



# TEMA 8

## Tipos de datos estructurados

Grado en Ingeniería Eléctrica

Grado en Ingeniería Electrónica Industrial y Automática

**Escuela Politécnica de Ingeniería de Ferrol**

<https://www.udc.es/epef>

8.1.- Vectores

8.2.- Cadenas de caracteres

8.3.- Matrices multidimensionales

8.4.- Reserva dinámica de vectores y matrices

8.5.- Estructuras

## 8.1.- Vectores

- También denominados *arrays*
- Es una colección de variables del mismo tipo que tienen un nombre común
- Cada elemento del vector se distingue de los demás por su índice (orden dentro del array)
- Un array se caracteriza por:
  - El tipo base (int, float, etc.)
  - El tamaño de la dimensión

- Sintaxis:

**tipo** nombre[tamaño]

Ejemplos:

```
int vector1[5];
```

```
float vector2[20];
```

## 8.1.- Vectores

- Cada elemento del array se identifica con su nombre seguido de un índice entre corchetes
- **El primer elemento del array es el de índice 0**
- Ejemplos:

`int vector1[6];` → vector de 6 elementos (del 0 al 5)

Primer elemento del vector: `vector1[0]`

Último elemento del vector: `vector1[5]`

Se le pueden asignar valores: `vector1[3] = 4;`

## 8.1.- Vectores

### ■ Ejemplo:

```
#include "stdio.h"

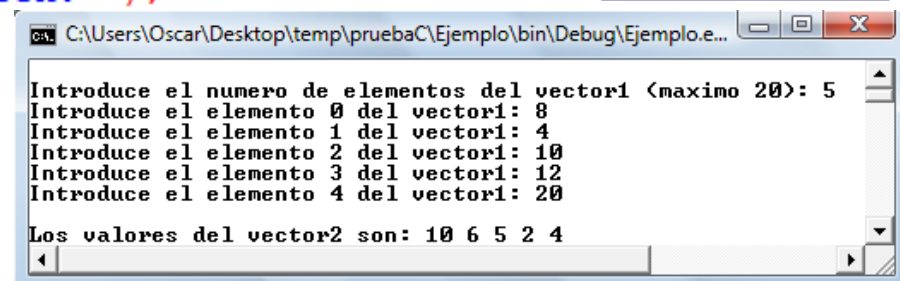
void main()
{
    int vector1[20], vector2[20], i, elementos;

    printf("\nIntroduce el numero de elementos del vector1 (maximo 20): ");
    scanf("%i", &elementos);

    for(i=0; i<elementos; i=i+1)
    {
        printf("Introduce el elemento %i del vector1: ", i);
        scanf("%i", &vector1[i]);
        vector2[elementos-i-1] = vector1[i]/2;
    }

    printf("\nLos valores del vector2 son: ");
    for(i=0; i<elementos; i++)
        printf("%i ", vector2[i]);
}
```

**RESULTADO:**



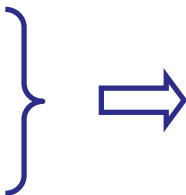
```
C:\Users\Oscar\Desktop\temp\pruebaC\Ejemplo\bin\Debug\Ejemplo.e...
Introduce el numero de elementos del vector1 (maximo 20): 5
Introduce el elemento 0 del vector1: 8
Introduce el elemento 1 del vector1: 4
Introduce el elemento 2 del vector1: 10
Introduce el elemento 3 del vector1: 12
Introduce el elemento 4 del vector1: 20

Los valores del vector2 son: 10 6 5 2 4
```

## 8.1.- Vectores

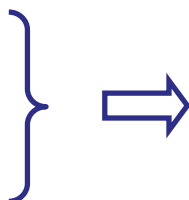
- Normas para el uso de arrays estáticos:
  - El tamaño se fija al hacer el programa y NO se puede variar durante la ejecución
  - NO se pueden usar variables para definir el tamaño de un array:

```
int N=5;  
int Vector[N];
```

 **ERROR**

- Sí se puede, y es aconsejable, usar constantes para definir el tamaño del array

```
#define MAXLEN 5  
int Vector[MAXLEN];
```

 **CORRECTO**

## 8.1.- Vectores

- Normas para el uso de arrays estáticos:
  - NO es posible utilizar más elementos de los que tiene el array, pero sí menos
  - Aunque los arrays tienen un tamaño fijo → NO hay que operar siempre con todos los datos
  - NO existe una instrucción para leer y escribir un array completo (excepto para un tipo especial) → hay que leer elemento a elemento

## 8.1.- Vectores

- Precauciones al utilizar arrays:
  - El lenguaje C **no comprueba los límites de los arrays**
  - Es responsabilidad del programador el no sobrepasar los límites
  - Si se sobrepasan los límites → se obtienen valores sin sentido
  - Ejemplo:

```
int b;  
int Datos[20];
```

```
b = Datos[25];    → No da error pero le asigna a b un  
                  valor desconocido
```



## 8.1.- Vectores

- Ejemplo: lectura de los datos de un vector y cálculo del máximo de ellos

```
#include "stdio.h"
#define MAXNUM 30
void main()
{
    int i, max, datos[MAXNUM], numdatos;

    printf("\nIntroduzca el numero de elementos (max. 30): ");
    scanf("%i", &numdatos);
    if (numdatos >= 1 && numdatos < MAXNUM)
    {
        for (i=0; i<numdatos; i++)
        {
            printf("\nIntroduzca el numero %i: ", i);
            scanf("%i", &datos[i]);
        }

        max = datos[0];
        for (i=1; i<numdatos; i++)
        {
            if (datos[i] > max)
                max = datos[i];
        }
        printf("\nEl maximo valor introducido es: %i", max);
    }
    else
        printf("\nError, el numero de elementos es incorrecto");
}
```

*Lectura de los datos del vector*

*Se inicializa el valor del máximo al del primer elemento del vector*

*Se calcula el máximo comparando con el resto de elementos del vector*

## 8.1.- Vectores

- Al igual que las variables, los vectores pueden inicializarse al ser declarados

- Ejemplo:

```
int A[5] = {1, 2, 3, 4, 5};
```

- Si se indican menos valores que los declarados:
  - Los valores se asignan a las primeras posiciones
  - El resto quedan inicializados a 0
- Si se especifican más valores de los declarados se produce un error
- Se puede omitir el tamaño al inicializar (lo calcula el compilador):

```
int B[] = {1, 1, 2, 1, 0, 3};
```

## 8.1.- Vectores

- Paso de arrays a funciones:
  - Siempre se realiza por referencia (mediante puntero)
  - Por ello no es necesario usar el operador & ya que el nombre del array es puntero a la primera posición del mismo
  - Sí es necesario usar el operador & cuando se pasa un único elemento del array
  - El parámetro del array en la cabecera de la función siempre se declara sin tamaño
  - Si es una matriz se deben especificar todas las dimensiones menos la primera

## 8.1.- Vectores

- Ejemplo: cálculo de la media de un vector

```
#include "stdio.h"
#define MAX 30

float vmedia(float vector[], int n)
{
    float resultado, suma=0;
    int i;

    for (i=0 ; i<n ; i++)
        suma = suma + vector[i];
    resultado = suma/n;

    return resultado;
}
```

```
void main()
{
    int numero, i;
    float media, vec[MAX];

    printf("\nCuantos elementos?: ");
    scanf("%d",&numero);

    if (numero > 0 && numero <= MAX)
    {
        for (i=0 ; i<numero ; i++)
        {
            printf("\nElemento %i: ", i+1);
            scanf("%f",&vec[i]);
        }
        media = vmedia(vec, numero);
        printf("\nLa media es: %.2f", media);
    }
    else
        printf("\nError, numero incorrecto");
}
```

## 8.2.- Cadenas de caracteres

- Es un array que cumple dos condiciones:
  - Sus elementos son de tipo char (caracteres)
  - Uno de ellos es el carácter 0 (NULL o carácter de código ASCII 0)
- El carácter 0 sirve para indicar el final de cadena:
  - No se tiene en cuenta al operar con la cadena
- **Importante:** No confundir el carácter 0 (carácter nulo, código ASCII 0) con el carácter '0' (código ASCII 48)
  - En muchas referencias bibliográficas se indica como \0 para evitar la confusión
- Cuando se hace una lectura ya se coloca un \0 al final de la cadena

## 8.2.- Cadenas de caracteres

- Declaración de una cadena:

```
char nombre[tamaño];
```

- Ejemplo:

```
char nombre[30]; → utilizables 29 (el otro para el \0)
```

- Inicialización de cadenas:

```
char alumno[19] = "Manuel López López";
```

```
char alumno[] = "Manuel López López";
```

```
char alumno[50] = "Manuel López López";
```

## 8.2.- Cadenas de caracteres

- Lectura de cadenas con **scanf**:
  - NO se usa la & antes de la cadena de caracteres (la variable de la cadena es un puntero al primer elemento)
  - La función para de leer al encontrar un espacio en blanco, un tabulador, o un retorno de carro
  - Para que el scanf lea cualquier carácter hasta el final de línea:
    - Usar el modificador [**^\n**]
  - Ejemplos:

```
void main()
{
    char palabra[15];
    scanf("%14s", palabra);
}
```

```
void main()
{
    char frase[50];
    scanf("%49[^\n]s", frase);
}
```

## 8.2.- Cadenas de caracteres


- Escritura de cadenas con **printf**:

```
void main()
{
    char cadena[] = "Mi texto";
    float dato = 12.5;
    printf("Mi cadena : %s y el número: %f", cadena, dato);
}
```



## 8.2.- Cadenas de caracteres

- Utilización y acceso a las cadenas:
  - Se puede acceder elemento a elemento como en cualquier array
  - El final de la cadena se alcanza cuando el valor es igual a 0 (ó '\0')
  - NO se puede asignar una cadena a otra:

```
void main()
{
    char cad1[40], cad2[40];
    cad2 = "Fundamentos de informática";
    cad1 = cad2;  INCORRECTO
}
```

## 8.2.- Cadenas de caracteres

- Ejemplo: contar el número de espacios en blanco en una cadena

```
#include "stdio.h"

void main()
{
    char cadena[200];
    int i, blancos=0;

    printf("\nTeclea una cadena: ");

    scanf("%[^\\n]199s", cadena);

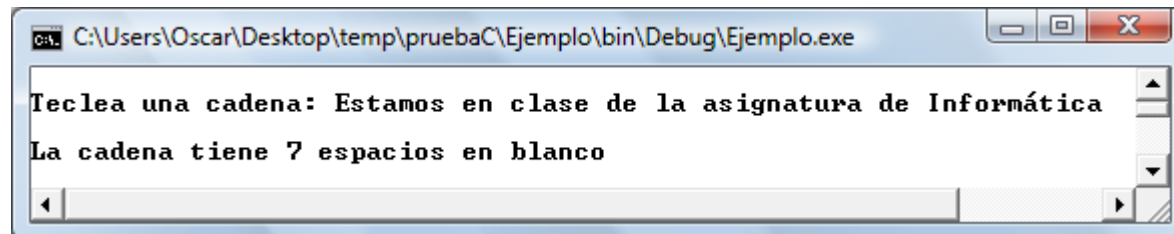
    for (i=0; cadena[i] != '\\0'; i++) /* También vale: cadena[i] != 0 */
        if (cadena[i] == ' ')
            blancos++;

    printf("\nLa cadena tiene %d espacios en blanco", blancos);
}
```

Lectura de **toda** la cadena:

- El `[^\\n]` indica que se leerá cualquier carácter hasta un retorno de carro
- El 199 se emplea para especificar que como máximo se leerán 199 caracteres (si se teclean más no se tienen en cuenta)

### RESULTADO:



## 8.2.- Cadenas de caracteres

- El lenguaje C proporciona funciones para operar con cadenas de caracteres (en string.h):
  - **strlen(cadena)** : devuelve un número entero que indica la longitud de la cadena indicada
  - **strcmp(cad1, cad2)** : compara dos cadenas carácter a carácter (distingue minúsculas y mayúsculas) y devuelve como resultado:
    - 0     si son iguales
    - < 0   si cad1 < cad2
    - > 0   si cad1 > cad2
  - **stricmp(cad1, cad2)** : igual que strcmp pero ignorando la diferencia entre minúsculas y mayúsculas

## 8.2.- Cadenas de caracteres

- Funciones de cadenas (en string.h):
  - **strupr(cadena)** : convierte a mayúsculas una cadena
  - **strlwr(cadena)** : convierte a minúsculas una cadena
  - **strcpy(destino, origen)** : copia el contenido de una cadena en otra
  - **strcat(cad1, cad2)** : pega la segunda cadena al final de la primera
- Las funciones en string.h SÓLO sirven para cadenas de caracteres, NO para arrays de números

## 8.2.- Cadenas de caracteres

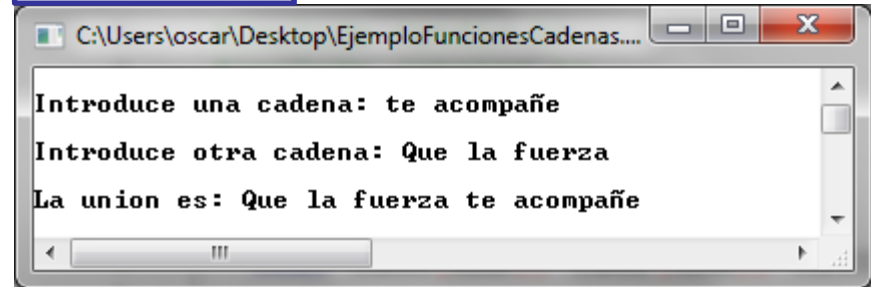
### ■ Ejemplo:

```
#include <stdio.h>
#include <string.h>
#define MAX 50

void main()
{
    char cad1[MAX], cad2[MAX], cad3[MAX];

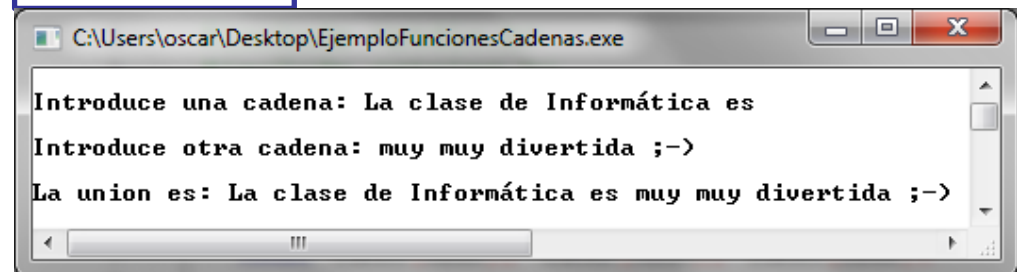
    printf("\nIntroduce una cadena: ");
    scanf("%[^\n]49s", cad1);
    getchar(); <-----
    printf("\nIntroduce otra cadena: ");
    scanf("%[^\n]49s", cad2);
    if (strlen(cad1) > strlen(cad2))
    {
        strcpy(cad3, cad1);
        strcat(cad3, " ");
        strcat(cad3, cad2);
    }
    else
    {
        strcpy(cad3, cad2);
        strcat(cad3, " ");
        strcat(cad3, cad1);
    }
    printf("\nLa union es: %s\n", cad3);
}
```

### RESULTADO:



*Esta función lee un único carácter. Necesaria para leer el retorno de carro que queda del scanf anterior. Si no se usa no lee adecuadamente la cadena el siguiente scanf*

### RESULTADO:



## 8.3.- Matrices multidimensionales

- Sintaxis para la declaración de una matriz:

tipo nombre[tamaño1][tamaño2] ... [tamañoN]

- Ejemplos:

float Matriz[6][4];

int Datos[4][5][6][8];

- Ejemplo de asignación de un dato en un elemento de una matriz:

Matriz[1][2] = 3.2;

- Igual que en los arrays unidimensionales el índice de cada dimensión comienza en 0
- Las normas de uso de los vectores son aplicables también en este caso

## 8.3.- Matrices multidimensionales

- Ejemplo: calcular la matriz transpuesta de una dada (parte 1)

```
#include <stdio.h>
#define MAX_FIL 50
#define MAX_COL 50
```

```
void imprimir_matriz(int matriz[][MAX_COL], int filas, int columnas)
{
    int i, k;

    for(i=0; i<filas; i++)
    {
        for(k=0; k<columnas; k++)
            printf("%i ", matriz[i][k]);
        printf("\n");
    }
}
```

*Cuando la función recibe una matriz  
no es necesario indicar la primera  
dimensión (el número máximo de filas)*

*Se le pasa como argumento el número  
de filas que se usan en la matriz*

*Se le pasa como argumento el  
número de columnas que se  
usan en la matriz*

*Se muestran en pantalla los datos de una matriz (la que se le  
pase como argumento a la función) en forma matricial*

*El primer índice especifica la  
fila de la matriz a la que  
queremos acceder*

*El segundo índice se usa para  
indicar la columna de la matriz*

## 8.3.- Matrices multidimensionales

- Ejemplo: calcular la matriz transpuesta de una dada (parte 2)

```
void main()
{
    int i,k, Matriz[MAX_FIL][MAX_COL], Traspuesta[MAX_COL][MAX_FIL];
    int filas, columnas;

    printf("Introduce el numero de filas de la matriz: ");
    scanf("%i", &filas);
    printf("Introduce el numero de columnas de la matriz: ");
    scanf("%i", &columnas);

    if (filas<=MAX_FIL && columnas<=MAX_COL)
    {
        for(i=0; i<filas; i=i+1)
            for(k=0; k<columnas; k=k+1)
            {
                printf("Matriz[%i][%i] = ", i, k);
                scanf("%i", &Matriz[i][k]);
                Traspuesta[k][i] = Matriz[i][k];
            }
        printf("\nMatriz =\n");
        imprimir_matriz(Matriz,filas,columnas);
        printf("\nTraspuesta =\n");
        imprimir_matriz(Traspuesta,columnas,filas);
    }
    else
        printf("\nERROR, el numero de filas o columnas es incorrecto.");
}
```

*Lectura de los datos de la matriz y cálculo de los elementos correspondientes en la transpuesta*

*Se llama a la función para imprimir la matriz original (que está en la variable Matriz)*

*Se llama de nuevo a la función para imprimir la matriz transpuesta. Se le pasa como primer dato la variable columnas, ya que en la transpuesta ese dato indica las filas, y como segundo argumento la variable filas.*



## 8.3.- Matrices multidimensionales

- Ejemplo de ejecución del programa anterior:

```
Introduce el numero de filas de la matriz: 2
Introduce el numero de columnas de la matriz: 3
Matriz[0][0] = 1
Matriz[0][1] = 2
Matriz[0][2] = 3
Matriz[1][0] = 4
Matriz[1][1] = 5
Matriz[1][2] = 6

Matriz =
1 2 3
4 5 6

Traspuesta =
1 4
2 5
3 6
```

## 8.3.- Matrices multidimensionales

- Inicialización de una matriz:

```
int Matriz[2][3] = {3, 4, 1, 6, 7, 8};
```

Resultado: 
$$\begin{pmatrix} \mathbf{3} & \mathbf{4} & \mathbf{1} \\ \mathbf{6} & \mathbf{7} & \mathbf{8} \end{pmatrix}$$

- Se puede omitir la primera dimensión pero no las demás:

```
int C[][2] = {-1, -2, -3, -4, -5};
```

Resultado: 
$$\begin{pmatrix} -\mathbf{1} & -\mathbf{2} \\ -\mathbf{3} & -\mathbf{4} \\ -\mathbf{5} & \mathbf{0} \end{pmatrix}$$

## 8.3.- Matrices multidimensionales

- Arrays de cadenas (matriz de caracteres):
  - Se pueden usar para tener mensajes en una tabla
- Ejemplo:

```
char meses[][15] =
```

```
{“enero”, “febrero”, “marzo”,  
  “abril”, “mayo”, “junio”, “julio”, “agosto”,  
  “septiembre”, “octubre”, “noviembre”,  
  “diciembre” };
```

## 8.3.- Matrices multidimensionales

### ■ Ejemplo:

```
#include <stdio.h>
#include <string.h>
#define MAX 15
void main()
{
    char meses[][MAX] = {"enero", "febrero", "marzo", "abril", "mayo", "junio", "julio",
                        "agosto", "septiembre", "octubre", "noviembre", "diciembre"};

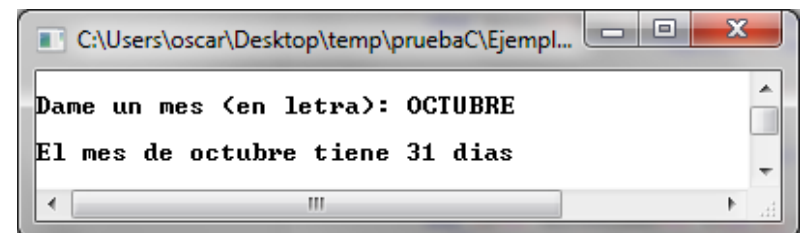
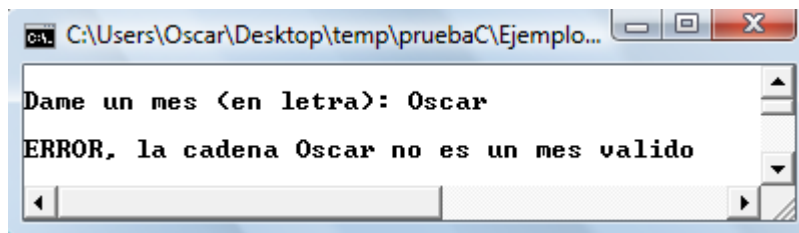
    int i, encontrado, duracion[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    char mes[MAX];

    printf("\nDame un mes (en letra): ");
    scanf("%[^\\n]14s", mes);

    for (i=0, encontrado=0; i<12 && encontrado==0; i++)
        if (strcmp(mes, meses[i])==0)
            encontrado = 1;

    if (encontrado == 1)
        printf("El mes de %s tiene %d dias\\n", strlwr(mes), duracion[i-1]);
    else
        printf("\\nERROR, la cadena %s no es un mes valido\\n", mes);
}
```

Se le resta 1 a la variable i porque cuando finaliza el bucle anterior se le incrementa 1 antes de salir (ya que el i++ se ejecuta antes que la condición)



## 8.4.- Reserva dinámica de memoria

- Reserva dinámica de memoria:
  - Es posible reservar dinámicamente la memoria para un vector o una matriz en tiempo de ejecución
    - Definiéndolo como un puntero
  - Es necesario reservar memoria antes de que los elementos del vector/matriz sean procesados
  - Funciones relacionadas:
    - void \* **malloc** (int bytes): solicita un bloque de memoria del tamaño que se indique (en **bytes**)
    - void **free** (void \*puntero): libera memoria obtenida con el *malloc*
    - int **sizeof**(tipo): devuelve el tamaño que ocupa en bytes el tipo indicado

## 8.4.- Reserva dinámica de memoria para vectores

- Ejemplo con reserva dinámica de memoria de un vector: cálculo del producto de todos los elementos de un vector

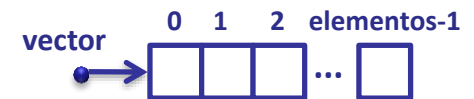
```
#include <stdlib.h>
#include <stdio.h>

void main()
{
    int *vector, elementos, i, producto=1;

    printf("Introduce el numero de elementos del vector: ");
    scanf("%i", &elementos);

    vector = (int *) malloc(elementos * sizeof(int));
    if (vector == NULL)
        printf("\nError, no se pudo reservar memoria.");
    else
    {
        for (i=0; i<elementos; i++)
        {
            printf("Introduce el elemento %i: ", i);
            scanf("%i", &vector[i]);
            producto = producto * vector[i];
        }
        free(vector);
    }
    printf("\nProducto de los elementos del vector = %i\n", producto);
}
```

*Reserva dinámica de memoria para tantos elementos como los indicados*



Se define el vector como un puntero

*Si el puntero devuelto apunta a NULL no se pudo reservar la memoria solicitada*

*Al finalizar con el vector se libera la memoria usada*

## 8.4.- Reserva dinámica de memoria para vectores

- Ejemplo con reserva dinámica de memoria de un vector: cálculo del producto de todos los elementos de un vector (con funciones)

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

int producto_vec(int *vector, int elementos)
{
    int i, producto=1;

    for (i=0; i<elementos; i++)
    {
        printf("Introduce el elemento %i: ", i);
        scanf("%i", &vector[i]);
        producto = producto * vector[i];
    }
    return producto;
}
```

## 8.4.- Reserva dinámica de memoria para vectores

- Ejemplo con reserva dinámica de memoria de un vector: cálculo del producto de todos los elementos de un vector (con funciones)

```
void main()
{
    int *vector, elementos, producto;

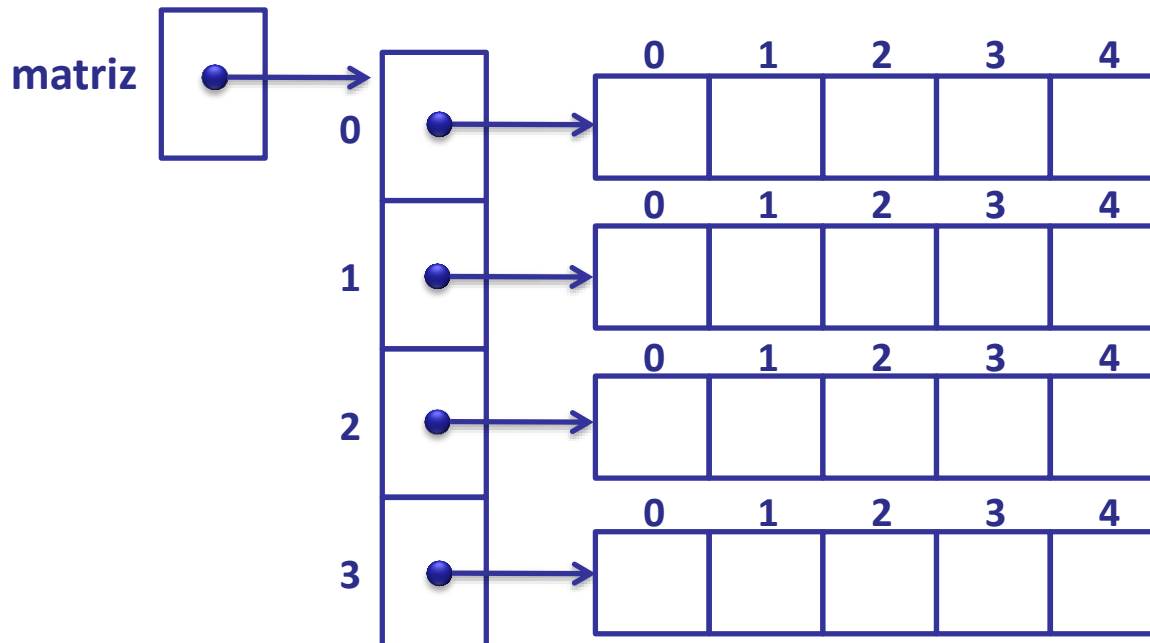
    printf("Introduce el numero de elementos del vector: ");
    scanf("%i", &elementos);

    vector = (int *) malloc(elementos * sizeof(int));
    if (vector == NULL)
        printf("\nError, no se pudo reservar memoria.");
    else
    {
        producto = producto_vec(vector, elementos);
        free(vector);
    }
    printf("\nProducto de los elementos del vector = %i\n", producto);
}
```



## 8.4.- Reserva dinámica de memoria para matrices

- No se gestiona la matriz como una sucesión de elementos contiguos, como en un vector, sino como ***un vector dinámico de vectores dinámicos***



- Ejemplo de declaración: `int **matriz;`  
(*puntero a un puntero de enteros*)

## 8.4.- Reserva dinámica de memoria para matrices

- Ejemplo de reserva de memoria para una matriz (sin control de errores total):

```
int ** reserva_matriz_int (int nfil, int ncol)
{
    int **matriz, i;

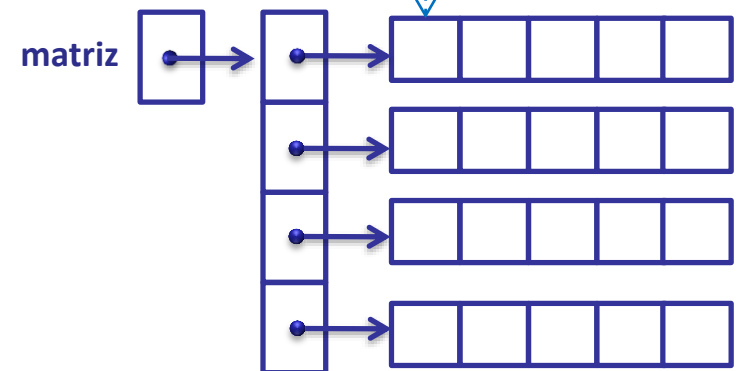
    matriz = (int **) malloc(nfil * sizeof (int *));

    if (matriz != NULL)
    {
        for (i = 0; i < nfil; i++)
            matriz[i] = (int *) malloc (ncol * sizeof(int));

        return matriz;
    }
}
```

*En primer lugar se reserva espacio para el vector que apuntará a cada una de las filas de la matriz*

*En cada una de las iteraciones del bucle se reserva espacio para una fila diferente*



## 8.4.- Reserva dinámica de memoria para matrices

- Ejemplo de liberación de memoria para una matriz:

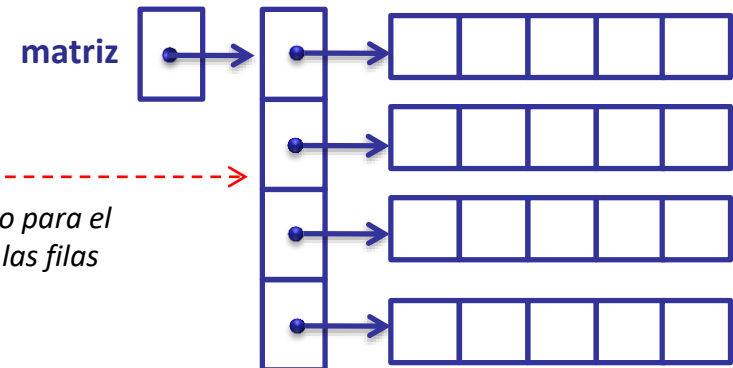
```
void libera_matriz(int **matriz, int nfil)
{
    int i;

    if (matriz != NULL)
    {
        for (i = 0; i < nfil; i++)
            if (matriz[i] != NULL)
                free(matriz[i]);

        free(matriz);
    }
}
```

*En cada una de las iteraciones del bucle se libera espacio para una fila diferente*

*Al final se libera espacio para el vector que apuntaba a las filas*



## 8.4.- Reserva dinámica de memoria para matrices

- Ejemplo de uso de las funciones de reserva y liberación de memoria:

```
void main()
{
    int **matriz, fil, col;

    do {
        printf("Introduce filas y columnas: ");
        scanf("%i%i", &fil, &col);
    } while (fil < 1 || col < 1);

    matriz = reserva_matriz_int (fil, col);
    if (matriz != NULL)
    {
        /* Aquí estarían las operaciones a realizar con la matriz */

        libera_matriz(matriz, fil);
    }
    else
        printf("Error: no se pudo reservar memoria.");
}
```

## 8.5.- Estructuras

- Estructuras (o registros):
  - Es una estructura de datos que contiene varios componentes elementales más simples
  - Cada componente se denomina *campo*
  - Diferencias con un array
    - Puede almacenar tipos distintos
    - Cada elemento se identifica por un nombre (no un número)
  - Se definen con la palabra clave **struct**
  - Ejemplo:

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

## 8.5.- Estructuras

- Ejemplos de declaración de variables y arrays de estructuras:

```
struct alumno a;  
struct alumno a, b;  
struct alumno alumnos[100];
```

- Para acceder a los campos del registro:

- Con el nombre de la variable un punto y el campo:

- Ejemplos:

```
a.edad = 20;  
strcpy(a.nombre, "Juan López");  
printf("Nombre: %s, Edad: %d", a.nombre, a.edad);  
alumnos[3].edad = 20;
```

## 8.5.- Estructuras

- Ejemplo: lectura de los datos (nombre y edad) de unos alumnos y cálculo de la media de edad

```
#include "stdio.h"
#define MAX 50
struct tipo_alumno
{
    char nombre[40];
    int edad;
};

void main()
{
    struct tipo_alumno alumno[MAX];
    int numero, i, suma = 0;
    float media;
```

*(continúa en la siguiente columna)*

```
printf("\nIntroduce el numero de alumnos: ");
scanf("%i", &numero);
for (i=0; i<numero; i++)
{
    printf("\nAlumno %d", i+1);
    printf("\nNombre?: ");
    scanf("%[^\\n]39s", alumno[i].nombre);
    printf("Edad?: ");
    scanf("%d", &alumno[i].edad);
}
for (i=0; i<numero; i++)
{
    printf("\nNombre: %s", alumno[i].nombre);
    suma = suma + alumno[i].edad;
}
media = (float) suma/numero;
printf("\nLa media de edad es: %.2f", media);
}
```