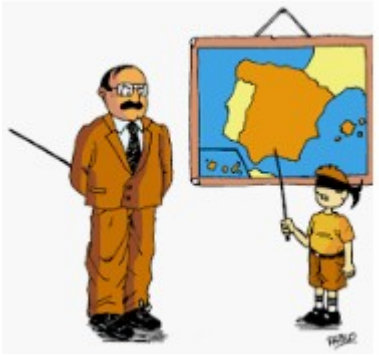
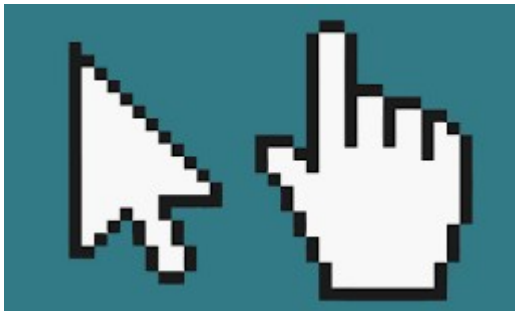
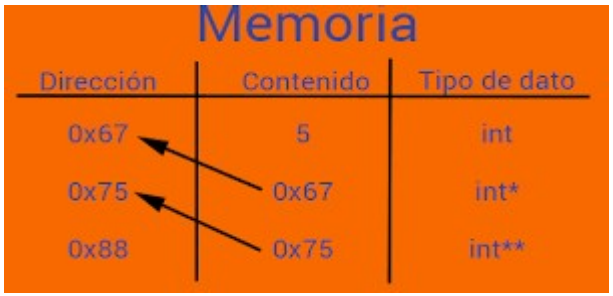

EL PUNTERO

Punteros!!






Memoria
Dirección
Contenido
Tipo de dato
0x67
5
int
0x75
0x67
int*
0x88
0x75
int**

Punteros

- Un puntero es un tipo de datos predefinido de C y de muchos otros lenguajes.
- Su particularidad es que como valor almacena una **dirección de memoria!**

Suele ser la dirección que ocupa otra variable en memoria

Punteros

En la declaración de punteros y posterior asignación de una dirección de memoria a los mismos, se utilizan respectivamente ***** y **&**.

* ⟶ permite obtener el contenido de un objeto apuntado por un puntero.

& ⟶ permite obtener la dirección que ocupa una variable en memoria.

Punteros

¿Cómo se declara una variable de tipo puntero?

Como cualquier otra variable, solo que hay que especificar el tipo de dato al que apuntará. Se utiliza el operador *:

<tipo> * identificador;

Ejemplos:

int *a, *b; → a y b son 2 variables, de tipo puntero a entero

char *c; → c es una variable puntero a char

si se pone void *a el puntero a puede ir cambiando de tipo al que apunta a medida que se ejecuta el programa.

IMPORTANTE

ES IMPORTANTE NOTAR EL DOBLE USO DEL *

PARA DEFINIR UNA VARIABLE SE USA

```
int *
char *
void *
double *
...
```

PARA OBTENER EL VALOR DE UNA VARIABLE SE USA

```
*a = 4;        // se asigna 4 a la dirección de memoria apuntada po
 b = *a;       // b = 4;
```

Punteros

¿Cómo se usa una variable de tipo puntero?

- Para obtener la dirección de una variable, se usa el operador &:

int num; → Esta instrucción reserva memoria para un int.

int *punt_num; → Esta instrucción reserva memoria para un puntero a int.

punt_num = # → Asigna a punt_num la dirección de la variable num

En este caso: **punt_num** apunta a **num**

Punteros

¿Cómo se accede al valor apuntado por un puntero?

- Para obtener el valor de lo apuntado por un puntero se utiliza el operador * :

```
int num;
int *punt_num;
punt_num = &num;
*punt_num = 10;
```

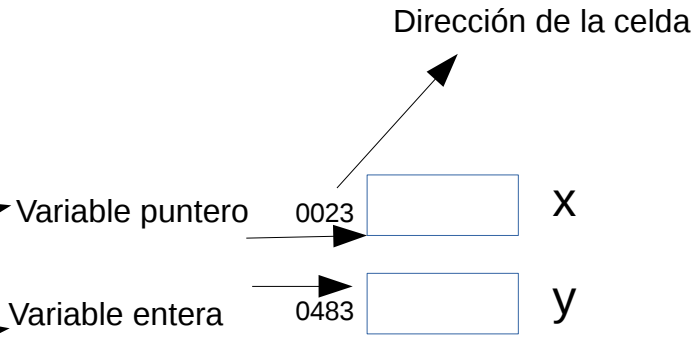
Vamos con un ejemplo!!!

Punteros

```c
#include <stdio.h>

int main(){

    int y, *x;

    x = &y;
    *x = 100;

    printf("y = %d ", y);

    return 1;

}
```

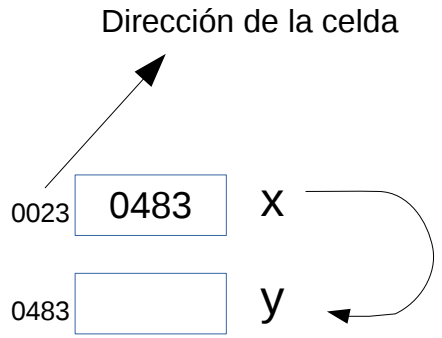
Punteros

```c
#include <stdio.h>

int main(){

    int y, *x;

    x = &y;
    *x = 100;

    printf("y = %d ", y);

    return 1;

}
```



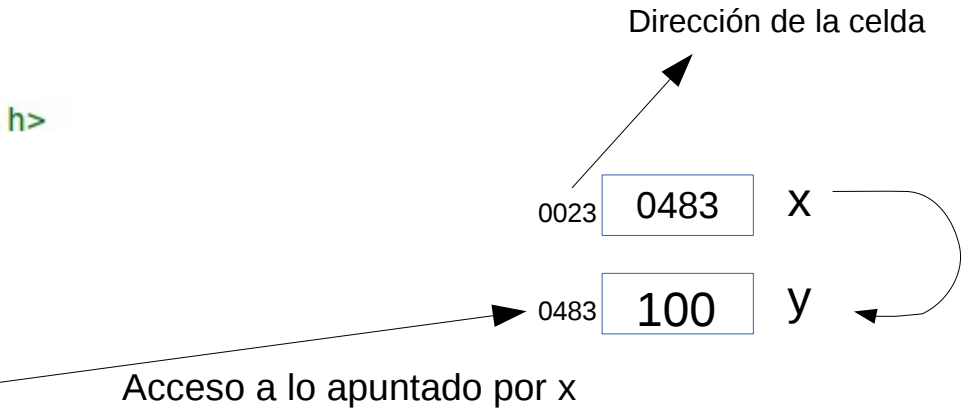
Punteros

```c
#include <stdio.h>

int main(){

    int y, *x;

    x = &y;
    *x = 100;

    printf("y = %d ", y);

    return 1;

}
```



Punteros

```c
#include <stdio.h>

int main(){

    int y, *x;

    x = &y;
    *x = 100;

    printf("y = %d ", y);

    return 1;

}
```



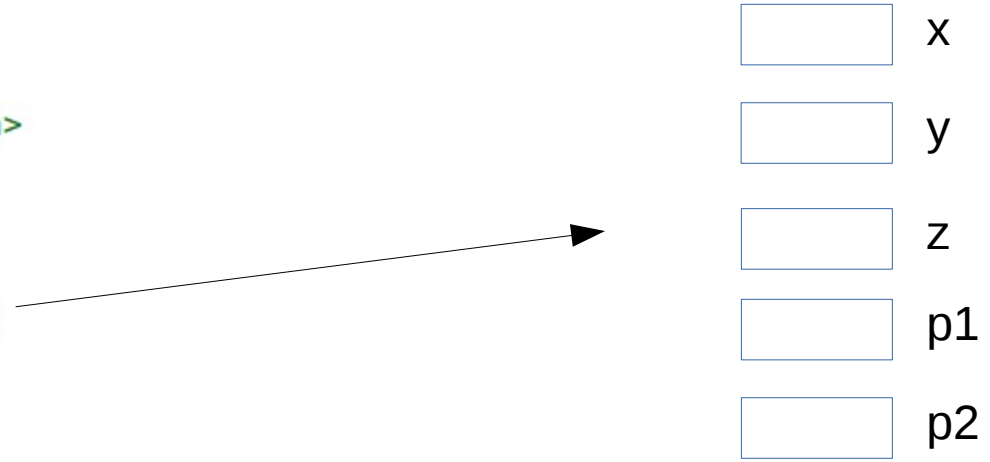

```
>gcc punterosBasico.c -o puntero
>./puntero
y = 100
>
```

Punteros

```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```

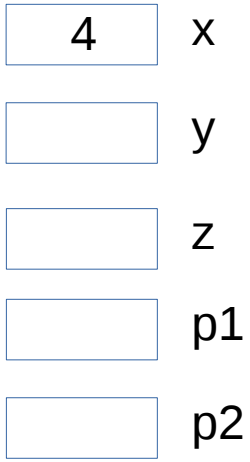
Punteros

```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```



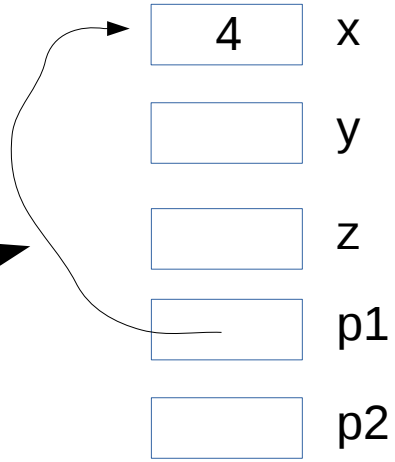
Punteros

```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```



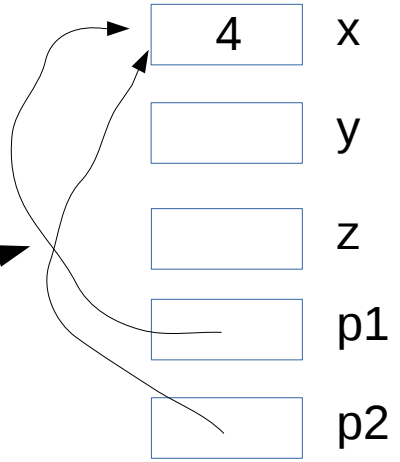
Punteros

```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```



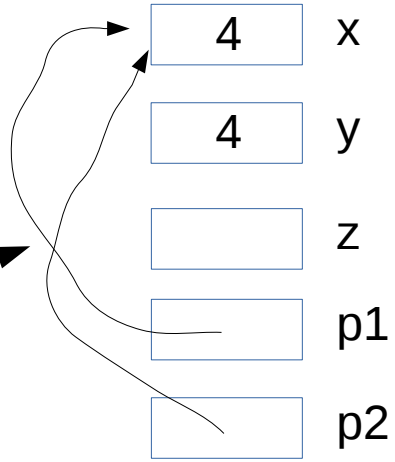
Punteros

```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```



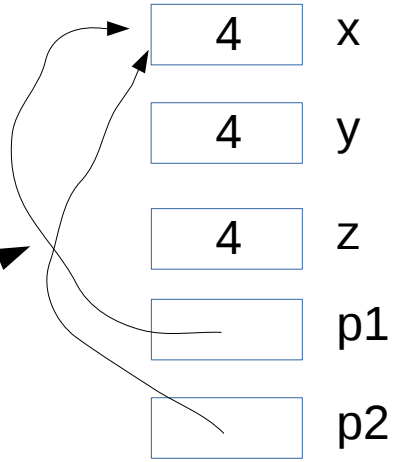
Punteros

```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```

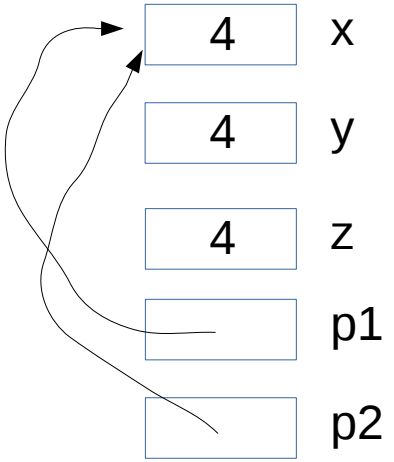


Punteros

```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```



Punteros




```c
#include <stdio.h>

int main() {

    int x, y, z;
    int *p1, *p2;

    x = 4;
    p1 = &x;
    p2 = p1;
    y = *p1;
    z = *p2;

    printf(" x = %d \n", x);
    printf(" p1 = %p | contenido de p1 = %d \n", p1, *p1);
    printf(" p2 = %p | contenido de p2 = %d \n", p2, *p2);
    printf(" y = %d \n", y);
    printf(" z = %d \n", z);

    return 0;

}
```

```
>gcc punterosBasico_2.c -o punteros2
>./punteros2
 x = 4
 p1 = 0x7fffc31a4594 | contenido de p1 = 4
 p2 = 0x7fffc31a4594 | contenido de p2 = 4
 y = 4
 z = 4
>
```

Punteros

El pasaje de parámetros **por referencia** se implementa con **punteros** en el lenguaje C.

Es por esta razón que en los llamados a función, se antecede con & un parámetro pasado por referencia y en el encabezamiento se agrega el * antes del nombre de dicho parámetro.

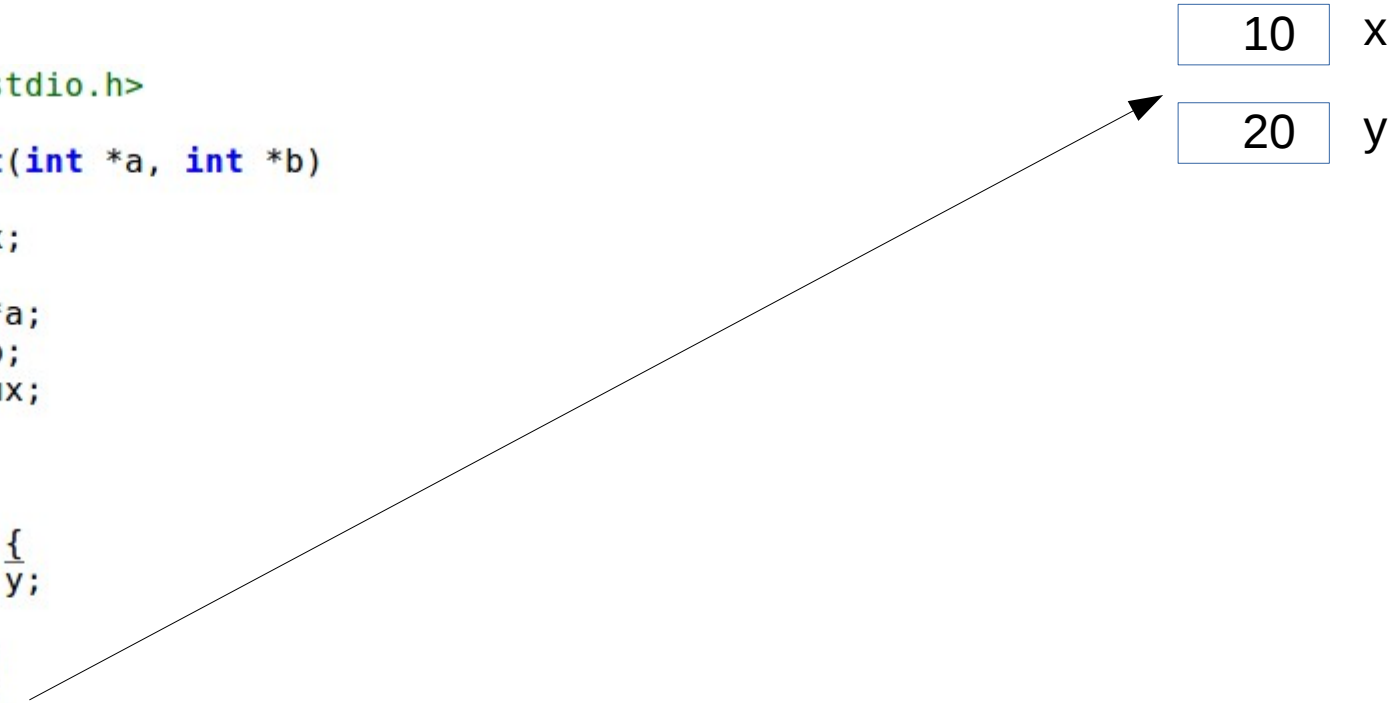
Vamos con un ejemplo!!!

Punteros

```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```

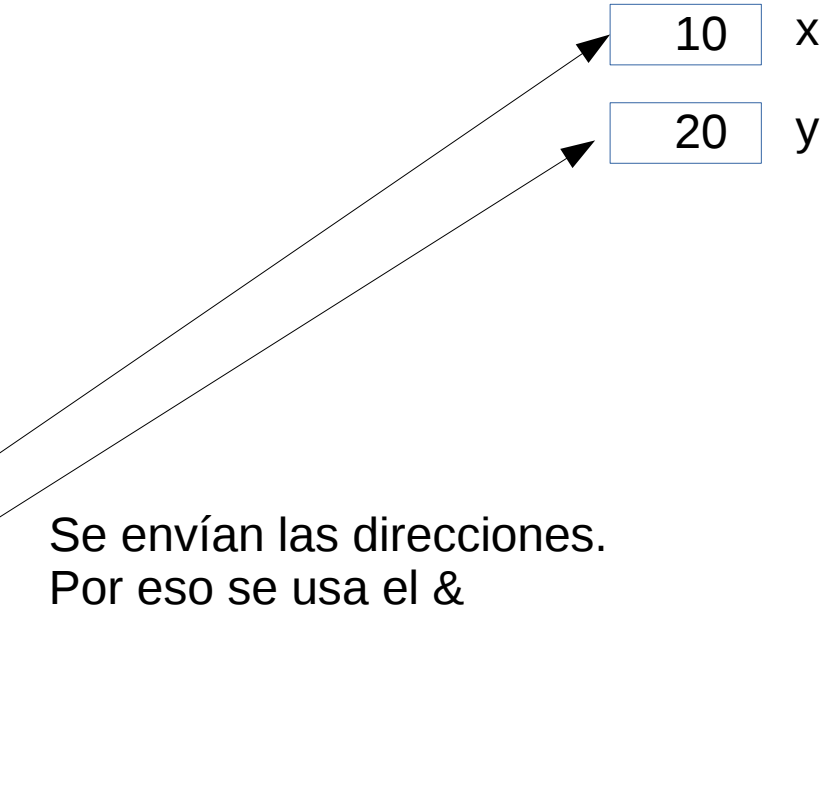
Punteros

```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```



Punteros

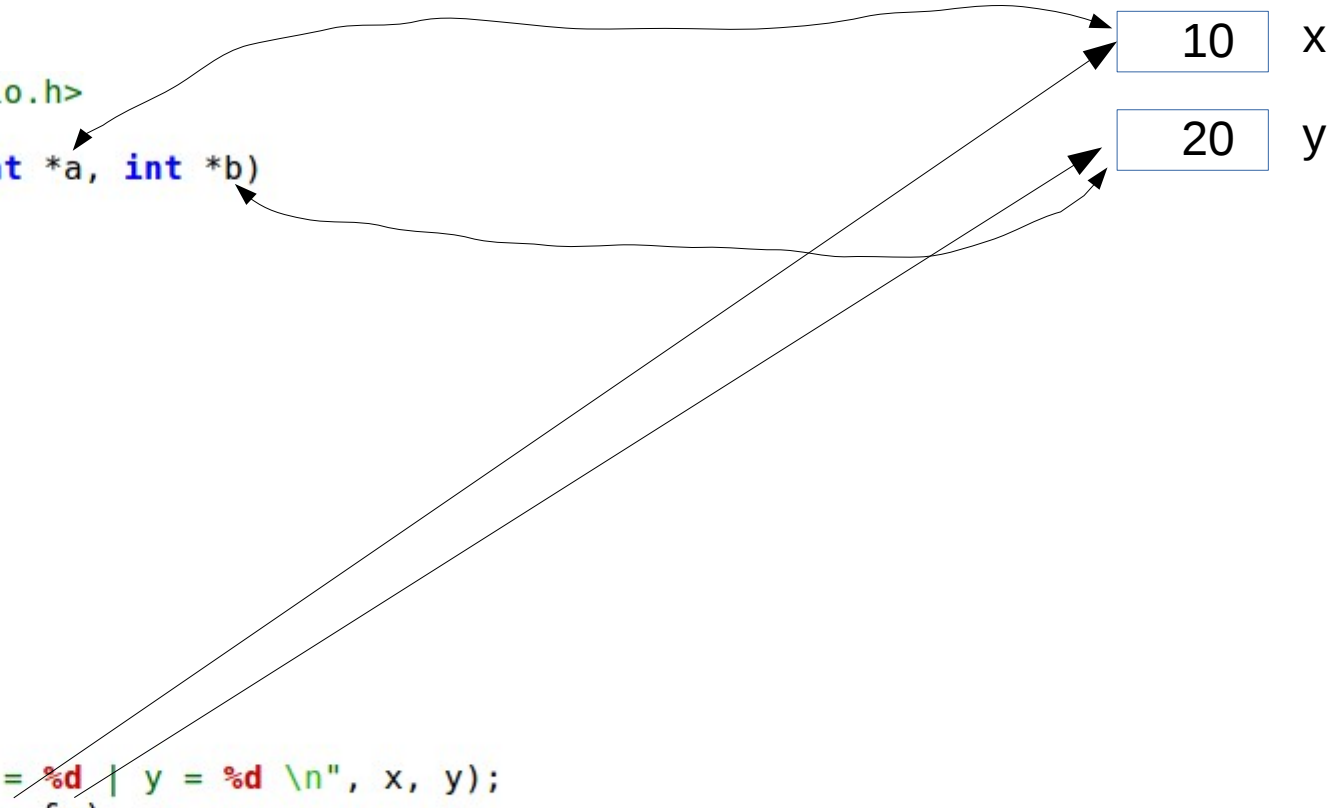
```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```



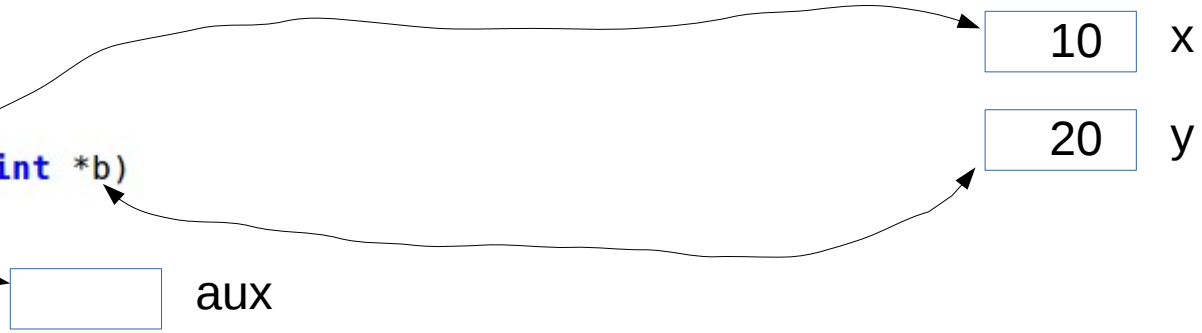
Se envían las direcciones.
Por eso se usa el &

Punteros

```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```

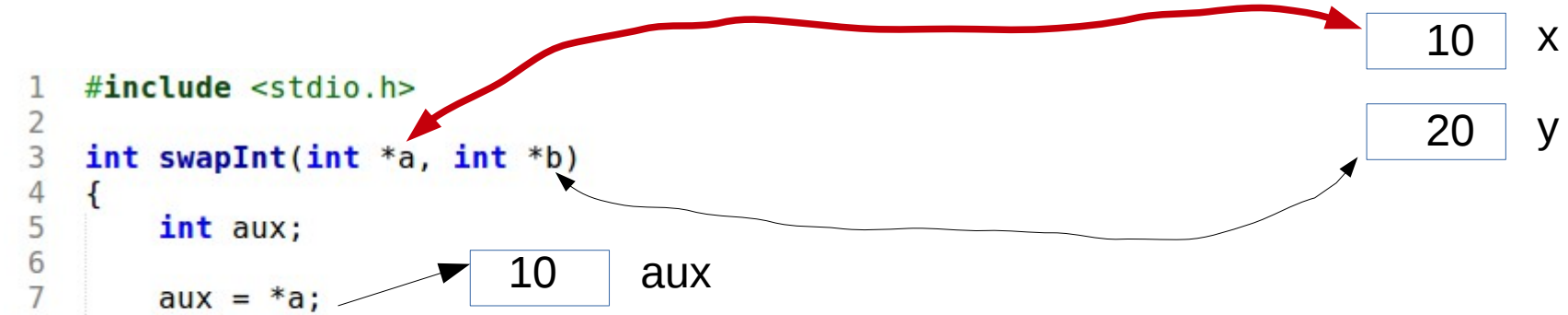


Punteros




```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```

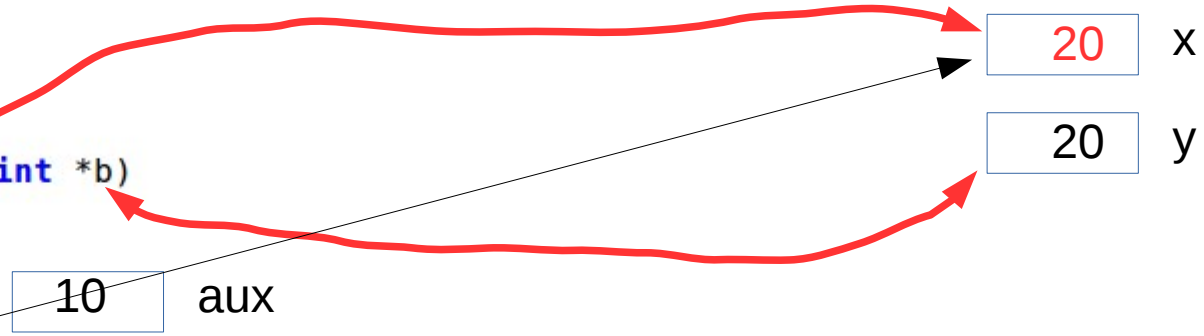
Punteros



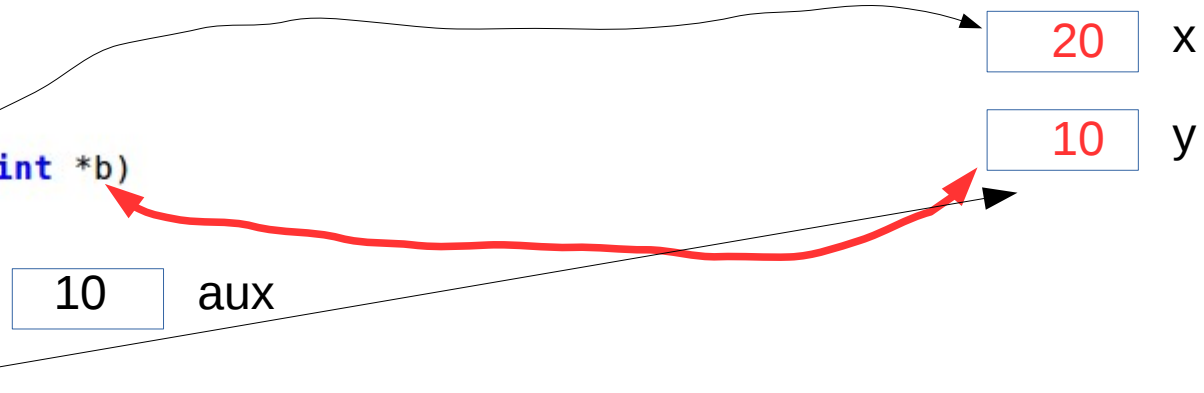

```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```

Punteros

```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```



Punteros




```c
#include <stdio.h>

int swapInt(int *a, int *b)
{
    int aux;

    aux = *a;
    *a = *b;
    *b = aux;
}


int main() {
    int x, y;

    x = 10;
    y = 20;

    printf("x = %d | y = %d \n", x, y);
    swapInt(&x, &y);
    printf("x = %d | y = %d \n", x, y);
}
```

Punteros

```
1  #include <stdio.h>
2
3  int swapInt(int *a, int *b)
4  {
5      int aux;
6
7      aux = *a;
8      *a = *b;
9      *b = aux;
10 }
11
12
13 int main() {
14     int x, y;
15
16     x = 10;
17     y = 20;
18
19     printf("x = %d | y = %d \n", x, y);
20     swapInt(&x, &y);
21     printf("x = %d | y = %d \n", x, y);
22 }
```

20	x
10	y

Finaliza la función swapInt():
- los parametros a y b no existen más
- la variable local aux no existe más
- los valores de x e y quedan intercambiados

```
>gcc swap.c -o swap
>./swap
x = 10 | y = 20
x = 20 | y = 10
>
```

Punteros

Otro ejemplo, uso de structs por parámetro. Uso del operador ->

```c
typedef struct {
  int r,i;
}Complejo_T;


int main() {

  Complejo_T a,b,c,d;

  a.r = 1;
  a.i = 2;

  leer_complejo(&b);
  imprimir_complejo(a);
  imprimir_complejo(b);
```

Pasado por referencia

```c
int leer_complejo(Complejo_T *x)
{
    int a,b;
    scanf("%d", &a);
    scanf("%d", &b);

    x->r = a;
    x->i = b;

    return 0;
}
```

Pasado por referencia

Acceso a los campos r e i.

Se usa el operador - > para acceder a los campos de un puntero a registro y en este caso, x, es un puntero a Complejo_T.

No se usa “.” ya que es un puntero.

Punteros

Aritmética de punteros:

Si se suma o resta un número entero a un puntero, lo que se produce implícitamente es un incremento o decremento de la dirección de memoria contenida por dicho puntero. El número de posiciones de memoria incrementadas o decrementadas depende, no sólo del número sumado o restado, sino también del tamaño del tipo de datos apuntado por el puntero.

```
nombre puntero = nombre puntero + N;
```

se interpreta internamente como:

```
nombre puntero = dirección + N * tamaño tipo de datos;
```

Punteros

Aritmética de punteros:

Es importante advertir que aunque C permite utilizar aritmetica de punteros, esto constituye una práctica no recomemdable. Las expresiones aritmeticas que manejan punteros son difíciles de entender y generan confusión, por ello son una fuente inagotable de errores en los programas.

Pero en C no es necesario usar expresiones aritméticas con punteros, pues C proporciona una notación alternativa mucho más clara (por suerte!!)

Punteros

Punteros y vectores/matrices:

Los vectores y matrices en C están fuertemente relacionados con los punteros.

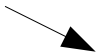
En C, el nombre de una tabla (vector o matriz) se trata internamente como un **puntero que contiene la dirección del primer elemento** de dicha tabla.

De hecho, el nombre de una tabla es una constante de tipo puntero al primer elemento de dicha tabla (y C no permite modificar el valor de dicha constante puntero).

Si tenemos:

```
int vector[15];
```

Entonces, **`vector`** es equivalente a **`&vector[0]`**



La dirección a la primer posición del arreglo!!

La instrucción vector = vector +1; genera un error!! porque es una constante puntero!!

Punteros

Punteros y vectores/matrices:

C permite el uso de punteros que contengan direcciones de los elementos de una tabla para acceder a ellos y a cada uno de sus componentes:

```
int vector[15];

int *puntero, aux;

puntero = vector;

puntero = puntero + 3;

aux = *puntero;
```

⟶ esto es equivalente a `aux = vector[3];`

Pero la relación entre punteros y tablas en C va aún más allá. Una vez declarado un puntero que apunta a los elementos de una tabla, pueden usarse los corchetes para indexar dichos elementos, como si de una tabla se tratase. Así, siguiendo con el ejemplo anterior, sería correcto escribir:

```
scanf( "%c", puntero+2 ); /* puntero[0] equivale a vector[2] */

puntero[7] = 97; /* puntero[7] equivale a *(puntero +7) */
```

Punteros

Analizar y explicar qué realiza el siguiente programa:

```c
#include <stdio.h>
#define DIM 10


void main() {

    int v1[DIM], v2[DIM];
    int i, fuerza1, fuerza2;
    int *fuerte;

    /* Lectura de los vectores. */
    for (i= 0; i< DIM; i++)
        scanf( "%d ", &v1[i] );

    for (i= 0; i< DIM; i++)       // si tenemos int *paux; paux=v2
        scanf( "%d ", &v2[i] );   // seria lo mismo poner scanf( "%d ", paux+i );

    /* Calculo de la fuerza de los vectores. */
    fuerza1 = 0;
    fuerza2 = 0;
    for (i= 0; i< DIM; i++) {
        fuerza1 = fuerza1 + v1[i];
        fuerza2 = fuerza2 + v2[i];
    }

    if (fuerza1 > fuerza2)
        fuerte = v1;
    else
        fuerte = v2;

    /* Escritura del vector mas fuerte. */
    for (i= 0; i< DIM; i++)
        printf( "%d ", fuerte[i] );
}
```