



# Programación avanzada en C

Apuntadores y memoria dinámica

# Apuntadores

- Es una variable que **contiene la dirección de memoria** de otra variable.
- Se puede apuntar a cualquier tipo de variable.
- El operador unario **&** **devuelve la dirección de memoria de la variable a la que apunta.**
- El operador de referencia **\*** **devuelve el contenido de la variable a la que apunta.**
- El nombre de un arreglo es siempre un apuntador a la dirección del primer elemento de un arreglo.

# Errores comunes en apuntadores

- No asignar una dirección de memoria válida al apuntador antes de usarlo. Ejemplo:
- No inicializar el apuntador en NULL:

**int \*apuntador= NULL;**

INCORRECTO	CORRECTO
<pre>int *x;  *x = 100;</pre>	<pre>int *x= NULL; int y; x = &amp;y; *x = 100;</pre>

# Proceso 1

➤ `int dato= 2;`  
➤ `int *apunt;`

variables

`dato`

`apunt`

Memoria  
física

Direcc  
memoria

2

100

?

2000

➤ `apunt= &dato;`

variables

`dato`

`apunt`

Memoria  
física

Direcc  
memoria

2

100

100

2000

➤ `*apunt= 5;`

variables

`dato`

`apunt`

Memoria  
física

Direcc  
memoria

5

100

100

2000

```
//variable dato de tipo entera inicializada en 2
int dato=2;

printf("La variable entera dato se inicializa y contiene: %d",dato);
printf("\n\tLa direccion de memoria de dato en hexadecimal es: %p", &dato);
printf("\n\tLa direccion de memoria de dato en decimal es: %d", &dato);

//el tipo del apuntador debe ser igual al tipo de la variable a la q apunta
int *apuntador;
//int *apuntador=NULL;

printf("\n\nLa variable apuntador entera contiene: %d",apuntador);
printf(" xq no se ha inicializado ni apuntado a ninguna variable ");
printf("\n\tLa direccion de memoria de apuntador en hexa es %p",&apuntador);
printf("\n\tLa direccion en memoria de apuntador en decimal es: %d\n", &apuntador);

//el operador unario o monádico & devuelve la
// direccion de memoria de una variable
apuntador = &dato;

printf("\n\nOperador: apuntador = &dato\n");
printf("\n\tahora apuntador contiene la direccion en hexa de dato = %p", apuntador);
// %p muestra la direccion en hexadecimal de una variable o apuntador
printf("\n\tLa direccion en memoria de dato en decimal es: %d", apuntador);

//el operador de indireccion * devuelve el contenido de la
// variable apuntada por el apuntador
*apuntador = 5;

printf("\n\nOperador: *apuntador = 5\n");
printf("\n\tcambia el contenido de la variable a la q apunta, ahora dato = %d",dato);
```

```
I:\2015-1\TEO_PAMN\1-1-arreglos\apuntadores\ejm_apunt\apuntador1.exe

La variable entera dato se inicializa y contiene: 2
    La direccion de memoria de dato en hexadecimal es: 0028FF1C
    La direccion de memoria de dato en decimal es: 2686748

La variable apuntador entera contiene: 70 xq no se ha inicializado ni apuntado a
ninguna variable
    La direccion de memoria de apuntador en hexa es 0028FF18
    La direccion en memoria de apuntador en decimal es: 2686744

Operador: apuntador = &dato

    ahora apuntador contiene la direccion en hexa de dato = 0028FF1C
    La direccion en memoria de dato en decimal es: 2686748

Operador: *apuntador = 5

    cambia el contenido de la variable a la q apunta, ahora dato = 5_
```

Imprimir dirección de memoria de una variable en **hexadecimal**:

```
printf("%p", &apuntador);
```

Imprimir dirección de memoria de una variable en **decimal**:

```
printf("%d", &apuntador);
```

## PROCESO 2

- `int x=1;`
- `int y=2;`
- `int *apuntador;`
- `apuntador = &x;`
- `y = *apuntador;`
- `x = apuntador;`
- `*apuntador = 3;`

Hecho por Huicho :)

```
I:\2015-1\TEO_PAMN\1-1-arreglos\apuntadores\ejm_apunt\apuntador_...
La variable entera X se inicializa y contiene: 1
  La direccion en memoria de X en decimal es: 2686748
La variable entera Y se inicializa y contiene: 2
  La direccion en memoria de Y en decimal es: 2686744
La variable apuntador entera contiene: 8
  xq no se ha inicializado ni apuntado a ninguna variable
  y su direccion inicial de memoria es 2686740

Operador: apuntador = &X
  Ahora apuntador contiene la direccion en memoria de X: 2686748
Operador: y = *apuntador
  la var Y recibe el contenido de la variable X a la q se apunta
  X vale: 1
  Y ahora vale: 1
Operador: x = apuntador
  la var X recibe el contenido de apuntador q es la direccion de X
  La direccion en memoria de X en decimal es: 2686748
  X ahora vale: 2686748
Operador: *apuntador = 3
  cambia el contenido de la variable a la q apunta
  X ahora vale: 3
```

# Memoria dinámica

- Se usa cuando se sabe exactamente cuánta memoria se usará durante la ejecución y así no desperdiciarla pidiendo espacio innecesario en un arreglo estático.
- Se emplea con apuntadores para reservar el espacio en tiempo de la ejecución.
- Pueden pedirse arreglos unidimensionales o matrices.
- Puede liberarse la memoria después de usarla aún cuando no haya finalizado el programa.

# Malloc

- Pedirla a `<stdlib.h>`
- Prototipo: `void * malloc(tamaño en bytes)`
- Para saber el tamaño en bytes de un tipo de dato:  
`sizeof(double);`
- Se puede convertir el puntero void al tipo de dato usado a través de un casting:  
`int *apuntador= NULL;`  
`apuntador= (int *)malloc(10*sizeof(int));`
- Si no hay memoria suficiente el apuntador devuelve NULL.

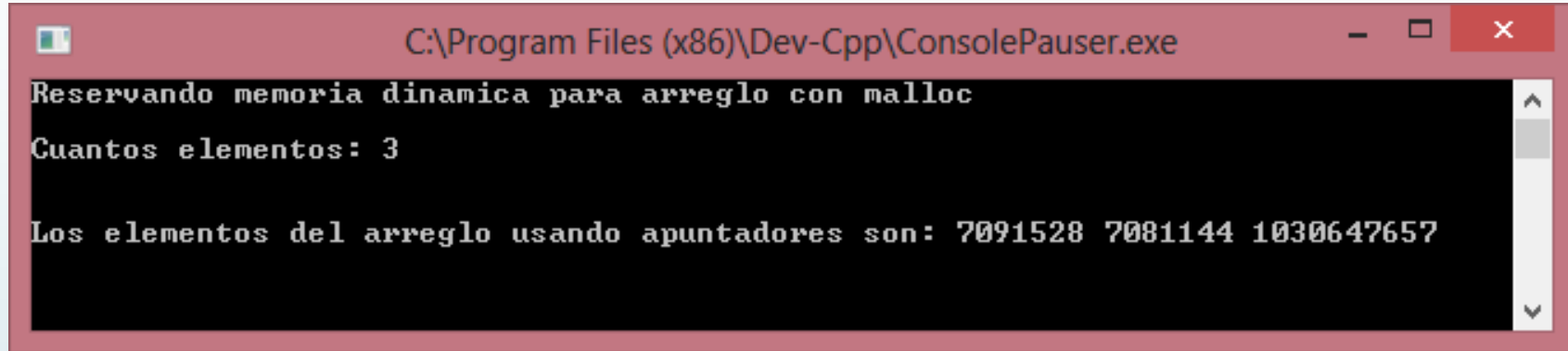


# Calloc

- Pedirla a `<stdlib.h>`
- Hace lo mismo que malloc pero inicializa en 0 los elementos
- Prototipo: `void * calloc(num elementos, tamaño en bytes)`
- Para saber el tamaño en bytes de un tipo de dato:  
`sizeof(double);`
- Se puede convertir el puntero void al tipo de dato usado a través de un casting:  
`int *apuntador= NULL;`  
`apuntador= (int *)calloc(10,sizeof(int));`
- Si no hay memoria suficiente el apuntador devuelve NULL.

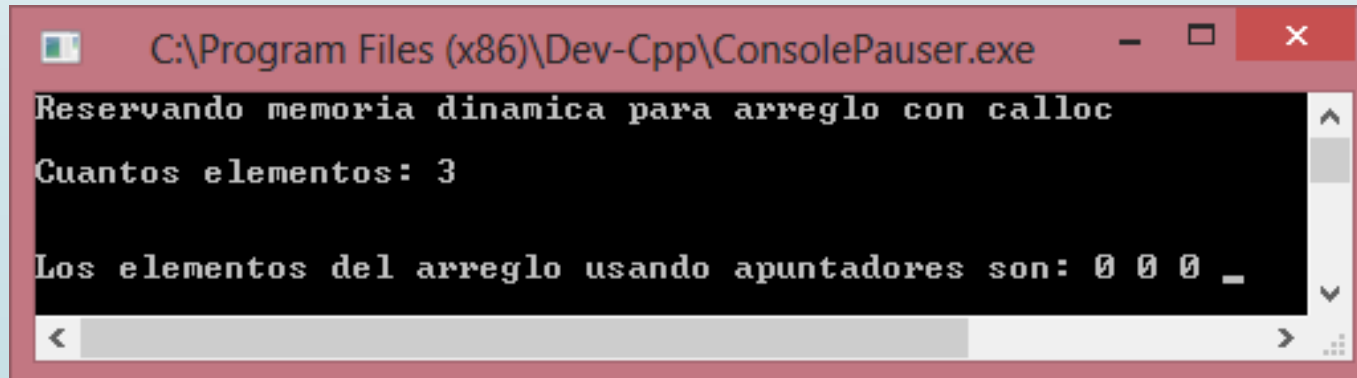
# Diferencias Malloc vs Calloc

## ► Malloc



```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Reservando memoria dinamica para arreglo con malloc
Cuantos elementos: 3
Los elementos del arreglo usando apuntadores son: 7091528 7081144 1030647657
```

## ► Calloc



```
C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Reservando memoria dinamica para arreglo con calloc
Cuantos elementos: 3
Los elementos del arreglo usando apuntadores son: 0 0 0 _
```

# Aritmética de apunadores

- Si `p` es un apunador a un tipo de dato **al sumar 1 obtendrá la siguiente dirección de memoria** de la variable

`*(p + 1)`

`*(p + i)`

`*(p ++)`

`*(p += i)`

Hecho por Huicho :)

```
int arreglo[5]= {1,2,3,4,5};  
int *ap=NULL;  
ap= arreglo;
```

VARIABLES		Memoria física	Dirección de memoria
<code>*ap</code>	<code>arreglo[0]</code>	1	100
<code>*(ap+1)</code>	<code>arreglo[1]</code>	2	101
<code>*(ap+2)</code>	<code>arreglo[2]</code>	3	102
<code>*(ap+3)</code>	<code>arreglo[3]</code>	4	103
<code>*(ap+4)</code>	<code>arreglo[4]</code>	5	104

# Aritmética de apuntadores

## Lectura

- Recorrer arreglo con ciclo for conociendo el inicio, condición e incremento/decremento.

```
for(i=0; i<numeroelem; i++)  
{  
  
    printf("\n\tDame el elemento[%d]: ",i);  
    scanf("%d", (apuntador+i));  
  
}
```

### Arreglo dinámico

```
int *apuntador= NULL;  
apuntador= (int *)malloc(10*sizeof(int));
```

Hecho por Huicho :)

### Arreglo estático

```
int arreglo[5];  
int *apuntador=NULL;  
apuntador= arreglo;
```

# Aritmética de apunadores

## *Impresión*

- Recorrer arreglo con ciclo for conociendo el inicio, condición e incremento/decremento.

```
for(i=0; i<numeroelem; i++)  
{  
    printf("\n\tElemento[%d]: %d", i, *(apuntador+i));  
}
```

### Arreglo dinámico

```
int *apuntador= NULL;  
apuntador= (int *)malloc(10*sizeof(int));
```

Hecho por Huicho :)

### Arreglo estático

```
int arreglo[5];  
int *apuntador=NULL;  
apuntador= arreglo;
```

```

#include <stdio.h>
#include <stdlib.h> //para exit y free

int main(int argc, char * argv[])
{
    int *apuntador= NULL, numelem=0, i=0;

    printf("Reservando memoria dinamica para arreglo con malloc ");
    printf("\n\nCuantos elementos: ");
    scanf("%d", &numelem);

    apuntador = (int *)malloc(numelem*sizeof(int));

    if(apuntador==NULL) //valida si se pudo reservar memoria
    {
        printf("\n Memoria insuficiente, terminar");
        exit(0); //termina la ejecucion justo ahi
    }

    for(i=0; i<numelem; i++) //ciclo para leer
    {
        printf("\n\tDame el elemento[%d]: ",i);
        scanf("%d", &apuntador[i]);
    }

    printf("\n\nLos elementos del arreglo son: ");
    for(i=0; i<numelem; i++) //ciclo para imprimir
    {
        printf("%d ", apuntador[i]);
    }

    free(apuntador); //libera la memoria solicitada
    fflush(stdin); //limpia buffer temporal del teclado
    getchar(); //mantiene la pantalla estatica
    return 0;
}

```

C:\Program Files (x86)\Dev-Cpp\ConsolePauser....

```

Reservando memoria dinamica para arreglo con malloc
Cuantos elementos: 3
    Dame el elemento[0]: 1
    Dame el elemento[1]: 2
    Dame el elemento[2]: 3

Los elementos del arreglo son: 1 2 3

```

```

#include <stdio.h>
#include <stdlib.h> //para exit y free

int main(int argc, char * argv[])
{
    int *apuntador= NULL, numelem=0, i=0;

    printf("Reservando memoria dinamica para arreglo con malloc");
    printf("\n\nCuantos elementos: ");
    scanf("%d", &numelem);

    apuntador = (int *)malloc(numelem*sizeof(int));

    if(apuntador==NULL) //valida si se pudo reservar memoria
    {
        printf("\n Memoria insuficiente, terminar");
        exit(0); //termina la ejecucion justo ahi
    }

    for(i=0; i<numelem; i++) //ciclo para leer
    {
        printf("\n\tDame el elemento[%d]: ",i);
        scanf("%d", (apuntador+i));
    }

    printf("\n\nLos elementos del arreglo son: ");
    for(i=0; i<numelem; i++) //ciclo para imprimir
    {
        printf("%d ", *(apuntador+i));
    }

    free(apuntador); //libera la memoria solicitada
    fflush(stdin); //limpia buffer temporal del teclado
    getchar(); //mantiene la pantalla estatica
    return 0;
}

```

```

int arreglo[5]={1,2,3,4,5};
int *apuntador=NULL;
int valor=0, i=0;
printf("Aritmetica de apuntadores en arreglos\n\n");

printf("\nLa direccion del primer elemento es: %d", &arreglo[0]);

/*apuntador apunta a la direccion del elemento arreglo[0]
apuntador = &arreglo[0];

//otra forma de pasar la direccion
apuntador = arreglo;

printf("\n\nAhora el apuntador tiene la direccion del arreglo: %d", apuntador);

//a valor se le asigna el contenido de apuntador (el elemento 0 de arreglo)
valor = *apuntador;
printf("\n\nEl elemento 0 de arreglo usando apuntador es: %d \n\n", valor);

printf("Recorriendo el arreglo sumando 1 al apuntador: \n");
//recorriendo arreglo con ciclo y apuntador
for(i=0; i<5; i++)
{
    printf("\n\tElemento[%d]: %d", i, *(apuntador+i));
}

printf("\n\nModificando elemento del arreglo por su apuntador: \n");
//cambiar el valor del elemento arreglo[2] con apuntador
*(apuntador+2)= 100;

//recorriendo arreglo con ciclo y apuntador para ver el cambio
for(i=0; i<5; i++)
{
    printf("\n\tElemento[%d]: %d", i, *(apuntador+i));
}

```

```

I:\2015-1\TEO_PAMN\1-1-arreglos\apuntadores\ejm_...
Aritmetica de apuntadores en arreglos

La direccion del primer elemento es: 2686704
Ahora el apuntador tiene la direccion del arreglo: 2686704
El elemento 0 de arreglo usando apuntador es: 1
Recorriendo el arreglo sumando 1 al apuntador:

    Elemento[0]: 1
    Elemento[1]: 2
    Elemento[2]: 3
    Elemento[3]: 4
    Elemento[4]: 5

Modificando elemento del arreglo por su apuntador:

    Elemento[0]: 1
    Elemento[1]: 2
    Elemento[2]: 100
    Elemento[3]: 4
    Elemento[4]: 5_

```

# Pasar de estático a dinámico

```
char nombre[10];
```

```
apuntador= (char *)malloc(longi+1*sizeof(char));
```

## VARIABLES

nombre[0]

nombre[1]

nombre[2]

nombre[3]

nombre[4]

nombre[5]

nombre[6]

nombre[7]

nombre[8]

nombre[9]

## Memoria física

'L'

'U'

'i'

's'

'\0'

'\0'

'\0'

'\0'

'\0'

'\0'

longi= strlen(nombre)

Sumar 1 a longi para  
guardar también el  
carácter de  
terminación

Después de reservar el espacio,  
cómo se copia de un arreglo al otro?

## VARIABLES

apuntador[0]

apuntador[1]

apuntador[2]

apuntador[3]

apuntador[4]

## Memoria física

'L'

'U'

'i'

's'

'\0'



# Matrices - Apuntador de apuntadores

- Se declara el apuntador de apuntadores y variables

```
int  **matriz= NULL;  
int  renglones, columnas, i;
```

- Conociendo el número de renglones y columnas se recorre la matriz
- Se crean los espacios de las columnas para cada renglón.

```
matriz = (int **)malloc(renglones*sizeof(int*));  
for(i=0; i<renglones; i++)  
{  
    matriz[i]= (int *)malloc(columnas*sizeof(int));  
}
```

Hecho por Huicho :)

# Aritmética de apuntadores

## Lectura

```
for(i=0; i<renglones; i++)
{
    for(j=0; j<columnas; j++)
    {
        printf("\n\tDame el elemento[%d][%d]: ",i,j);
        scanf("%d", (*(matriz+i)+j));
    }
}
```

**Apuntador de apuntadores**

```
int **matriz= NULL;
```

# Aritmética de apuntadores

## *Impresión*

```
for(i=0; i<renglones; i++)  
{  
    for(j=0; j<columnas; j++)  
    {  
        printf("%d ", *(*(matriz+i)+j));  
    }  
    printf("\n");  
}
```

**Apuntador de apuntadores**

```
int **matriz= NULL;
```

# Paso de apuntador

FUNCIÓN	LLAMADA A FUNCIÓN	DESCRIPCIÓN
<pre>void leerarreglo(int *arreglo, int numeroelem) {     int i=0;     for(i=0; i&lt;numeroelem; i++)     {         printf("\n\tDame el elemento[%d]: ",i);         scanf("%d", (arreglo+i));     } }</pre>	<pre>int *apuntador= NULL; int numelem=0; ... apuntador= (int *)malloc(numelem*sizeof(int)); ...  leerarreglo(apuntador, numelem);</pre>	<p>RECIBE 2 argumentos:</p> <ul style="list-style-type: none"> <li>-apuntador a entero</li> <li>-número de cuadritos</li> </ul> <p>NO DEVUELVE</p>
<pre>void imprimirarreglo(int *arreglo, int numeroelem) {     int i=0;     printf("\n\nLos elementos del arreglo son: ");     for(i=0; i&lt;numeroelem; i++)     {         printf("%d ", *(arreglo+i));     } }</pre> <p>Hecho por Huicho :)</p>	<pre>int *apuntador= NULL; int numelem=0; ... apuntador= (int *)malloc(numelem*sizeof(int)); ...  imprimirarreglo(apuntador, numelem);</pre>	<p>RECIBE 2 argumentos:</p> <ul style="list-style-type: none"> <li>-apuntador a entero</li> <li>-número de cuadritos</li> </ul> <p>NO DEVUELVE</p>

# Funciones para reservar y liberar

FUNCIÓN	LLAMADA A FUNCIÓN	DESCRIPCIÓN
<pre>int *reserva(int numeroelem) {     int *apunta=NULL;     apunta = (int *)calloc(numeroelem,sizeof(int));      if(apunta==NULL)     {         printf("\n Memoria insuficiente, terminar");         getchar();         exit(0);     }     return apunta; }</pre>	<pre>int *apuntador= NULL; int numelem=0; ... apuntador= reserva(numelem);</pre>	<p>RECIBE 1 argumento: -número de cuadritos</p> <p>DEVUELVE -apuntador a entero</p>
<pre>void libera(int *arreglo) {     free(arreglo); }</pre> <p>Hecho por Huicho :)</p>	<pre>int *apuntador= NULL; int numelem=0; ... libera(apuntador);</pre>	<p>RECIBE 1 argumento: -apuntador a entero</p> <p>NO DEVUELVE</p>

# Paso de apuntador de apuntadores

FUNCIÓN	LLAMADA A FUNCIÓN	DESCRIPCIÓN
<pre>void leer_matriz(int **matriz, int renglones, int columnas) {     int i=0, j=0;     for(i=0; i&lt;renglones; i++)     {         for(j=0; j&lt;columnas; j++)         {             printf("\n\tDame el elemento[%d][%d]: ",i,j);             scanf("%d", &amp;matriz[i][j]);         }     } }</pre>	<pre>int **matriz= NULL; int renglones=0, columnas=0;  leer_matriz(matriz,renglones,columnas);</pre>	<p>RECIBE 3 argumentos:</p> <ul style="list-style-type: none"><li>-apuntador de apuntadores a entero</li><li>-número de renglones</li><li>-número de columnas</li></ul> <p>NO DEVUELVE</p>

# Paso de apuntador de apuntadores

FUNCIÓN	LLAMADA A FUNCIÓN	DESCRIPCIÓN
<pre>void imprimir_matriz(int **matriz, int renglones, int columnas) {     int i=0, j=0;     printf("\n\nLos elementos del arreglo son: \n\n");     for(i=0; i&lt;renglones; i++)     {         for(j=0; j&lt;columnas; j++)         {             printf("%d ", matriz[i][j]);         }         printf("\n");     } }</pre>	<pre>int **matriz= NULL; int renglones=0, columnas=0;  imprimir_matriz(matriz,renglones,columnas);</pre>	<p>RECIBE 3 argumentos:</p> <ul style="list-style-type: none"><li>-apuntador de apuntadores a entero</li><li>-número de renglones</li><li>-número de columnas</li></ul> <p>NO DEVUELVE</p>

# Paso de apuntador de apuntadores

## FUNCIÓN PARA LEER

```
void leer_matriz(int **matriz, int renglones, int columnas)
{
    int i=0, j=0;
    for(i=0; i<renglones; i++)
    {
        for(j=0; j<columnas; j++)
        {
            printf("\n\tDame el elemento[%d][%d]: ", i, j);
            scanf("%d", (*(matriz+i)+j));
        }
    }
}
```

## FUNCIÓN PARA IMPRIMIR

```
void imprimir_matriz(int **matriz, int renglones, int columnas)
{
    int i=0, j=0;
    printf("\n\nLos elementos del arreglo son: \n\n");
    for(i=0; i<renglones; i++)
    {
        for(j=0; j<columnas; j++)
        {
            printf("%d ", (*(matriz+i)+j));
        }
        printf("\n");
    }
}
```



# Función para reservar memoria de apuntador de apuntadores

## FUNCIÓN

```
int **reserva(int renglones, int columnas)
{
    int **matriz=NULL;
    int i=0;
    matriz = (int **)calloc(renglones,sizeof(int*));

    if(matriz==NULL)
    {
        printf("\n Memoria insuficiente, terminar");
        getchar();
        exit(0);
    }

    for(i=0; i<renglones; i++)
    {
        matriz[i]= (int *)calloc(columnas,sizeof(int));
    }
    return matriz;
}
```

Hecho por Huicho :)

## LLAMADA A FUNCIÓN

```
int **matriz= NULL;
int renglones=0, columnas=0;

...

matriz= reserva(renglones, columnas);
```

## DESCRIPCIÓN

RECIBE 2  
argumentos:

- número de renglones
- número de columnas

DEVUELVE

- apuntador de apuntadores a entero

# Función para liberar memoria de apuntador de apuntadores

FUNCIÓN	LLAMADA A FUNCIÓN	DESCRIPCIÓN
<pre>void libera(int **matriz, int renglones) {     int i=0;     for(i=0; i&lt;renglones; i++)     {         free(matriz[i]);     }     free(matriz); }</pre>	<pre>int **matriz= NULL; int renglones=0, columnas=0;  ...  libera(matriz, renglones);</pre>	<p>RECIBE 2 argumentos:</p> <ul style="list-style-type: none"><li>-apuntador de apuntadores a entero</li><li>-número de renglones</li></ul> <p>NO DEVUELVE</p>

# Apuntadores a estructuras

```
#include<stdio.h>

int main(int argc, char*argv[])
{
    struct alumno
    {
        char nombre[50];
        int edad;
    };

    struct alumno *apunt_alu= NULL; //apuntador a estructura alumno

    struct alumno alu; //variable de estructura alumno

    apunt_alu= &alu; //se asigna la direccion de memoria al apuntador

    printf("Este programa guarda datos de alumno en una estructura con apuntadores \n\n");

    printf("\nDame tu nombre: ");
    //gets(alu.nombre);
    gets(apunt_alu->nombre);

    printf("\nDame tu edad: ");
    //scanf("%d", &alu.edad);
    scanf("%d", &apunt_alu->edad);

    printf("\n\nTe llamas %s y tu edad es %d", apunt_alu->nombre, apunt_alu->edad);

    getchar();
    getchar();
    return 0;
}
```

- Declaración de apuntador a estructura:  
**struct caja \*apuntador= NULL;**
- Declaración de variable de estructura:  
**struct caja var\_estructura;**
- Paso de dirección de estructura a apuntador:  
**apuntador= &var\_estructura**
- Acceso a cuadrado de la caja apuntada:  
**apuntador -> cuadrado**

```
#include<stdio.h>

struct alumno
{
    char nombre[50];
    int edad;
};

void leer_estruct(struct alumno *a1);
void imprime_estruct(struct alumno *a);

int main(int argc, char*argv[])
{

    struct alumno *apunt_alu= NULL; //apuntador a estructura alumno

    struct alumno alu; //variable de estructura alumno

    apunt_alu= &alu; //se asigna la direccion de memoria al apuntador

    printf("Este programa guarda datos de alumno en una estructura con apuntadores \n\n");

    leer_estruct(apunt_alu);
    imprime_estruct(apunt_alu);

    getchar();
    getchar();
    return 0;
}

void leer_estruct(struct alumno *a1)
{
    printf("\nDame tu nombre: ");
    gets(a1->nombre);
    printf("\nDame tu edad: ");
    scanf("%d", &a1->edad);
}

void imprime_estruct(struct alumno *a)
{
    printf("\n\nTe llamas %s y tu edad es %d", a->nombre, a->edad);
}
```

## Apuntadores a estructuras y funciones