

ARREGLOS Y PUNTEROS

(Parte 04)

Elaborado por: Juan Miguel Guanira Erazo

PUCP

PUNTEROS A FUNCIÓN

DEFINICIÓN:

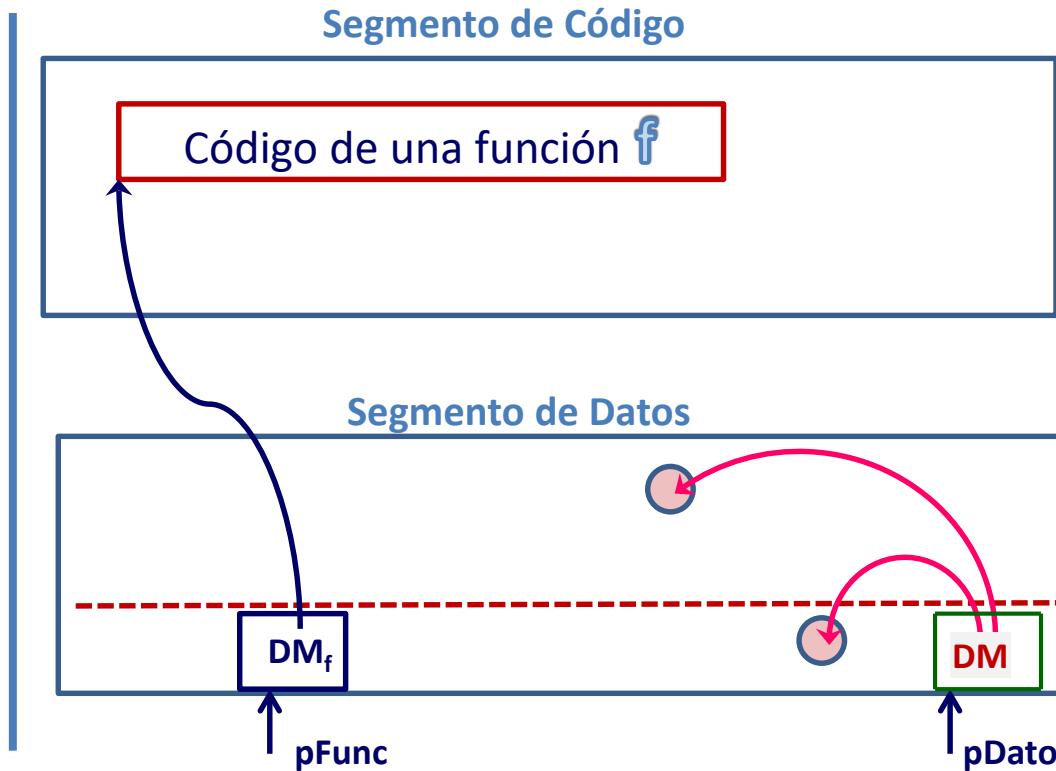
Un **"puntero a función"** es aquel que en lugar de apuntar a un dato, apunta al código de una función.

Esto se puede dar porque en todo programa se crea una la tabla de identificadores, y allí se colocan también los identificadores de las funciones con sus respectivas posiciones de memoria.

Por lo que un puntero puede almacenar esa dirección y ejecutar la función a través de puntero.

...***pDato**;

...***pFunc**;



DECLARACIÓN:

Antes de declarar un puntero a función debe tener muy claro lo siguiente:

```
double *a(int, int);
```

Esto no es un puntero a función, es un encabezado de una función llamada `a` que recibe dos enteros como parámetros y devuelve un puntero double.

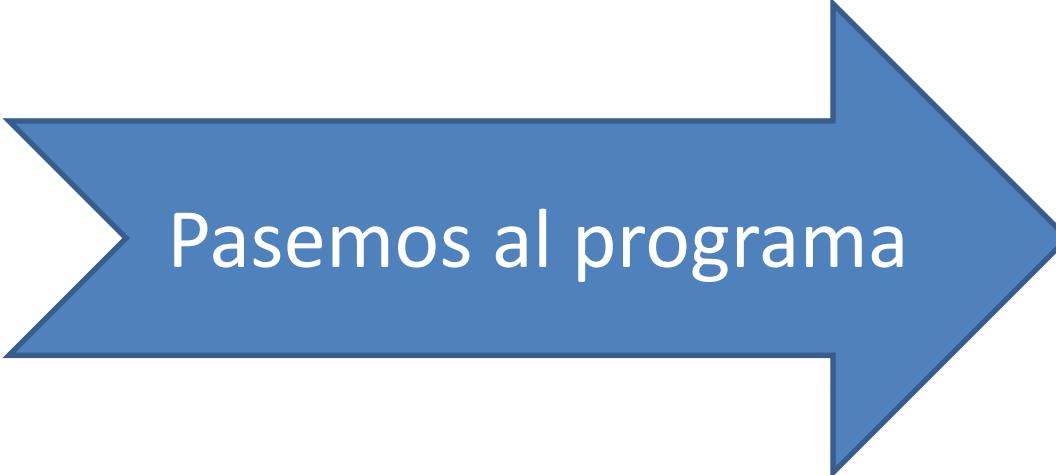
DECLARACIÓN:

Un puntero a función se define como:

```
double (*a) (int, int);
```

Esto se lee de la siguiente manera:

“a” es un puntero que puede apuntar a cualquier función cuyo encabezado indique que recibe dos enteros como parámetros y devuelve un double.



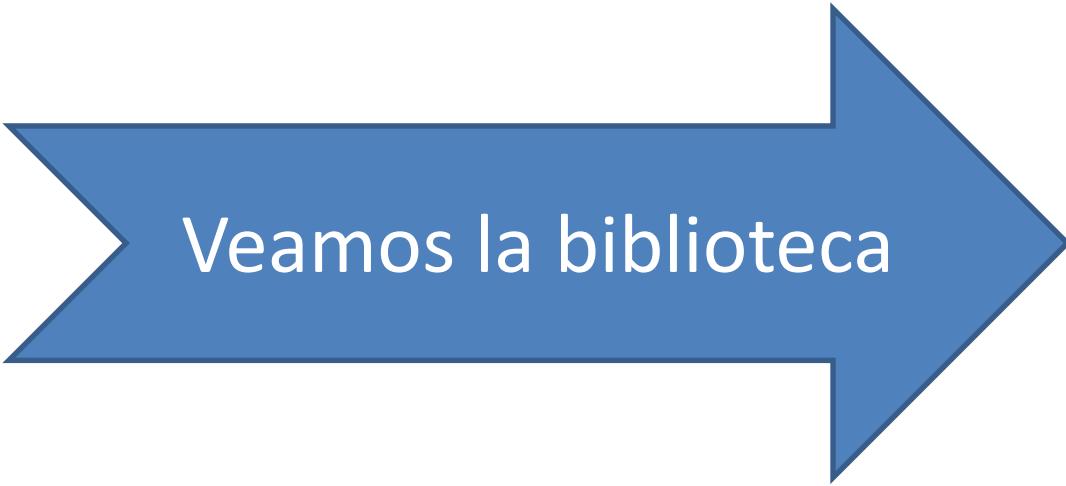
Pasemos al programa

USOS DE PUNTEROS A FUNCIÓN

Lo primero que veremos serán ejemplos de funciones, ya implementadas, en los que se aprecie el uso de los punteros a función.

Estas funciones son: **qsort** y **bsearch**

Ambas definidas en: **cstdlib**



Veamos la biblioteca

FUNCIÓN `qsort`

DEFINICIÓN:

qsort es una función “genérica”

Esto quiere decir que es capaz de ordenar cualquier conjunto de datos sin importar el tipo de dato que lo conforma ni la estructura que contenga a esos datos, y en el orden que se deseé.

INVOCACIÓN:

La función se invoca de la siguiente manera:

qsort (arr, nd, size, cmp);

Donde:

arr: es un arrglo de cualquier tipo.

nd: es el número de datos del arreglo.

size: es tamaño en bytes de los elementos que conforman el arreglo primario

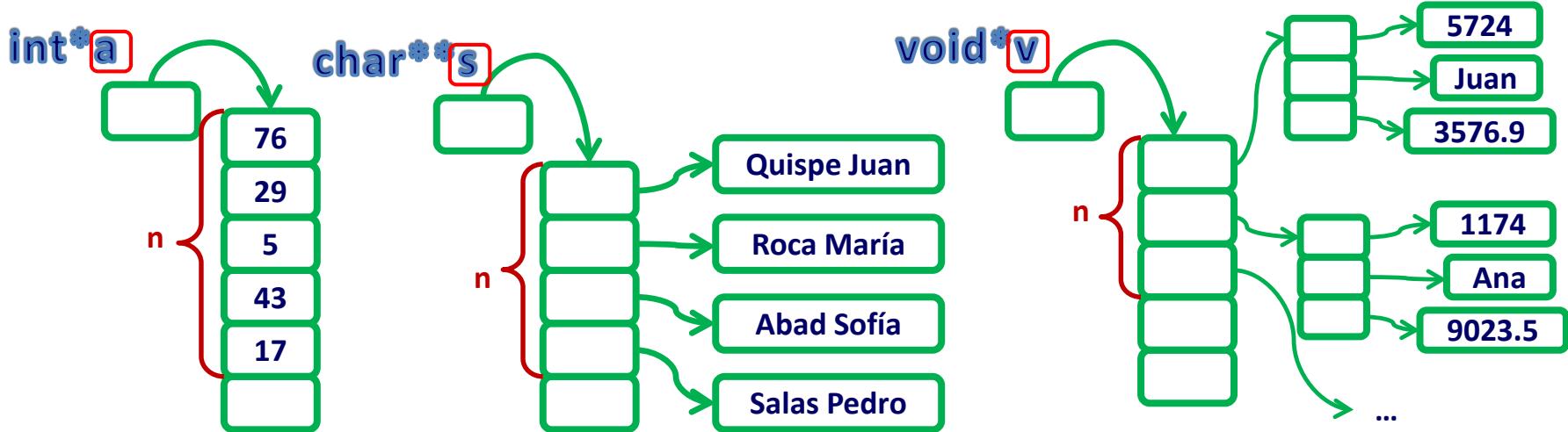
cmp: es el nombre de la función de comparación que regirá el orden de los datos en la ordenación.

ENCABEZADO:

```
void qsort (void*arr, int nd, int size,  
int (*cmp)(const void*,const void*));
```

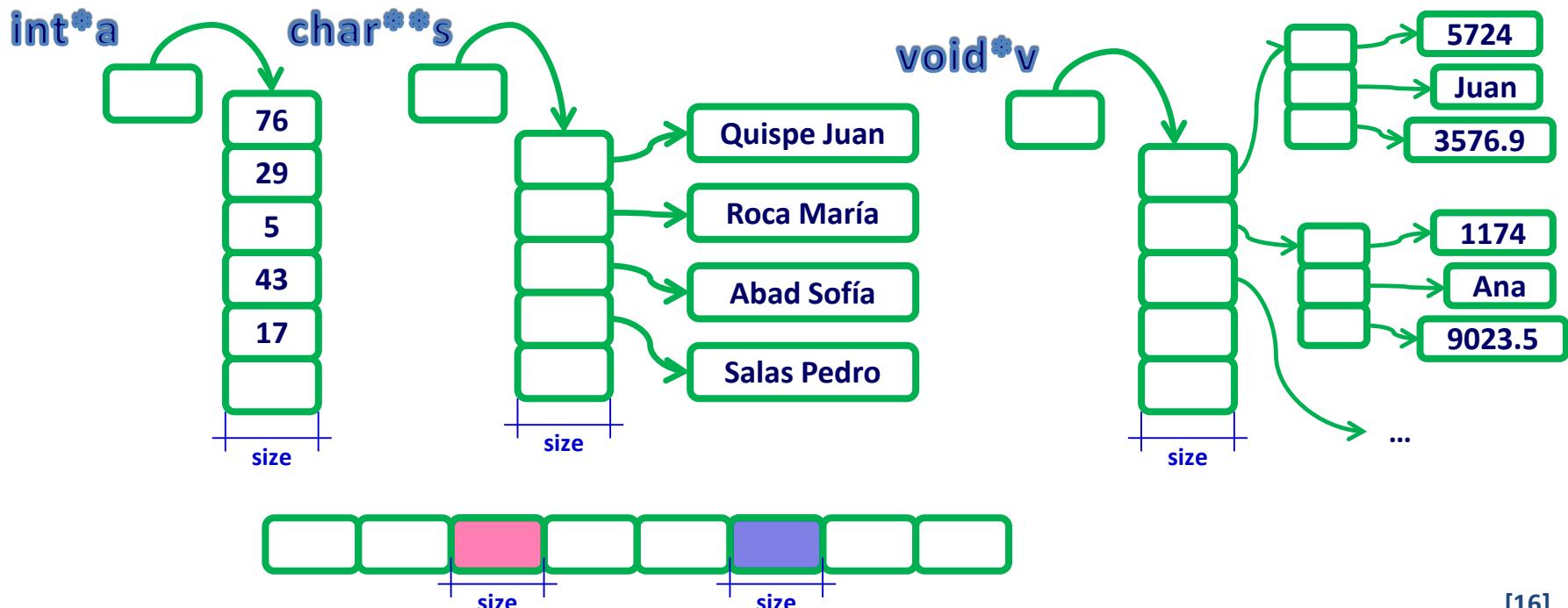
Como trabaja la función:

Imaginemos arreglos de los más simples a los más complejos:
qsort (arr, nd, size, cmp);



Como trabaja la función:

`qsort (arr, nd, size, cmp);`



Como trabaja la función:

qsort (arr, nd, size, cmp);

La función de comparación debe ser de la forma:

int cmp (const void *a, const void *b);

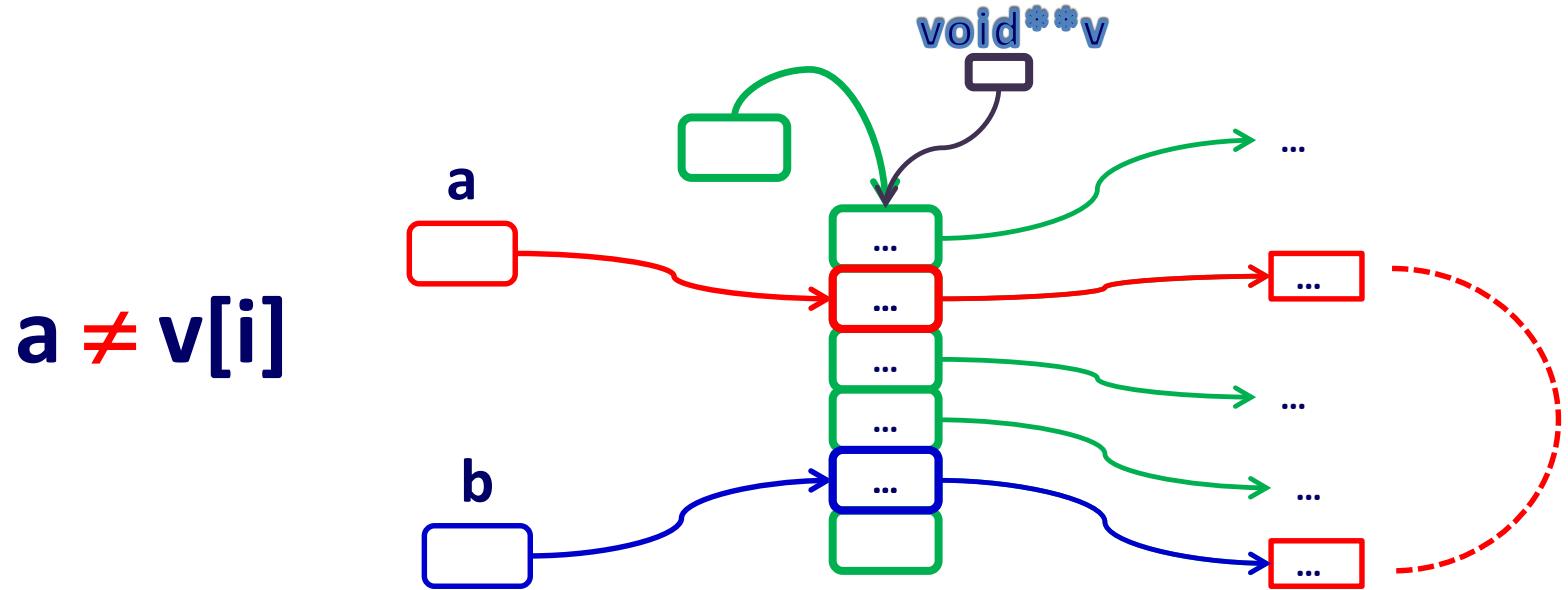
→ 0 si “a” == “b”

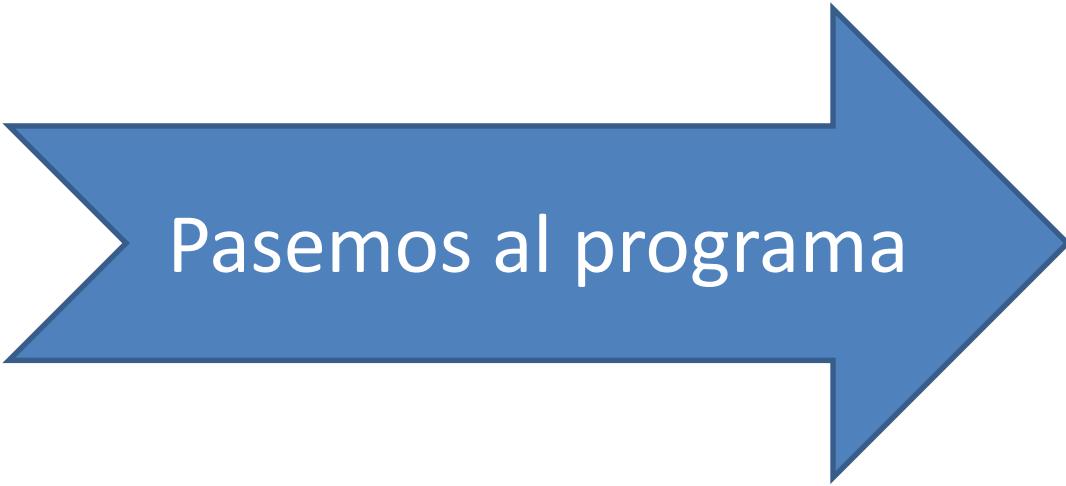
→ >0 si “a” > “b”

→ <0 si “a” < “b”

Como trabaja la función:

int cmp (const void *a, const void *b);

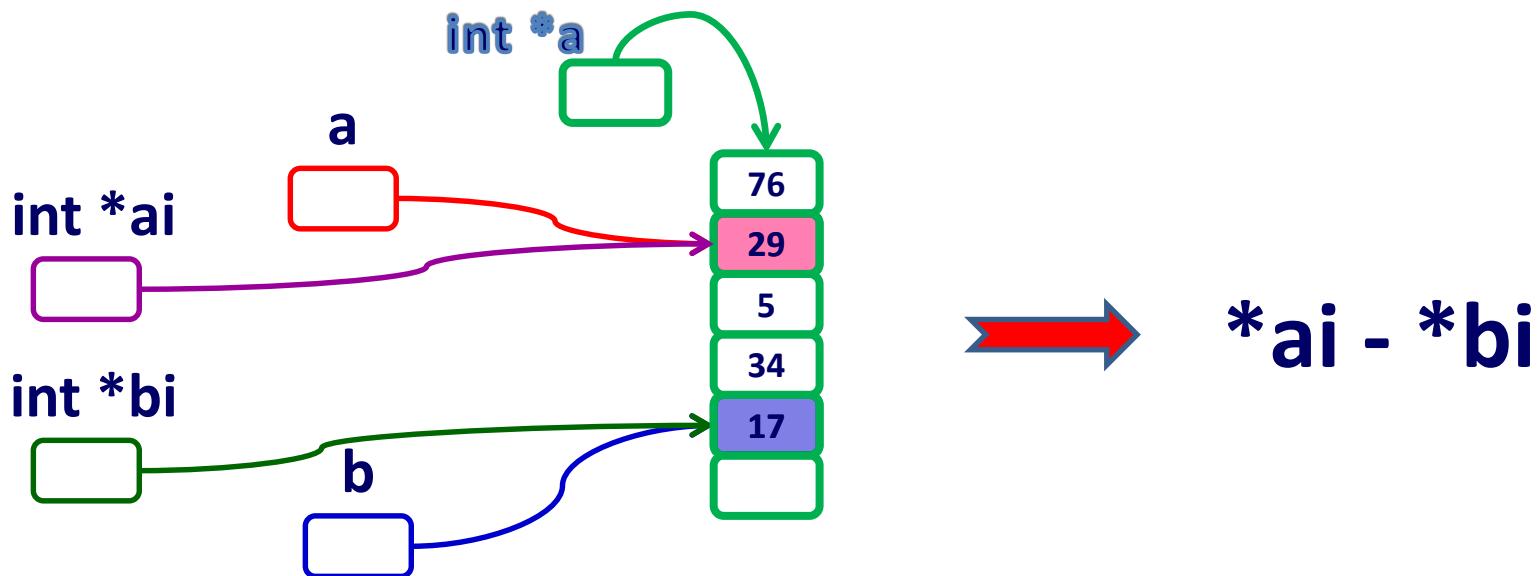




Pasemos al programa

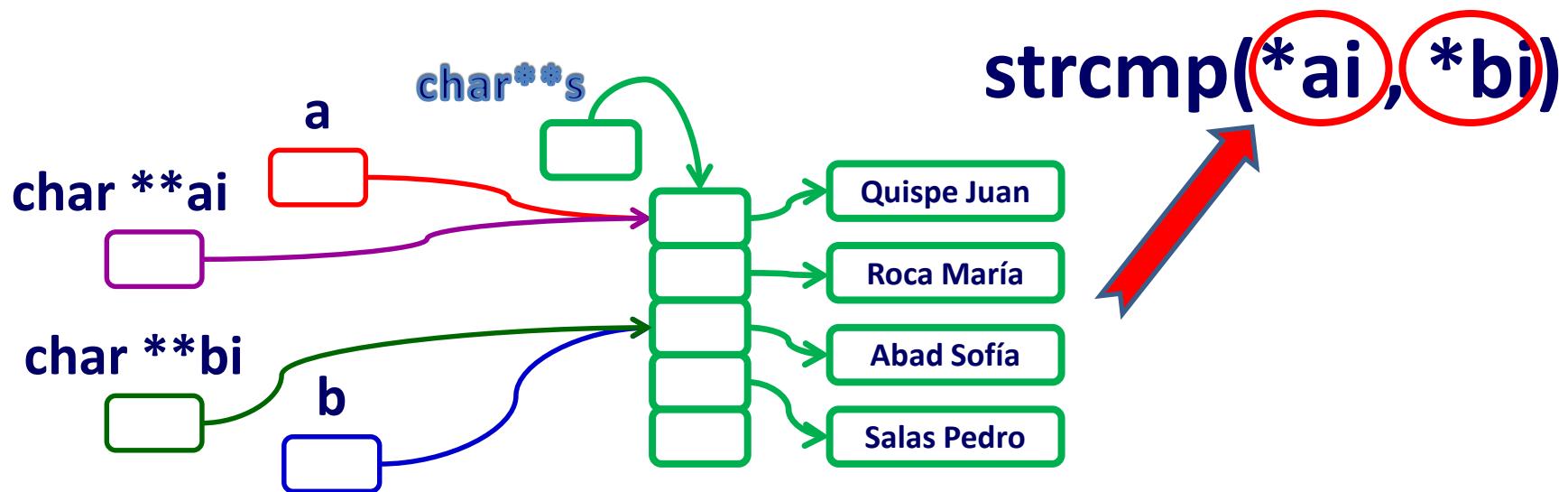
qsort con un arreglo de tipo int:

`int cmp (const void *a, const void *b);`



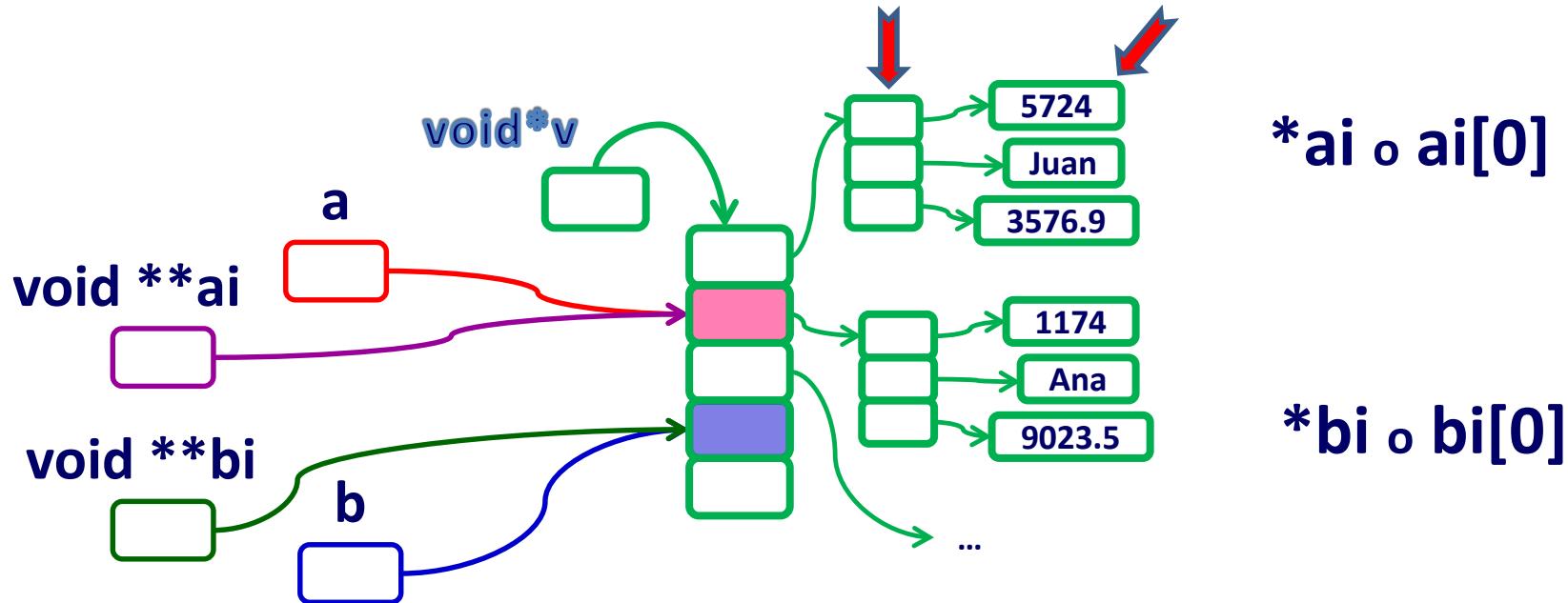
qsort con un arreglo de cadenas:

```
int cmp (const void *a, const void *b);
```



qsort con un arreglo de registros:

```
int cmp (const void *a, const void *b);
```



FUNCIÓN bsearch

ENCABEZADO:

```
void *bsearch (void *llave, void *arr,  
               int nd, int size,  
               int (*cmp)(const void *, const void *));
```

IMPLEMENTACIÓN DE FUNCIONES GENÉRICA

FUNCIÓN DE ORDENACIÓN GENÉRICA

La primera función que implementaremos será una de ordenación, empleando Quick Sort.

Esta función será más simple que la que se define en `cstdlib`, por lo que tendrá pequeñas limitaciones.

INVOCACIÓN:

La función la invocaremos de la siguiente manera:

ordenarG (arr, 0, n-1, cmp);

Donde:

arr: es un arrglo de cualquier tipo^(*).

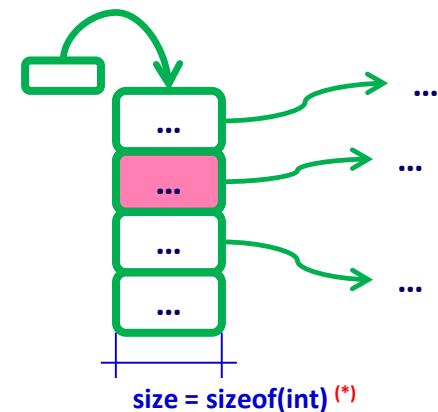
0 y **n-1**: son los límites del arreglo.

cmp: es el nombre de la función de comparación.

(*) recuerde que tendrá limitaciones.

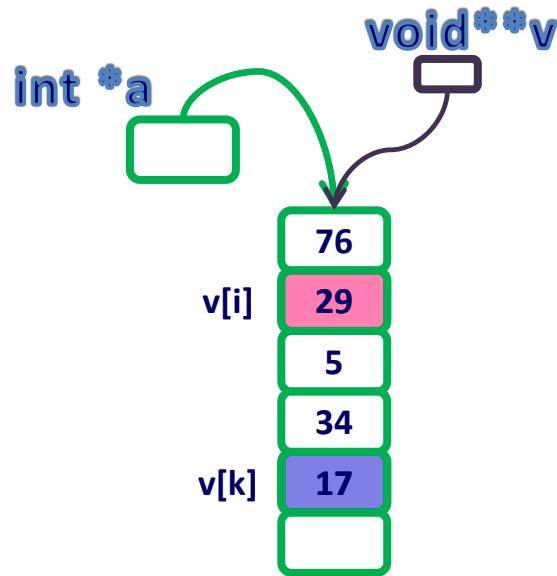
LIMITACIONES:

- 1.- Los elementos del arreglo primario tendrán siempre un tamaño fijo. Este será igual a **sizeof(int)**, lo que permitirá emplear cualquier arreglo de punteros.
- 2.- La función de comparación recibe dos punteros genéricos pero con el contenido de los elementos del arreglo primario



(*) $\text{size} = \text{sizeof}(\text{int}) = \text{sizeof}(\text{char}^*) = \text{sizeof}(\text{char}^{**}) = \text{sizeof}(\text{void}^*) = \text{sizeof}(\text{void}^{**}) = \dots$

ordenarG con un arreglo de tipo int:



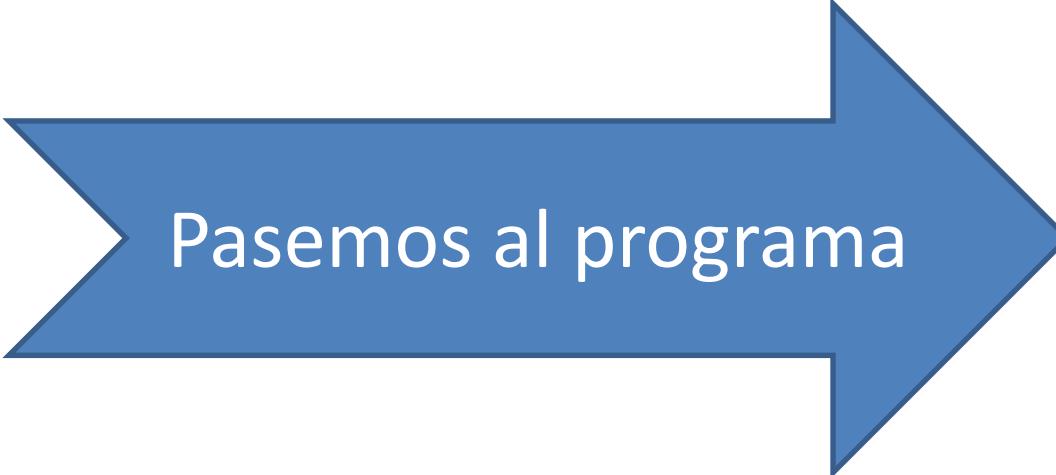
`cmp (v[i], v[k]);`



`void *di`

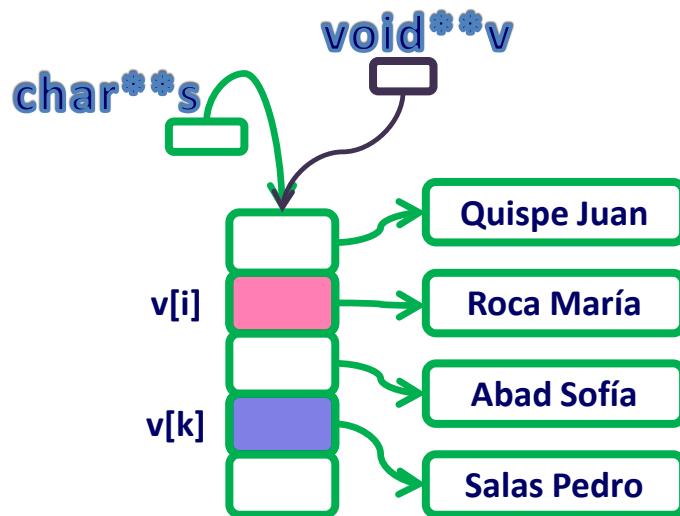
`void *dk`

`int *a = (int*)di int *b = (int*)dk`



Pasemos al programa

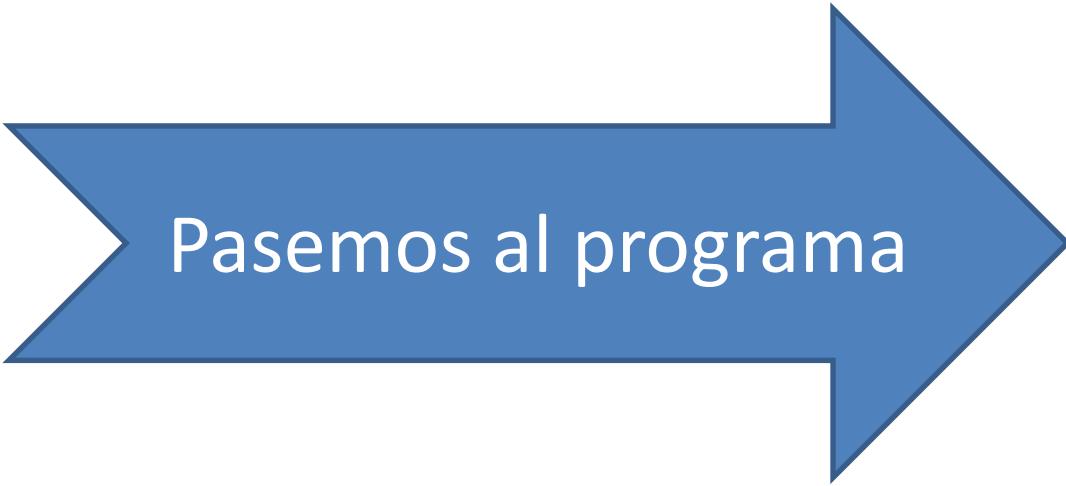
ordenarG con un arreglo de cadenas:



`cmp (v[i], v[k]);`

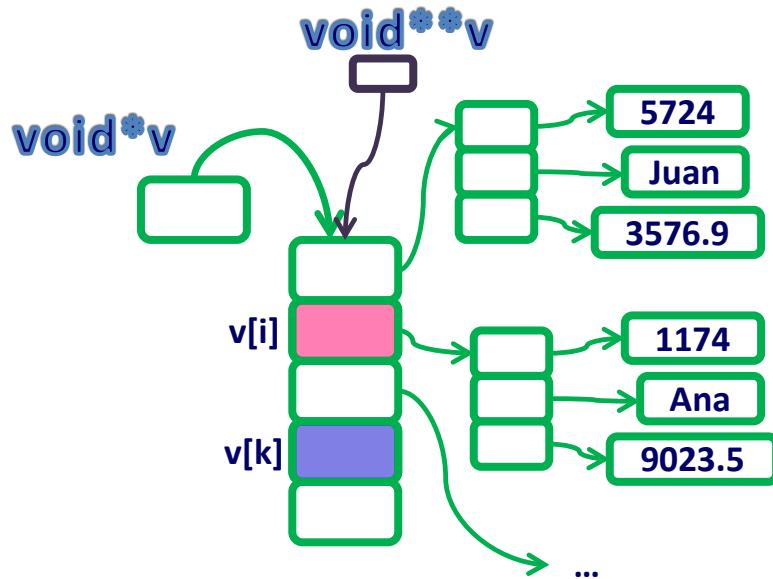
`void *di`

`char* a = (char*)di`



Pasemos al programa

ordenarG con un arreglo de registros:

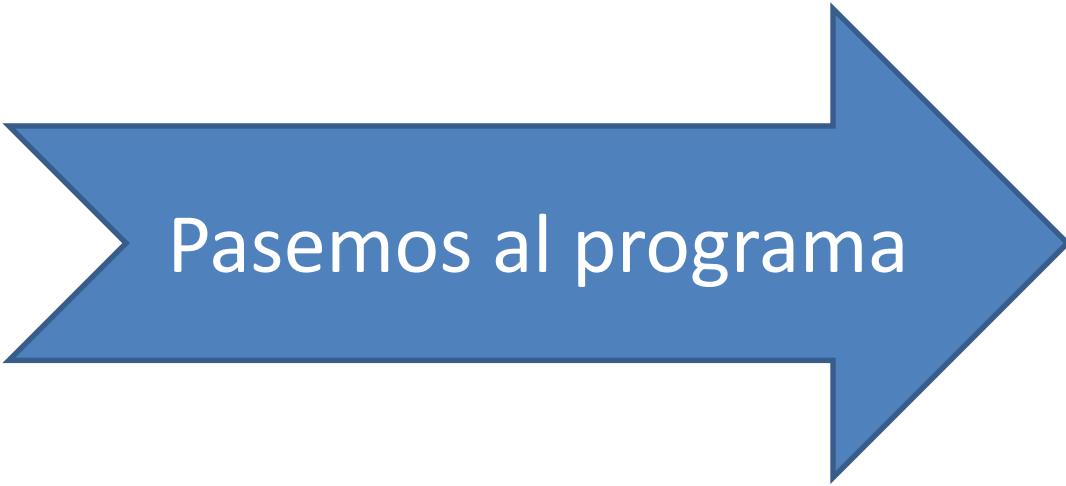


`cmp (v[i], v[k]);`



`void *di`

`void*** regA = (void***)di`

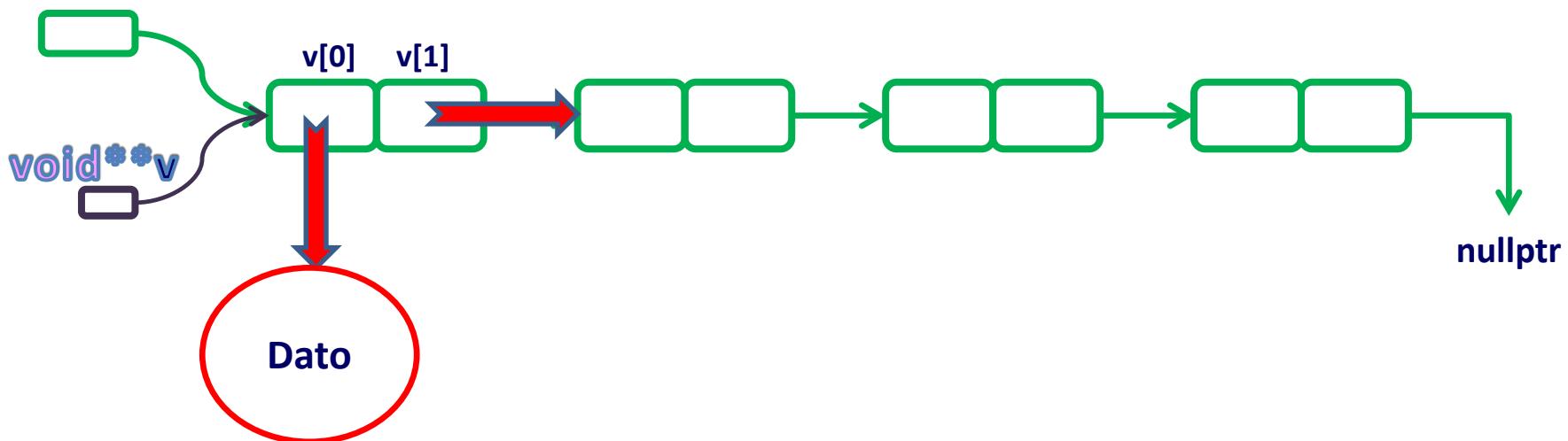


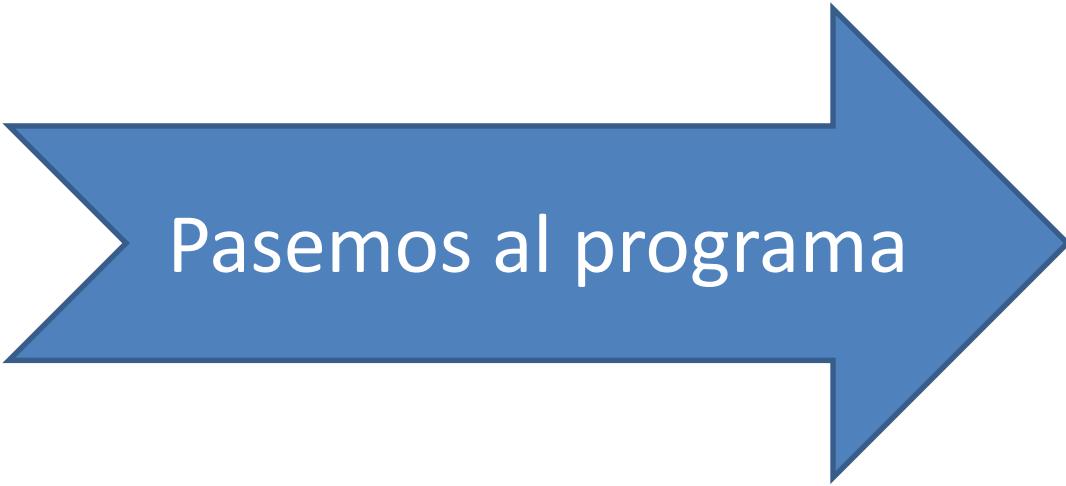
Pasemos al programa

BIBLIOTECA PARA MANIPULAR UNA LISTA LIGADA GENÉRICA

ESTRUCTURA:

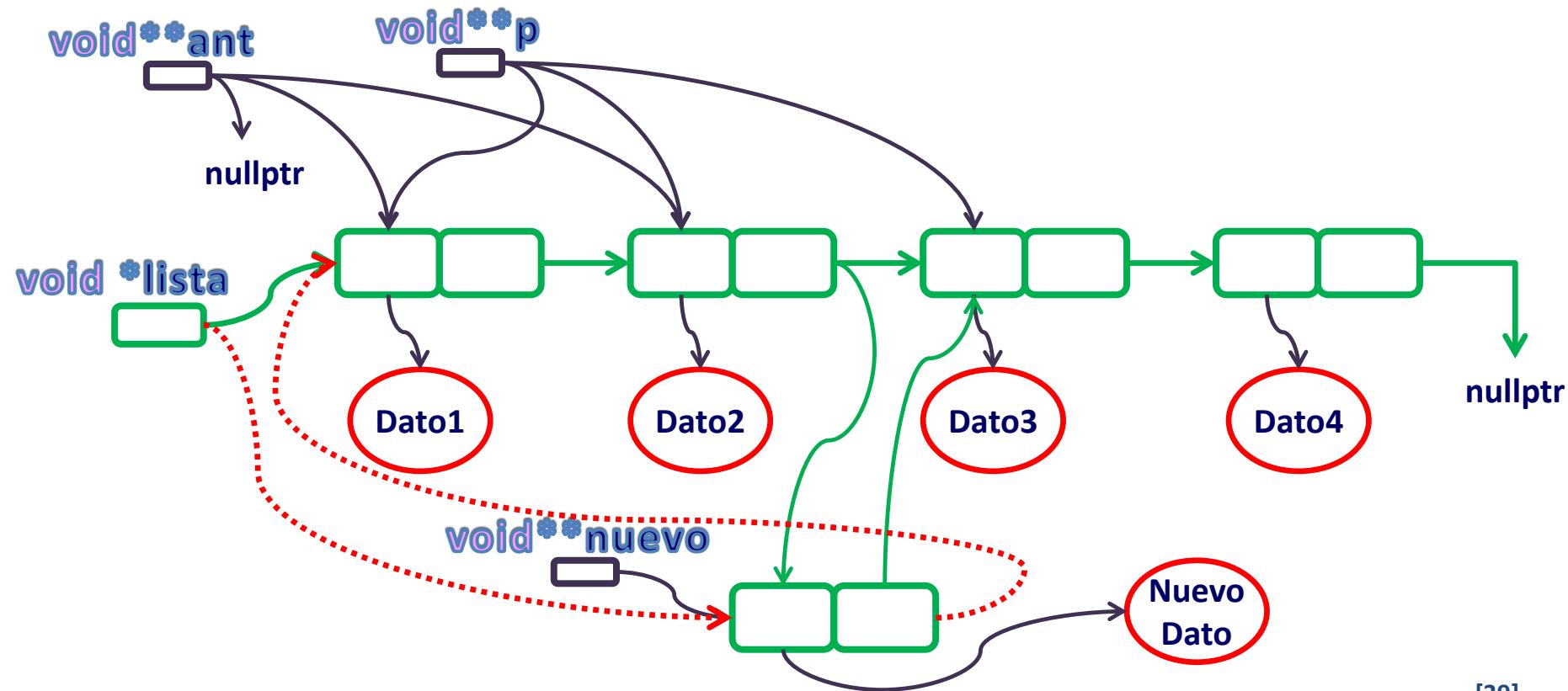
`void *lista`

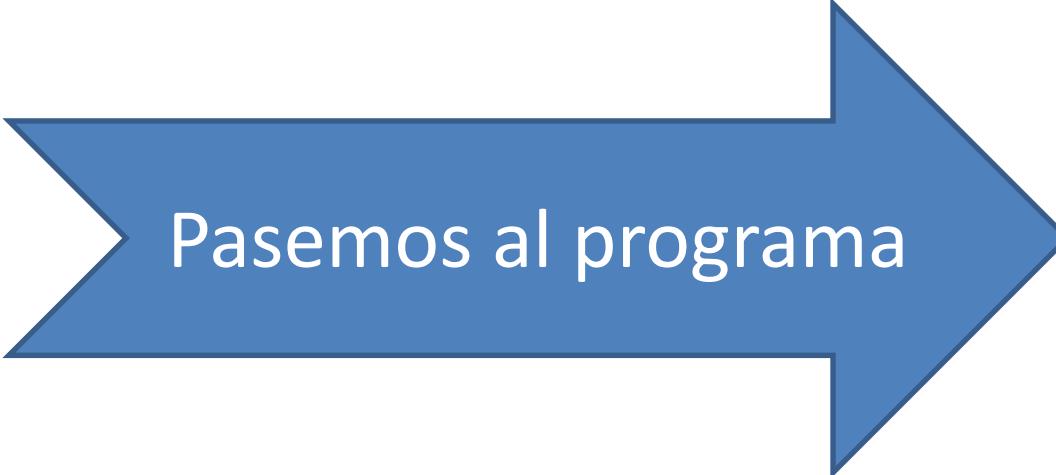




Pasemos al programa

INSERTAR UN DATO EN LA LISTA:





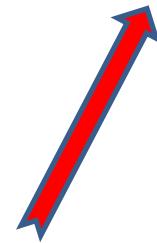
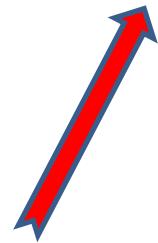
Pasemos al programa

PARÁMETROS DE LA FUNCIÓN main

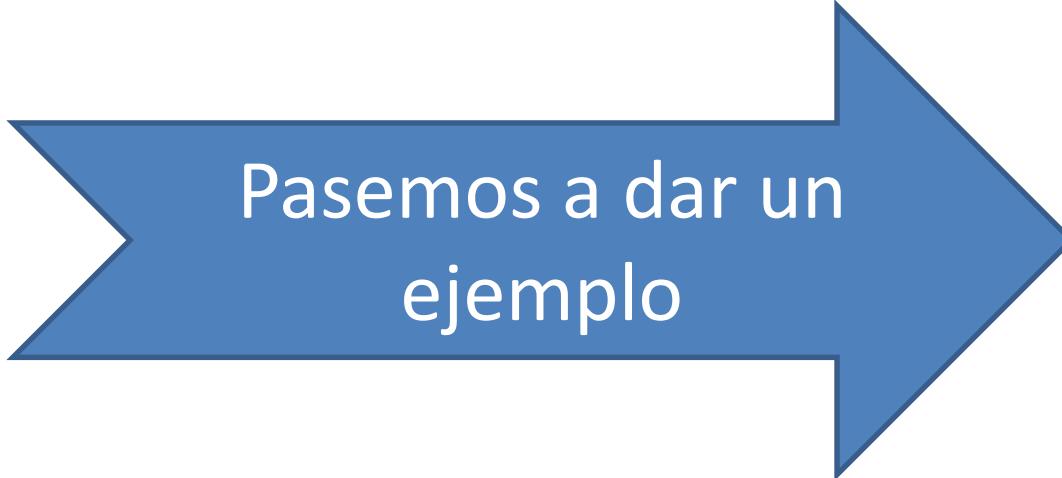
Siempre que empezamos un proyecto observamos esto en el archivo main.cpp:

```
int main (int argc, char** argv) {
```

```
}
```

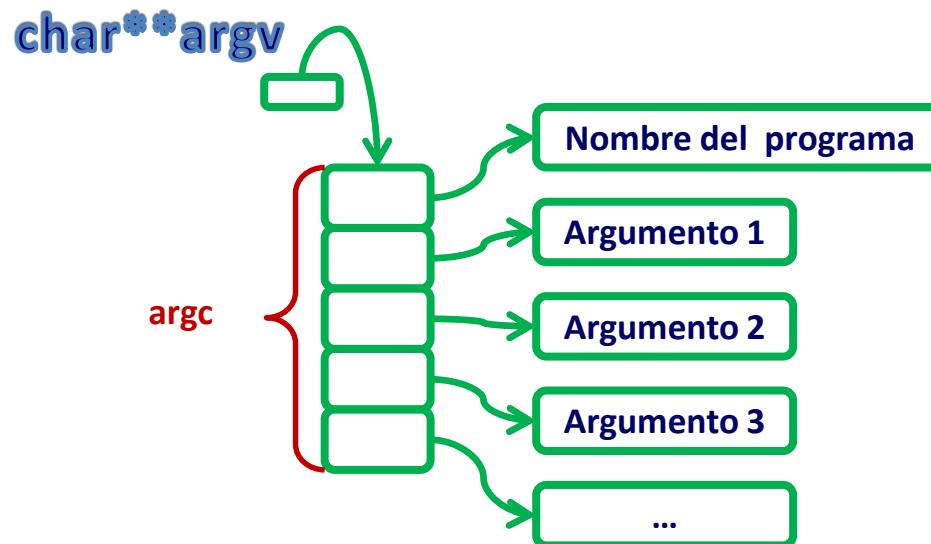


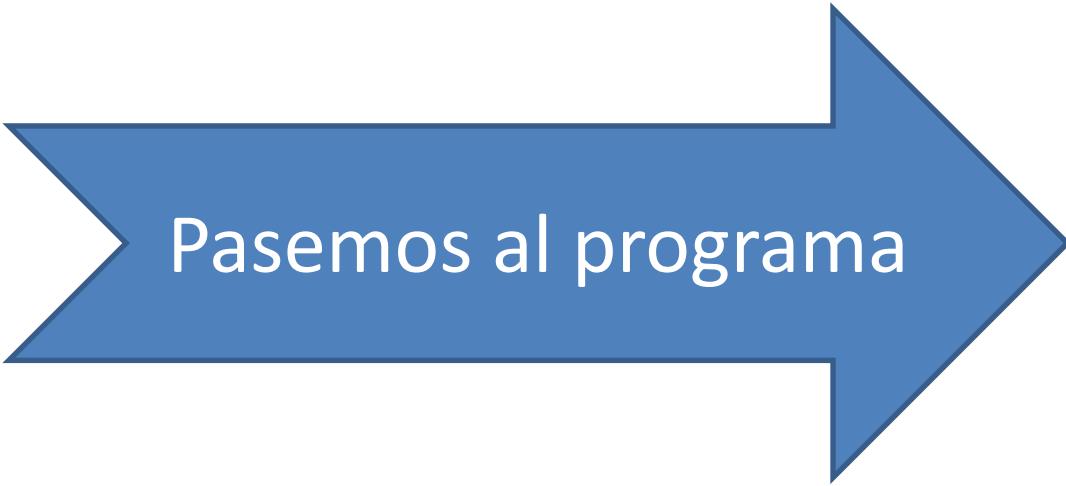
Estos son los parámetros de la función main.cpp:



Pasemos a dar un
ejemplo

`int main (int argc, char** argv)`





Pasemos al programa