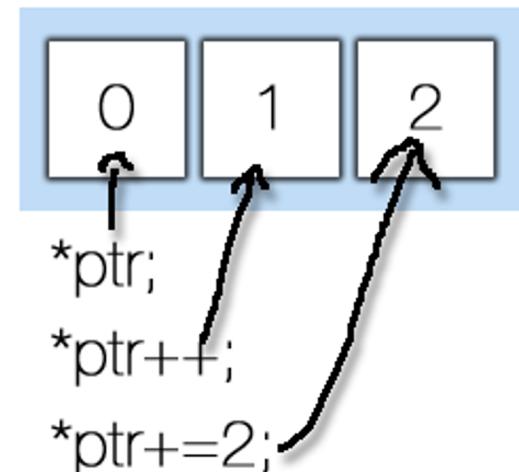


¿Qué veremos hoy?

- Punteros
- Punteros y vectores
- Punteros y matrices
- Punteros a funciones
- Ejercicios



```
int array[3] = {0,1,2};  
int *ptr = array;
```

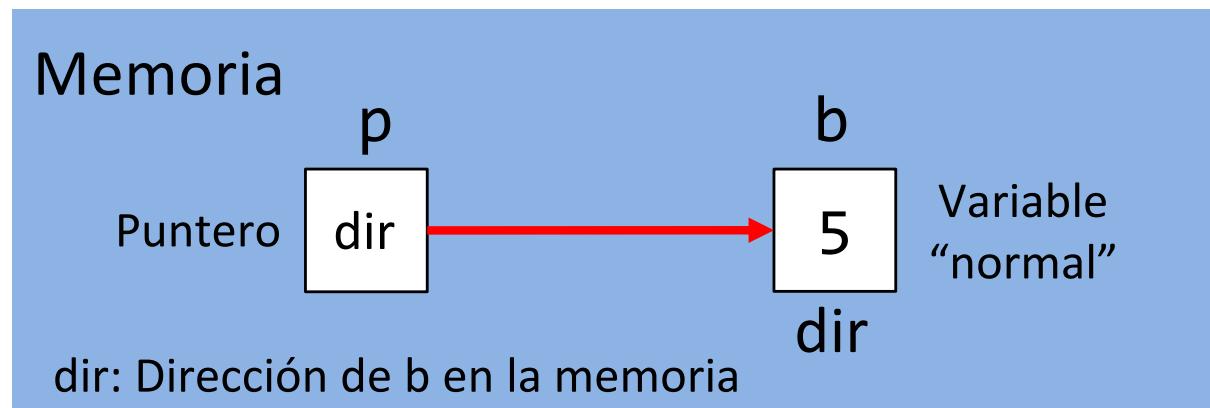


Slides Dr. Jose Colbes

Imágenes de: <https://splearnings.weebly.com/pointers-in-c.html>

Punteros - Definición

Básicamente, el puntero es una variable cuyo contenido es la dirección de otra variable. Los punteros son muy usados en C/C++, y son una de sus características principales.



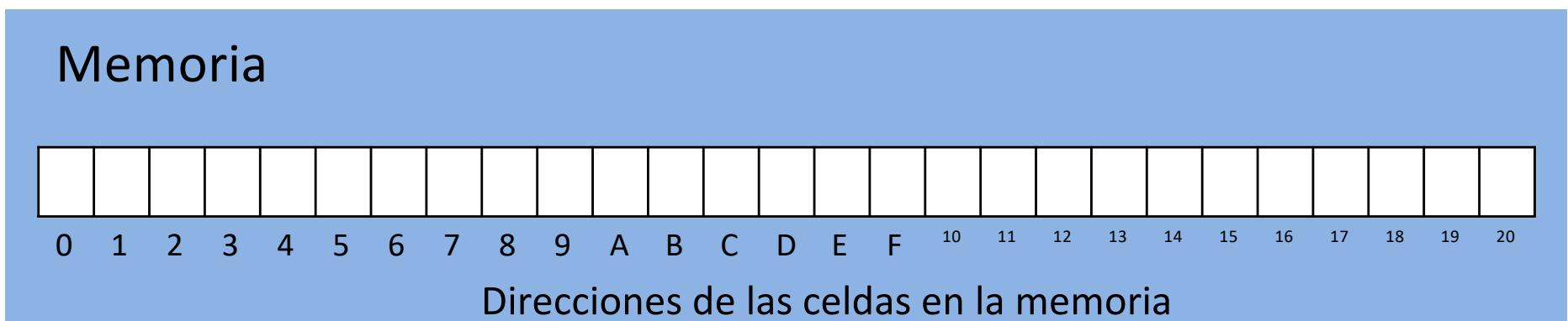
El operador **&** da la dirección de un objeto (variable), entonces la instrucción:

$$p = \&b;$$

asigna al puntero *p* la dirección de la variable *b*. Entonces se dice que "***p apunta a b***".

Memoria – Organización

Una máquina típica tiene un arreglo de celdas de memoria numerados o direccionados de forma consecutiva que pueden ser manipuladas de forma individual o en grupos contiguos.



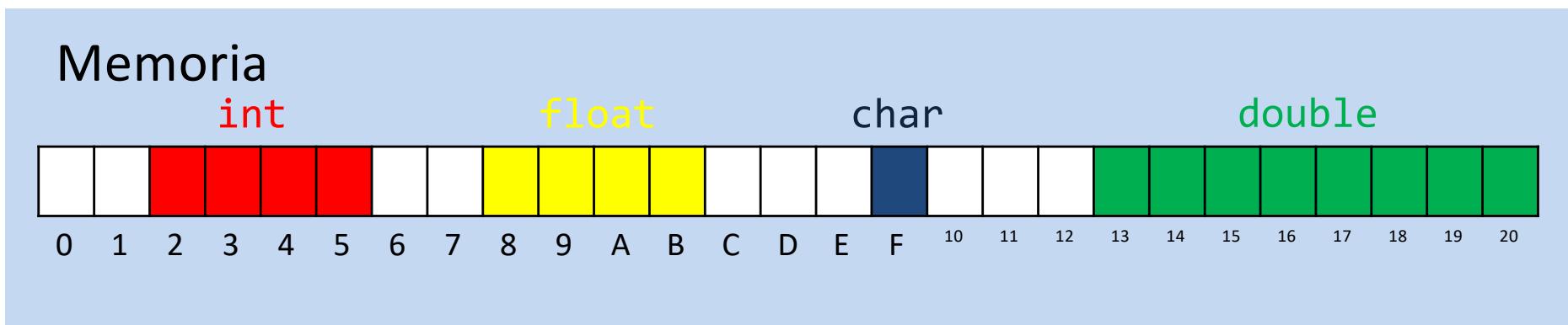
Cada celda es un byte (8 bits)

*Esta es sólo una representación

Tamaño de los tipos de datos

Cada tipo de dato primitivo (int, char, float, double) ocupa un cierto número de celdas en la memoria. Si queremos saber cuántos bytes ocupa un tipo de dato (o arreglos de tipos de datos), contamos con la función `sizeof()`.

*Cada celda es un byte



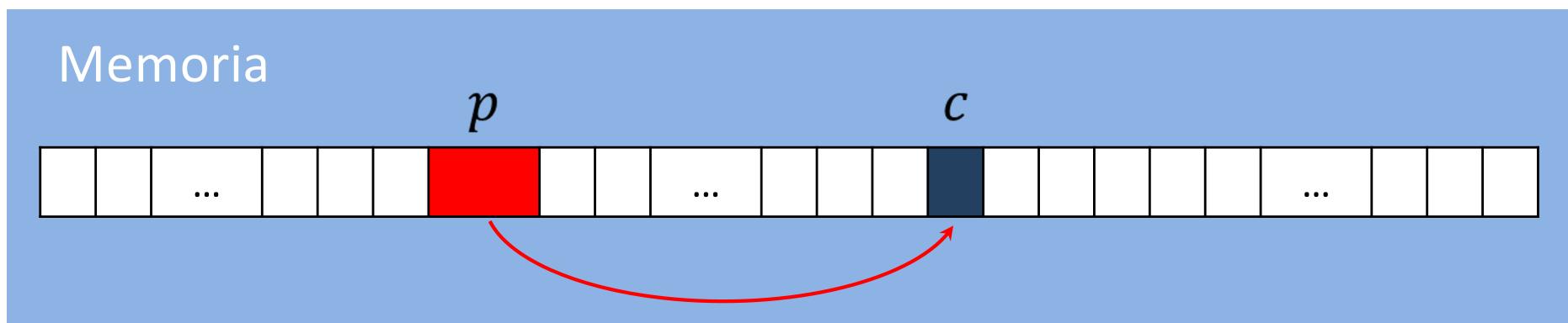
```
int main(){
    char a; cout<<"char: "<<sizeof(a)<<endl;
    int b; cout<<"int: "<<sizeof(b)<<endl;
    long c; cout<<"long: "<<sizeof(c)<<endl;
    float d; cout<<"float: "<<sizeof(d)<<endl;
    double e; cout<<"double: "<<sizeof(e)<<endl;
    long long f; cout<<"long long: "<<sizeof(f)<<endl;
    int g[5]; cout<<"int g[5]: "<<sizeof(g)<<endl;
    system("pause");
    return 0;}
```

C:\Users\oche_

char: 1
int: 4
long: 4
float: 4
double: 8
long long: 8
int g[5]: 20

Punteros – Declaración

Un puntero es un grupo de celdas (a menudo dos o cuatro) que puede contener una dirección. Así que si **c** es un char y **p** es un puntero que apunta a **c**, se podría representar la situación de la siguiente manera:



```
char *p;  
p = &c;
```

Se indica al tipo de dato al
que apunta!
(veremos más adelante la razón)

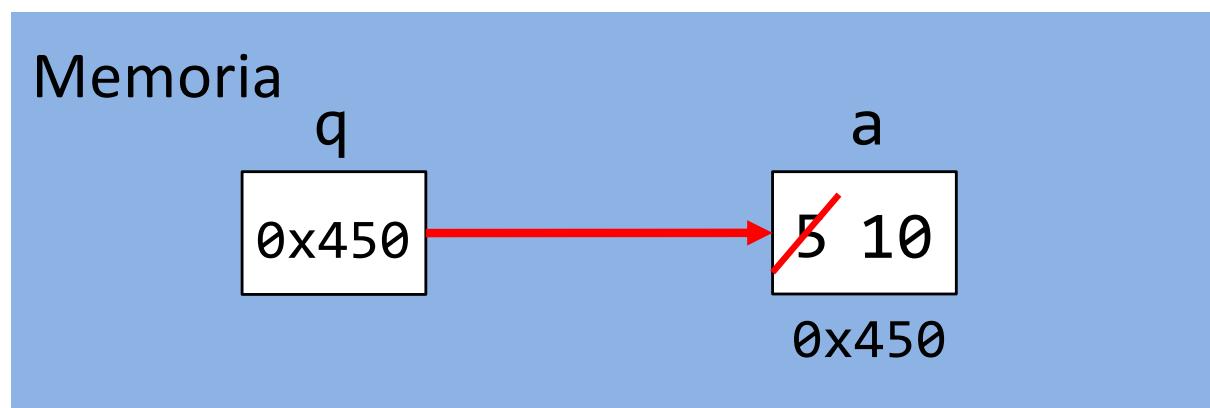
→ **p** contiene la dirección en memoria de la variable **c**

Punteros – Resumen

El puntero es una variable cuyo contenido es la dirección de otra variable.

Dos operaciones importantes son: **&** (para obtener la dirección de una variable), y ***** para acceder a la variable apuntada por el puntero.

```
int a=5;  
int *q; //se declara un puntero a int  
q = &a; //guardamos la dirección de a en q.  
*q=10; //modificamos el valor de a
```

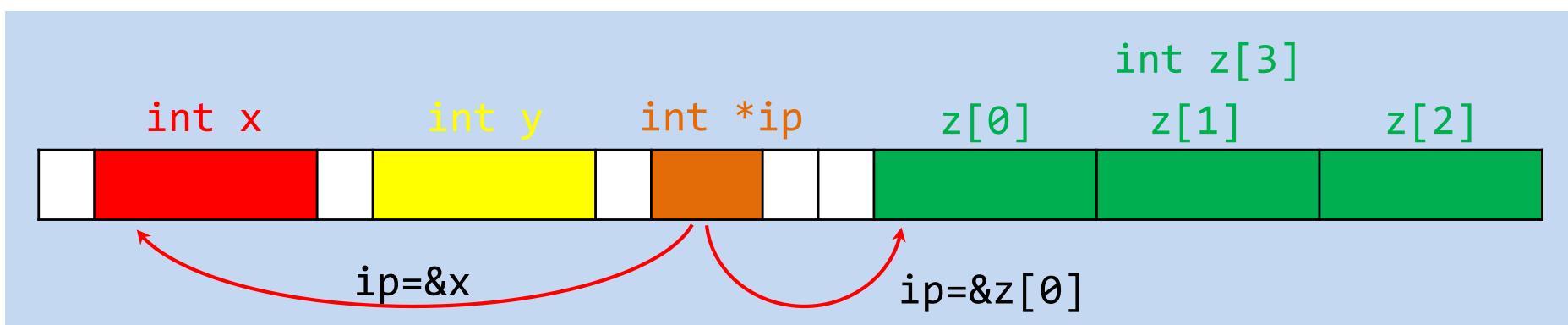


Punteros – Operador *

El operador * es el “desreferenciador”. Al aplicarlo a un puntero, se accede al objeto al que apunta. Un ejemplo del uso de estos operadores es el siguiente:

```
int x = 1, y = 2, z[3];
int *ip; // ip es un puntero a un int
ip = &x; // ip apunta a la variable x
y = *ip; // ahora y es 1
*ip = 0; // ahora x es 0
ip = &z[0]; // ahora ip apunta a z[0]
```

Observación: un puntero está restringido a apuntar a objetos de un tipo específico (por eso se definen con un tipo de dato).



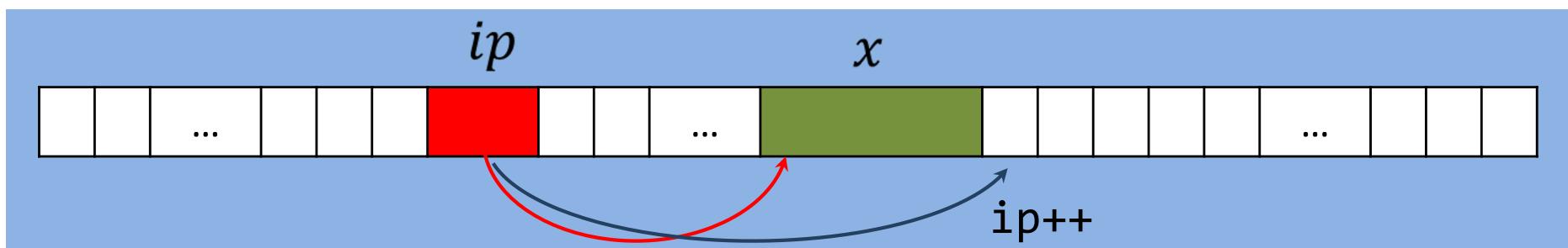
Punteros – Operaciones con *

Si:

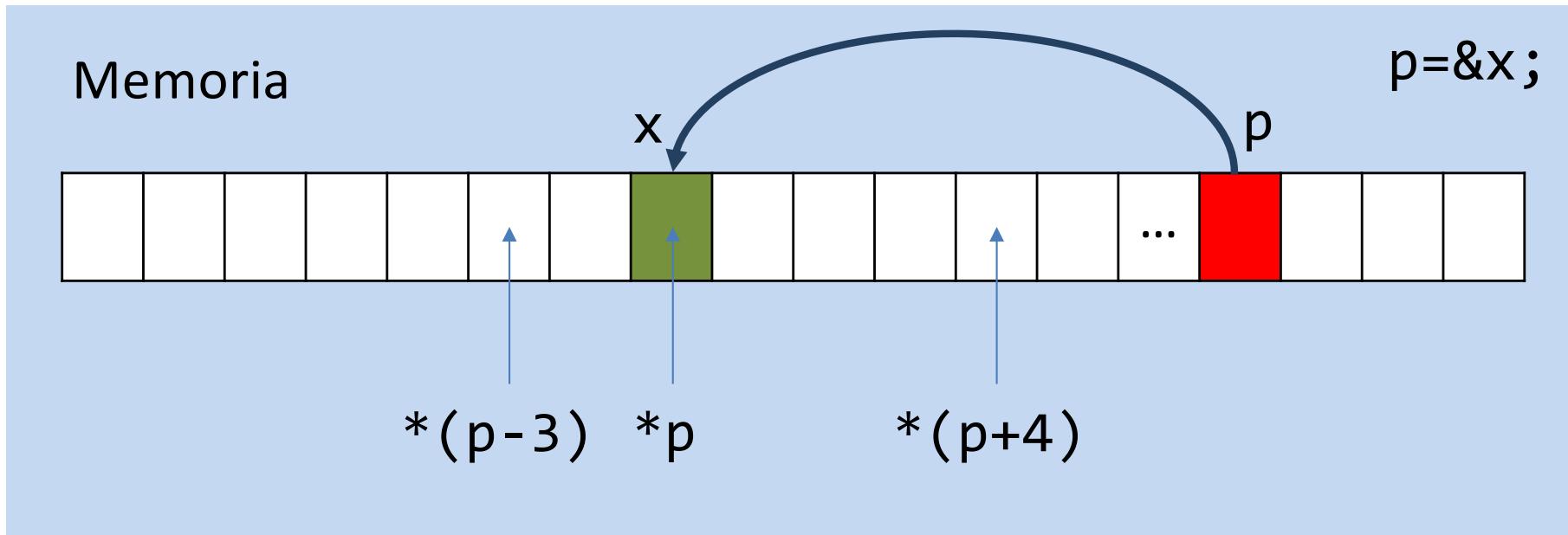
```
int x;  
int *p = &x;
```

Entonces las siguientes operaciones son válidas (para modificar el valor de x o para hacer otras operaciones):

```
*ip = *ip + 10; //incrementa x en 10  
y = *ip + 1; //es lo mismo que poner y=x+1  
*ip += 1; //incrementa el valor de x en 1  
++*ip; //también incrementa x en 1;  
(*ip)++ /*se usa el paréntesis para incrementar x. Si  
no se coloca, se incrementa el valor del puntero!*/
```



Aritmética de punteros



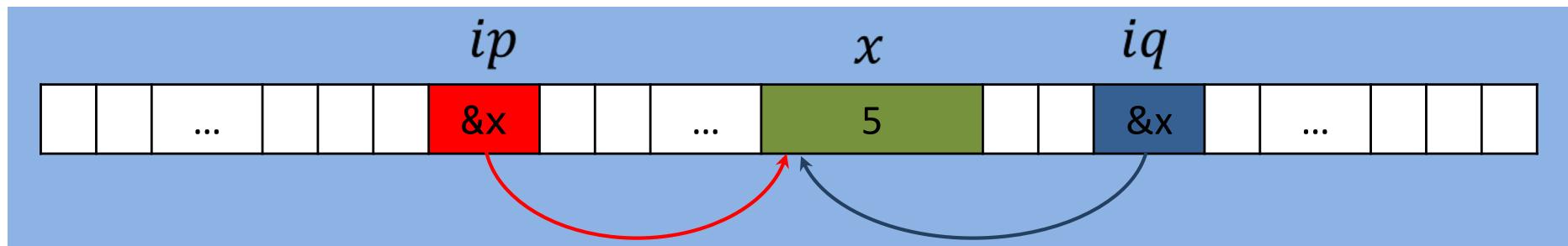
Si sumamos (o restamos) un valor entero a un puntero, estamos adelantando (o atrasando) al puntero una cantidad de bytes de acuerdo al tipo de dato del puntero. Esto nos servirá, entre otras cosas, para acceder a elementos de vectores y matrices!

*En esta representación cada celda es un `int` o puntero a `int`

Punteros – Asignación

Como los punteros son variables, se pueden hacer operaciones sin “desreferenciar”. Por ejemplo:

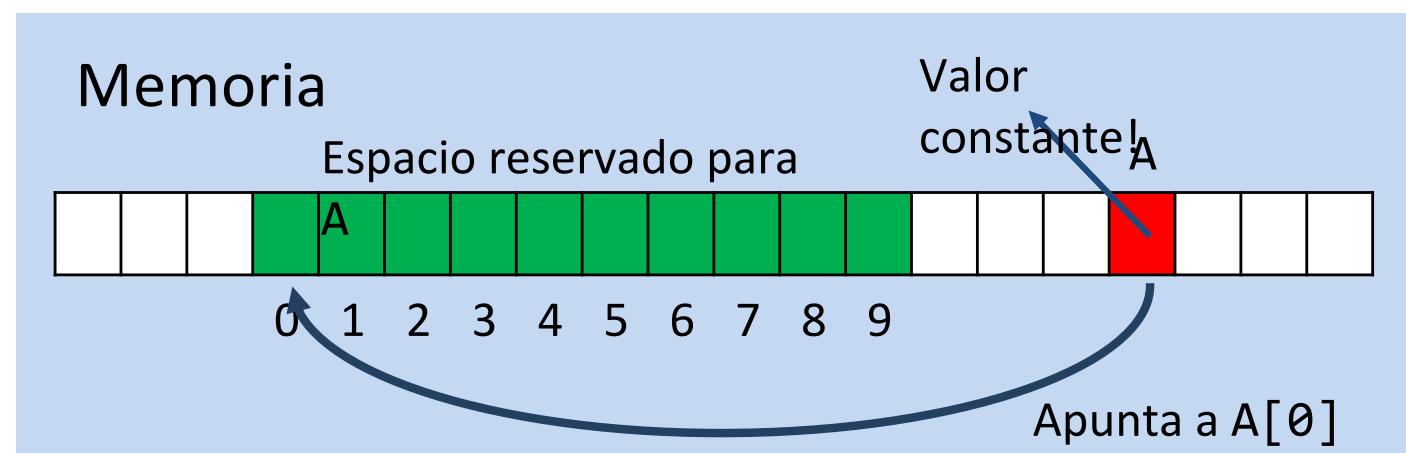
```
int x = 5;  
int *ip, *iq;  
ip = &x;  
iq = ip /*Se copia el valor almacenado en ip a iq, el  
cual es la dirección de x en este momento*/
```



Punteros y vectores

En C/C++, existe una fuerte relación entre los punteros y los vectores (en general, los arreglos); y la mayoría de las veces pueden usarse indistintamente (porque en realidad los nombres de los arreglos son **punteros constantes** que apuntan al primer elemento del arreglo).

La declaración `int A[10];` define un bloque de 10 elementos consecutivos llamados $A[0], \dots, A[9]$.



*En esta representación cada celda es un `int` o puntero a `int`

Punteros y vectores – Acceso

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
A	2	3	5	8	13	21	34	55	89	144
	A	$(A+2)$	$*(A+4)$	$*(A+6)$	$*(A+8)$					
	$*(A+1)$	$*(A+3)$	$*(A+5)$	$*(A+7)$	$*(A+9)$					

Formas equivalentes de acceder a un elemento k del vector A:

$$A[k] \longleftrightarrow *(A+k)$$

Por lo tanto, si tenemos un puntero p y hacemos $p=A$:

$$A \equiv &A[0] \equiv p$$

$$p[3] \equiv *(p+3) \equiv A[3] = 8$$

$$A[k] \equiv *(p+k)$$

Punteros y vectores

Como los nombres de los arreglos son **punteros constantes** que apuntan al primer elemento del arreglo, su valor no se puede modificar. Por lo tanto, hay ciertas operaciones que no están permitidas.

```
int a=5, b[]={1,2,3,4};  
int *p;  
p = &a;
```

Operaciones permitidas

```
p = b;  
p = &b[3];  
a = *(b+1);  
p = b+2;
```

Operaciones inválidas

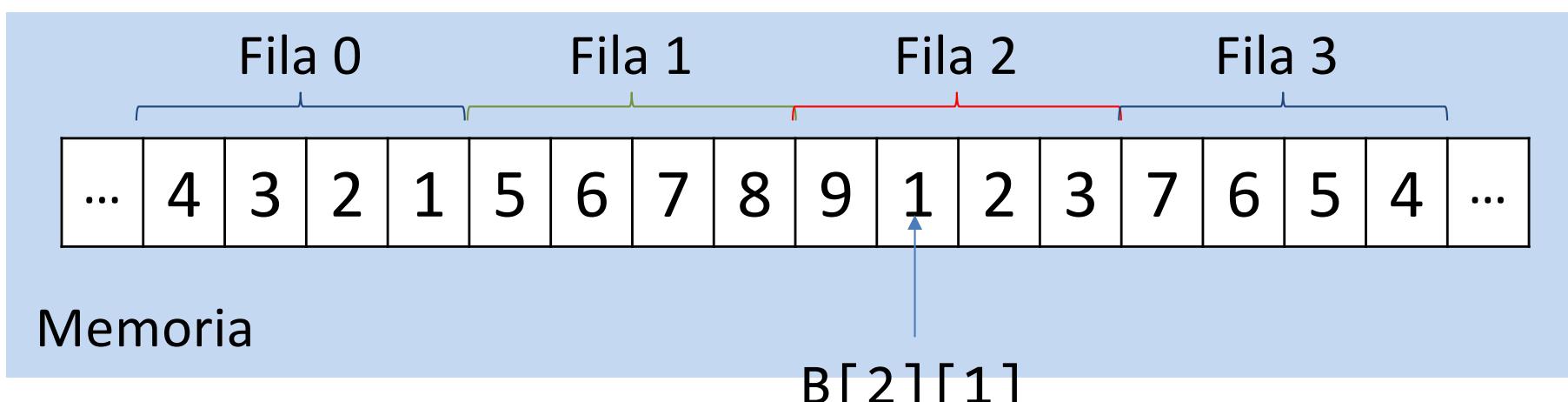
```
b++;  
b = b+3;  
b = &a;  
b = p
```

Punteros y matrices

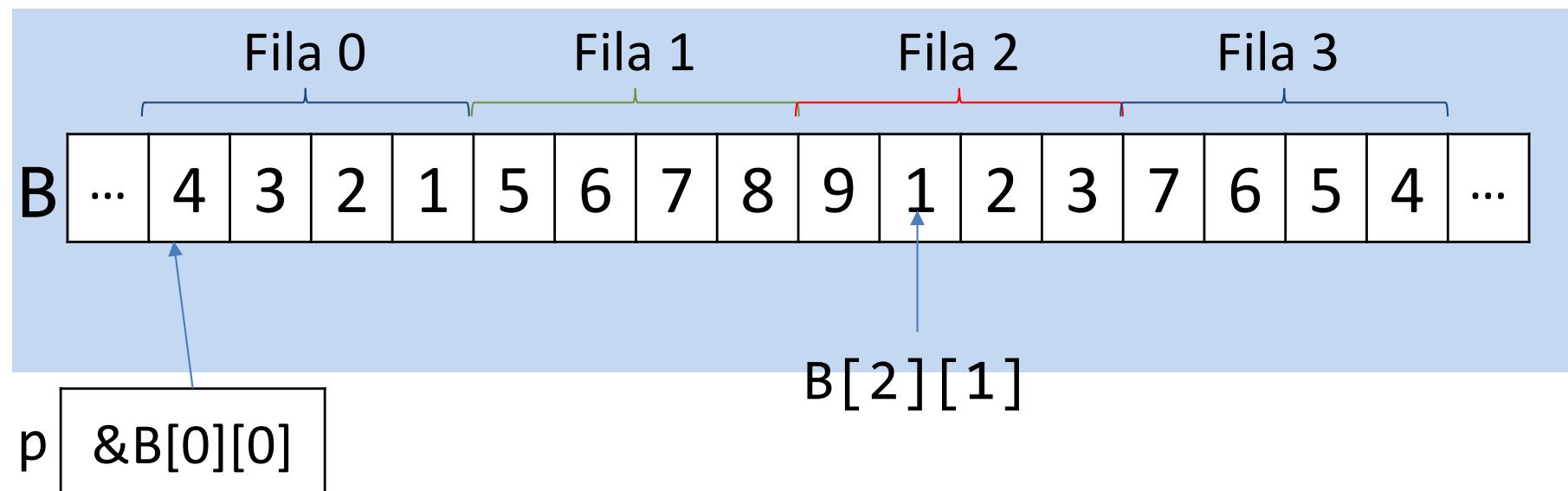
Las matrices pueden considerarse como un vector, donde cada elemento es otro vector. Por lo tanto, una representación esquemática de su almacenamiento en memoria es la siguiente:

		Columnas			
		0	1	2	3
Filas	0	4	3	2	1
	1	5	6	7	8
	2	9	1	2	3
	3	7	6	5	4

$B[2][1]$



Punteros y matrices

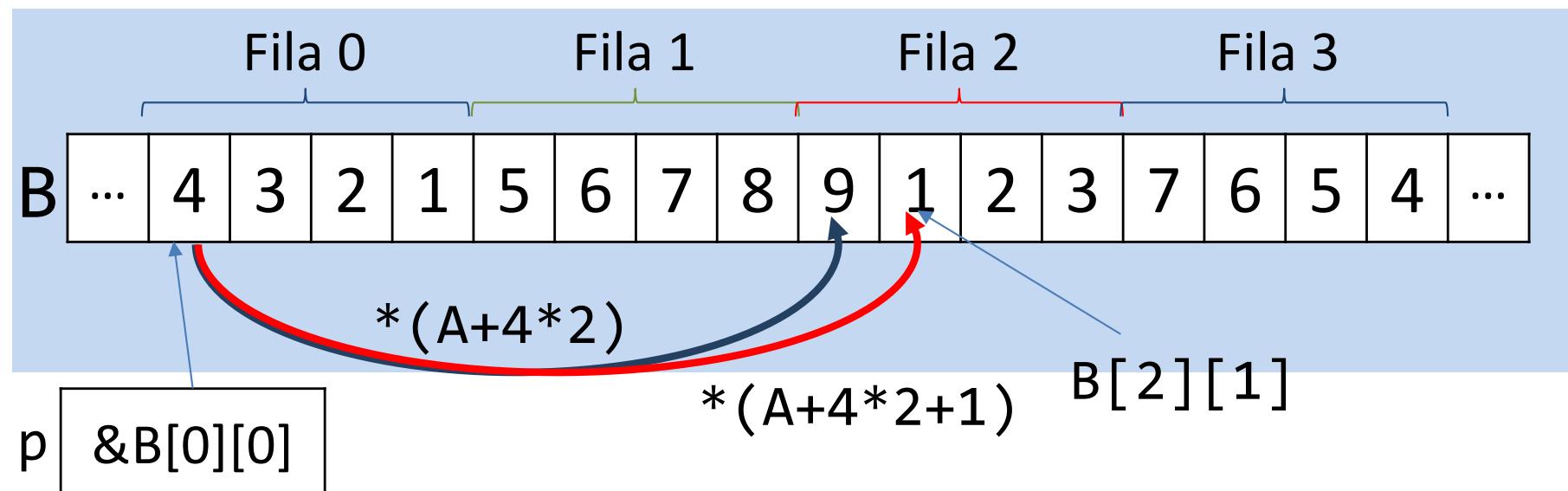


Entonces, si hacemos lo siguiente:

```
int  
*p;
```

¿Cómo podríamos acceder al elemento `B[2][1]` mediante `p`?

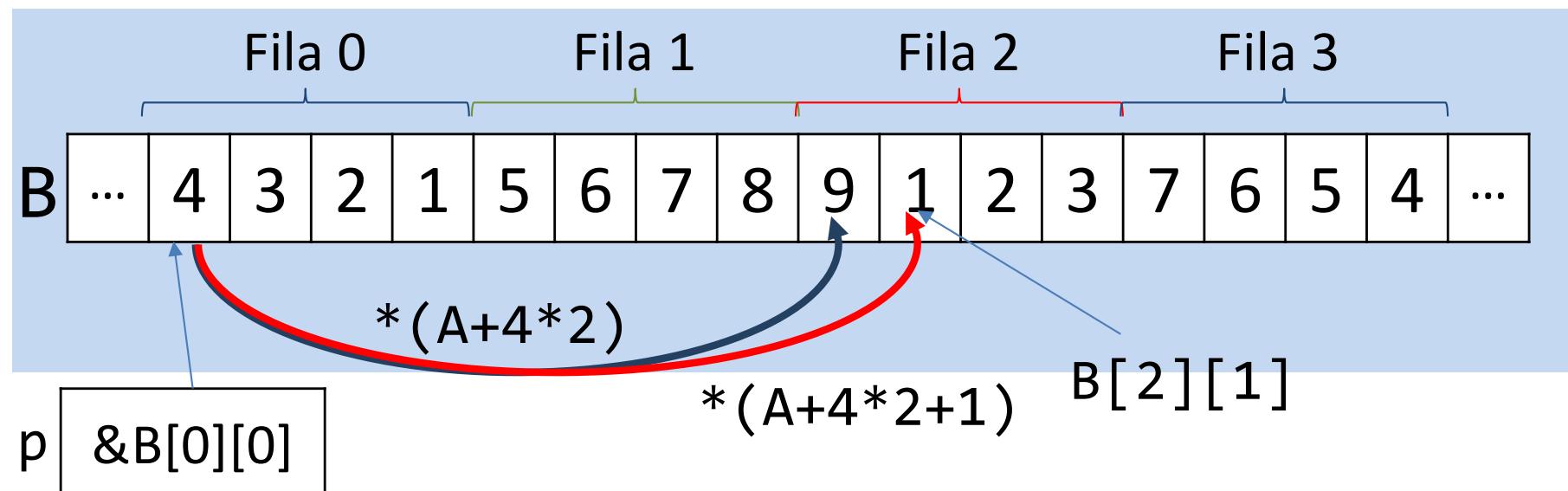
Punteros y matrices



Se tiene que:

$$B[2][1] \longleftrightarrow *(\text{A}+4*2+1)$$

Punteros y matrices



En general, para acceder al elemento $C[i][j]$ de una matriz C de N columnas:

$$C[i][j] \leftrightarrow *(p+i*M+j)$$

Fórmula de direccionamiento

Donde $p=\&C[0][0];$

Gracias por la atención

