

# PROGRAMACIÓ 1

Grado en Ingeniería Informática e I2ADE

## Tema 6

### Tipos de datos estructurados: Arrays



Dept. de Ciència de la Computació i Intel·ligència *a*rtificial  
Dpto. de Ciencia de la Computación e Inteligencia *a*rtificial



Universitat d'Alacant  
Universidad de Alicante

# Índice

2

1. Tipos de datos estructurados
2. El tipo array
3. Arrays unidimensionales
4. Arrays bidimensionales

# 1. Tipos de datos estructurados

3

- Una variable de tipo de dato estructurado, al contrario que un tipo de dato simple, puede almacenar más de un valor a la vez.
- Los tipos de datos estructurados en C que se van a estudiar en la asignatura son los siguientes:
  - Tipo **Array**. Todos los valores que almacena una variable de tipo array deben ser del mismo tipo de dato.
  - Tipo **Registro**. Una variable de tipo registro puede almacenar valores de distinto tipo de dato.
- Ejemplo: Consideremos una variable **z** que almacenará los números premiados en la bonoloto. Por lo tanto se almacenarán 6 valores cada vez:  
**z** = (1, 4, 6, 24, 13, 2);  
**z** = (3, 9, 12, 15, 23, 27);

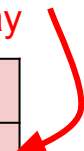
## 2. El tipo *array*

4

- Es una estructura en la que se almacena una colección finita, homogénea y ordenada de datos (elementos).
  - **Finita**: debe determinarse cuál será el número máximo de elementos que podrán almacenarse en el array.
  - **Homogénea**: todos los elementos deben ser del mismo tipo.
  - **Ordenada**: se puede determinar cuál es el  $n$ -ésimo elemento del array.
- Para referirse a un determinado elemento de un array se deberá utilizar un índice (encerrado entre corchetes `[]`) que especifique su posición relativa en el array.

Posición:	0	1	2	3	4	5	6
Elemento:	2	4	6	8	10	14	23
Forma de acceder:	<code>x[0]</code>	<code>x[1]</code>	<code>x[2]</code>	<code>x[3]</code>	<code>x[4]</code>	<code>x[5]</code>	<code>x[6]</code>

último elemento  
del array



**Nota:** En lenguaje C, el primer elemento del array se encuentra en la **posición** (índice) **cero**

## 2. El tipo *array*

5

### Clasificación de los *arrays*

- Los arrays se clasifican según el número de dimensiones en:
  - **Unidimensionales** (vectores)
  - **Bidimensionales** (matrices)
  - **Multidimensionales**, tres o más dimensiones
- La dimensión de un array es el número de índices utilizados para referenciar a uno cualquiera de sus elementos

Elemento 1
Elemento 2
Elemento 3
.....
Elemento n

**Array**  
unidimensional

Elemento 1,1	.....	Elemento 1,n
Elemento 2,1	.....	Elemento 2,n
Elemento 3,1	.....	Elemento 3,n
.....	.....	.....
Elemento m,1	.....	Elemento m,n

**Array**  
bidimensional

Elemento 1,1,1	.....	Elemento 1,n,1
Elemento 2,1,1	.....	Elemento 2,n,1
Elemento 3,1,1	.....	Elemento 3,n,1
.....	.....	.....
Elemento m,1,1	.....	Elemento m,n,1

**Array**  
multidimensional

# 3. Arrays unidimensionales


6

Un array de una dimensión, también llamado **vector**, es una serie de datos del mismo tipo que se almacenan en posiciones contiguas de memoria, y a los que se puede acceder directamente mediante un único índice.

## Ejemplo:

Supongamos que queremos almacenar la nota del examen de Programación 1 de 50 estudiantes. Por tanto necesitaremos:

1. Reservar 50 posiciones de memoria  $\Rightarrow$  lo más sencillo es declarar un array
2. Dar un nombre al array
3. Asociar una posición en el array a cada uno de los 50 estudiantes
4. Asignar las puntuaciones a cada una de dichas posiciones

Nombre del array		Posiciones del array	Valores almacenados	Direcciones de memoria
 <b>calificaciones</b>	{	calificaciones[0]	7.50	X
		calificaciones[1]	4.75	X + 1
		calificaciones[2]	5.25	X + 2
		...		
		calificaciones[49]	6.00	X + 49

# 3. Arrays unidimensionales

7

## Declaración

- Para poder utilizar una variable de tipo *array* (unidimensional) primero tenemos que declararla.
- Sintaxis:

```
tipo_de_dato nombre_array[num_elem] ;
```

- **tipo\_de\_dato**: indica el **tipo de los elementos** del array. Todos los elementos son del mismo tipo de dato.
  - **nombre\_array**: indica el **nombre del array**. Puede ser cualquier identificador válido.
  - **num\_elem**: indica el **número máximo de elementos** que podrá almacenar el array. Debe ser un valor constante numérico entero.
- Ejemplo: **float calificaciones[50];**

# 3. Arrays unidimensionales

8

## Inicialización y acceso a elementos de un *array*

- Al igual que cualquier otro tipo de variable, antes de utilizar un array debemos inicializar su contenido.
- Una posible forma de inicializar un array es accediendo a cada uno de sus componentes utilizando un bucle y asignarles un valor.
- Para acceder a una posición de un array utilizamos la siguiente sintaxis:

```
nombre_array[índice]
```

**Ejemplo**: Acceso a la calificación del alumno que ocupa la posición 5 en el array: `calificaciones[4];`

**Importante**: Utilizar valores de índices **fuera del rango** comprendido por el tamaño del array puede provocar errores en la ejecución de nuestro programa.



# 3. Arrays unidimensionales

9

## Ejemplo: Inicialización de un array

- Si se conocen los valores a almacenar en las posiciones del array al declararlo, se pueden asignar los valores al mismo tiempo que se declara el array:

```
// ejemplo inicialización array
```

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

```
int main() {
```

```
    int vectorA [4]  = {1, 5, 3, 9};
```

```
    int vectorB []   = {1, 5, 3, 9};
```

```
    int vectorC [10] = {1, 5, 3, 9};
```

```
    return 0;
```

```
}
```

### vectorB

Toma el número de valores como tamaño del vector

### vectorC

Es posible inicializar de manera parcial el array. En este caso, el resto de posiciones del array, de la 4 a la 9, se inicializan a 0.

# 3. Arrays unidimensionales

10

## Ejemplo (II): Inicialización de un array

- Podemos inicializar un array haciendo que el usuario introduzca los datos por teclado, de la siguiente forma:


```
// ejemplo inicialización array
#include <stdio.h>

void inicializar_Array(float cal[]);

int main () {
    float calificaciones[50];
    inicializar_Array(calificaciones);
    return 0;
}

// procedimiento para inicializar el array
void inicializar_Array(float cal[]) {
    int i;

    for ( i=0 ; i < 50 ; i++ ) {
        printf("Introduce la calificación %d: ", i);
        scanf("%f", &(cal[i]) );
    }
}
```



**Nota:** En lenguaje C, el paso de arrays a módulos es siempre **por referencia** y no hace falta indicarlo en la declaración del módulo

# 3. Arrays unidimensionales

11

## Cadenas de caracteres en C

- Una cadena de caracteres, también llamada ***string***, es una secuencia finita de caracteres consecutivos.
- Para almacenar cadenas de caracteres en C se utilizan ***arrays de caracteres***:

```
char nombre_array[];
```

- En un array de caracteres se puede almacenar cualquier texto alfanumérico: palabras, frases, nombres de personas, nombres de ciudades, códigos alfanuméricos, etc.

# 3. Arrays unidimensionales

12

## Cadenas de caracteres en C

- En lenguaje C, una cadena de caracteres se escribe entre comillas dobles:

```
"hola"
```

- En lenguaje C, todas las cadenas de caracteres deben finalizar con el carácter nulo `'\0'`, que debe almacenarse en el array a continuación del último carácter de la cadena:

'h'	'o'	'l'	'a'	'\0'					
0	1	2	3	4	5	6	7	8	9

- La cadena "hola":
  - se ha almacenado en un array de caracteres de tamaño 10
  - está formada por 4 caracteres (tiene longitud 4) pero ocupa en memoria el espacio de 5 caracteres, porque se almacena también el carácter `'\0'`.

```
char cad[10]="hola";
```

# 3. Arrays unidimensionales

13

## Funciones de C para el manejo de *arrays* de char

Función	Descripción	Uso
<code>scanf("%[^\\n]s", &amp;cadena)</code>	Lectura de una cadena de caracteres por teclado hasta fin de línea. La secuencia de caracteres leída se almacena en la variable <i>cadena</i> (array de caracteres)	Como procedimiento
Librería <string.h>		
Función	Descripción	Uso
<code>strcpy(cadena_destino, cadena_origen)</code>	Copia de cadenas. Copia el contenido de <i>cadena_origen</i> en <i>cadena_destino</i>	Como procedimiento
<code>strcat(cadena_1, cadena_2)</code>	Concatenación de cadenas. Concatena el contenido de <i>cadena_2</i> a <i>cadena_1</i>	Como procedimiento
<code>strcmp(cadena1, cadena2)</code>	Comparación alfabética de cadenas <u>si</u> <i>cadena1</i> < <i>cadena2</i> <u>entonces</u> devuelve un número < 0 <u>si</u> <i>cadena1</i> == <i>cadena2</i> <u>entonces</u> devuelve 0 <u>si</u> <i>cadena1</i> > <i>cadena2</i> <u>entonces</u> devuelve un número > 0	Como función
<code>strlen(cadena)</code>	Devuelve un tipo <i>int</i> que indica la longitud de la cadena de caracteres especificada como parámetro, es decir, el número de caracteres válidos de dicho array (hasta el carácter especial de fin de cadena '\\0', sin incluir éste)	Como función

# 3. Arrays unidimensionales

14

## Ejemplos:

- Procedimiento que imprime por pantalla el contenido de un array de elementos de tipo double

```
// imprime por pantalla los elementos de un array de tipo double
void print_Array(double a[], int len){
    int i;
    for (i=0; i < len; i++)
        printf("[%d] = %f\n", i, a[i]);
}
```

- Función que calcula la media de las notas de los alumnos

```
// disponemos de "len" notas de tipo float
float calcular_Media(float a[], int len){
    int i;
    float suma;

    suma = 0.0;
    for (i=0; i < len; i++)
        suma = suma + a[i];

    return(suma / len); // suponemos len > 0
}
```

# 3. Arrays unidimensionales

15

## Ejemplos (II):

Dado un array de enteros, mueve todos sus elementos una posición a la derecha. El desplazamiento será circular, es decir, el último elemento pasará a ser el primero.

```
void mover_En_Circular(int v[]){
    int i, ult;

    // guardar el valor de la última posición del array
    ult = v[LMAX - 1];

    // mover todos los elementos una posición a la derecha,
    // excepto el último
    for(i = LMAX - 1; i > 0; i--){
        v[i] = v[i - 1];

    // guardar en la primera posición el valor que teníamos en la última
    v[0] = ult;
    }
```

# 3. Arrays unidimensionales

16

## Ejemplos (III):

Dado un array de enteros, devolver el mayor valor, el número de ocurrencias de dicho valor, y la posición de la primera y última aparición en la que se encuentra almacenada.

```
void Ocurrencias(int v[], int *mayor, int *num_ocur, int *pos_pri, int *pos_ult){
    int i;

    *mayor = v[0]; // inicialmente el número mayor será el que está en la primera posición
    *num_ocur = 1;
    *pos_pri = 0;
    *pos_ult = 0;

    // recorrer el array: desde la segunda posición hasta la posición final (constante LMAX)
    for (i=1; i < LMAX; i++) {
        if (v[i] > *mayor) { // encontramos un nuevo número mayor
            *mayor = v[i];
            *num_ocur = 1;
            *pos_pri = i;
            *pos_ult = i;
        }
        else if (v[i] == *mayor) { // se encuentra una nueva ocurrencia del número mayor
            *num_ocur = *num_ocur + 1;
            *pos_ult = i;
        }
    }
}
```



# 3. Arrays unidimensionales

17

## Búsqueda de un elemento en un array. Búsqueda lineal

Se recorre el array desde la primera posición accediendo a posiciones consecutivas hasta encontrar el elemento buscado

```
// Búsqueda lineal de un elemento. Función para buscar un elemento "elem" en un array con
// TAM_MAX elementos. Devuelve la posición de "elem" en el array si lo encuentra o -1 si no lo
// encuentra
int Busqueda_Lineal(int nom_array[], int elem)
{
    int pos;
    bool encontrado;

    pos = 0;
    encontrado = false;

    while ( pos < TAM_MAX && !encontrado) { // terminamos la búsqueda si se alcanza el final
        if (nom_array[pos] == elem)        // del array o si se ha encontrado el elemento
            encontrado = true;
        else
            pos = pos + 1;
    }
    if (!encontrado)
        pos = -1;

    return(pos);
}
```

# 3. Arrays unidimensionales

18

## Búsqueda de un elemento en un array. Búsqueda binaria

Si los elementos del array están ORDENADOS podemos utilizar la **búsqueda binaria (dicotómica)**: reducimos la búsqueda dividiendo en mitades, de forma que se va acotando el intervalo de búsqueda dependiendo del valor a buscar.

```
// Búsqueda binaria en un array con TAM_MAX elementos ordenados de forma creciente
int Busqueda_Binaria(int nom_array[], int elem) {
    int pos_inicio, pos_fin, pos_media; // [pos_inicio, pos_fin] = intervalo actual de búsqueda
    bool encontrado = false;

    pos_inicio = 0; // primera posición del array
    pos_fin = TAM_MAX - 1; // última posición del array

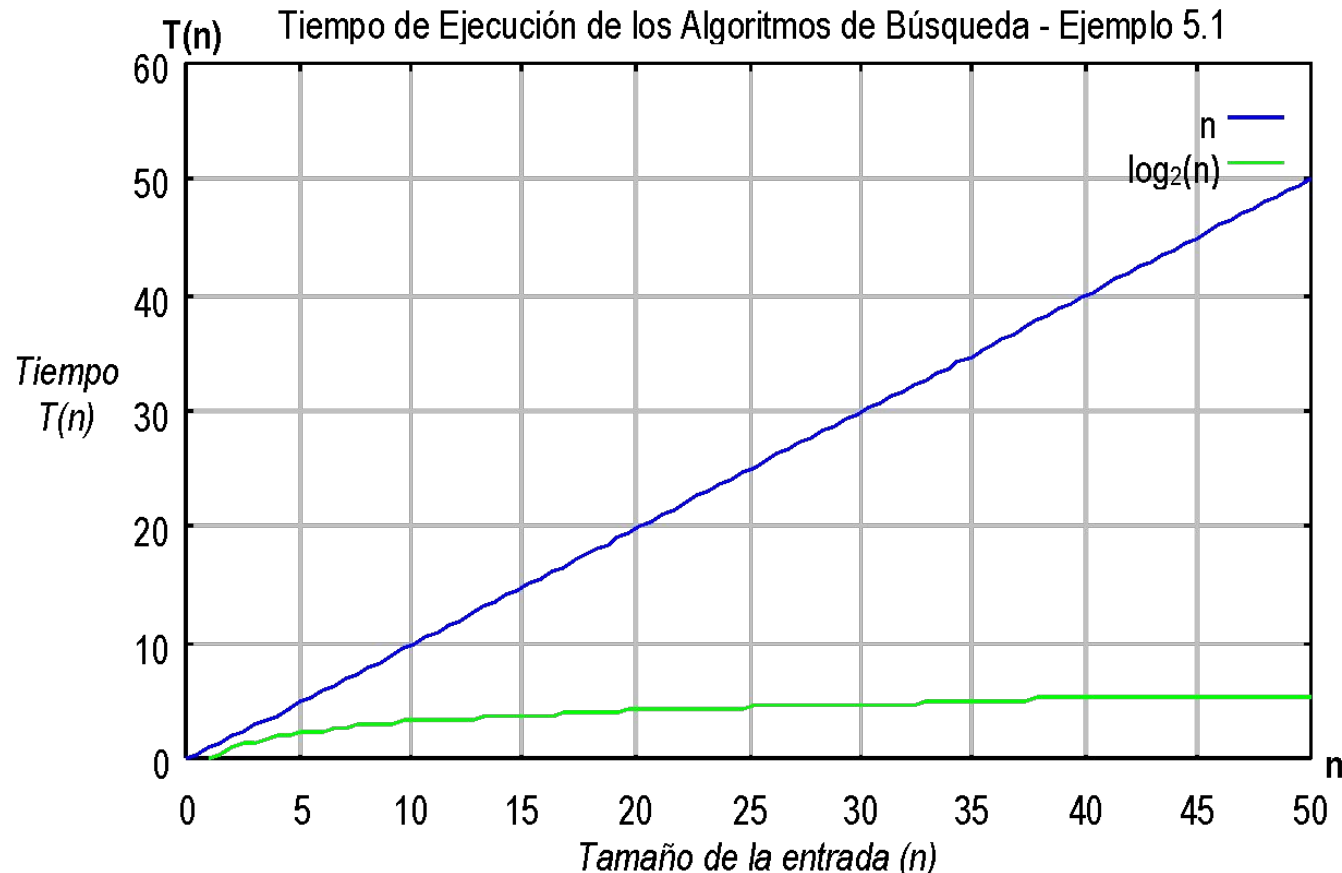
    while ( (pos_inicio <= pos_fin) && !encontrado) {
        pos_media = (pos_inicio + pos_fin) / 2; // posición intermedia del array
        if (elem == nom_array[pos_media]) // elemento encontrado en la posición pos_media
            encontrado = true;
        else if (elem > nom_array[pos_media] )
            pos_inicio = pos_media + 1; // el elemento hay que buscarlo en la mitad superior
        else
            pos_fin = pos_media - 1; // el elemento hay que buscarlo en la mitad inferior
    }
    if (!encontrado)
        pos_media = -1;
    return pos_media;
}
```

# 3. Arrays unidimensionales

19

## Búsqueda de un elemento en un array. Coste temporal.

- Búsqueda **lineal**: tiempo de ejecución lineal
- Búsqueda **binaria**: tiempo de ejecución logarítmico



# 3. Arrays unidimensionales

20

## Algoritmos de ordenación de arrays

- Es interesante y habitual la operación de ordenación en un array.
  - Ejemplo: mantener ordenado nuestro vector de calificaciones para poder consultar rápidamente las cinco mejores notas. Para ello tendríamos que ordenar nuestro vector de mayor a menor (en orden decreciente) y acceder a las cinco primeras posiciones del vector
- Existen muchos algoritmos para ordenar los elementos de un array.

# 3. Arrays unidimensionales

21

## Algoritmos de ordenación de arrays

- Ordenación por **intercambio** (método *burbuja*)

```
// v es el vector de elementos y n el
// número de elementos en el vector
void burbuja(int v[], int n) {
    int aux, i, j;

    for( i = 1; i < n; i++ )
        for( j = n-1; j >= i; j-- ) {
            if(v[j-1] > v[j]) {
                aux = v[j-1];
                v[j-1] = v[j];
                v[j] = aux;
            }
        }
}
```

# 3. Arrays unidimensionales

22

## Algoritmos de ordenación de arrays

### ■ Ordenación por **inserción directa**

Este tipo de ordenación puede compararse con la ordenación de una mano de cartas. Cada vez que cogemos una carta la insertamos en su posición correcta entre las que ya tenemos ordenadas en la mano.

La inserción divide el array en dos partes:

- La parte ordenada. Representa las cartas que tenemos en la mano. Está ordenada y crece en tamaño a medida que avanza la ordenación.
- La parte desordenada. Representa las cartas del mazo que vamos añadiendo. Está sin ordenar, y contiene los elementos que vamos a ir insertando en la parte ordenada. Esta segunda parte va decreciendo a medida que avanza la ordenación.

# 3. Arrays unidimensionales

23

## Algoritmos de ordenación de arrays

### ■ Ordenación por **inserción directa**

```
// v es el vector de elementos y n el
// número de elementos en el vector
void insercionDirecta(int v[], int n) {
    int aux, i, j;

    for( i = 1; i < n; i++ ) {
        aux = v[i];
        j = i - 1;
        while( j >= 0 && v[j] > aux) {
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = aux;
    }
}
```

# 3. Arrays unidimensionales

24

## Algoritmos de ordenación de arrays

### ■ Ordenación por **selección**

- **Paso 1:** buscar y seleccionar de entre todos los elementos que aún no estén ordenados el menor de ellos (si es un orden creciente).
- **Paso 2:** intercambiar las posiciones de ese elemento con el que está en el extremo izquierdo de los desordenados.



# 3. Arrays unidimensionales

25

## Algoritmos de ordenación de arrays

### ■ Ordenación por **selección**

```
// v es el vector de elementos y n el
// número de elementos en el vector
void seleccion(int v[], int n) {
    int aux, i, j, k;

    for( k = 0; k < n - 1; k++ ) {
        i = k;
        j = k + 1;
        while( j < n ) {
            if(v[j] < v[i];
                i = j;
                j = j + 1;
            }
        }
        aux = v[k];
        v[k] = v[i];
        v[i] = aux;
    }
}
```

# 4. Arrays bidimensionales

26

## Arrays bidimensionales

- También reciben el nombre de **matrices**.
- Se necesitan **2 índices** para acceder a cualquiera de sus elementos.

**Ejemplo:** Supongamos que queremos almacenar la nota del examen de 7 grupos de P1, cada uno de los cuales tiene 25 alumnos.

nombre del array bidimensional

notas

cada fila representa un grupo

notas[1][2]: Nota del grupo 1, alumno 2

cada columna representa un alumno en el grupo (fila)

	0	1	2	...	24
0					
1			7.2		
2					
...					
5					
6					

# 4. Arrays bidimensionales

27

## Declaración

- Para poder utilizar una variable de tipo *array* bidimensional, primero tenemos que declararla.
- Sintaxis:

```
tipo nombre_array[n_filas][n_columnas] ;
```

- **tipo**: tipo de dato de cada elemento del array. Todos los elementos del array son del mismo tipo.
- **nombre\_array**: nombre del array.
- **n\_filas**: número de filas del array (primera dimensión).
- **n\_columnas**: número de columnas del array (segunda dimensión).

## 4. Arrays bidimensionales

28

### Inicialización y acceso a un array bidimensional

- Una posible forma de inicializar un array bidimensional es accediendo a cada uno de sus componentes utilizando dos bucles (uno para cada dimensión) y asignarles un valor.
- Para acceder a una posición de un array bidimensional utilizamos la siguiente sintaxis:

```
nombre[indiceF][indiceC]
```

- **nombre**: nombre del array
- **indiceF**: posición de la primera dimensión (**fila**) del array a la que queremos acceder; debe ser un valor comprendido entre 0 y número de filas - 1.
- **indiceC**: posición de la segunda dimensión (**columna**) del array a la que queremos acceder; debe ser un valor comprendido entre 0 y número de columnas - 1.

Ejemplos: (Del ejemplo anterior)

```
notas[6][24]; // acceso a la nota del alumno 25 del grupo 7
notas[6];     // acceso a todas las notas del grupo 7 (array
              // unidimensional asociado a la fila 6)
```

# 4. Arrays bidimensionales

29

## Inicialización y acceso a un array bidimensional

- Si se conocen los valores, se puede inicializar de la siguiente forma:

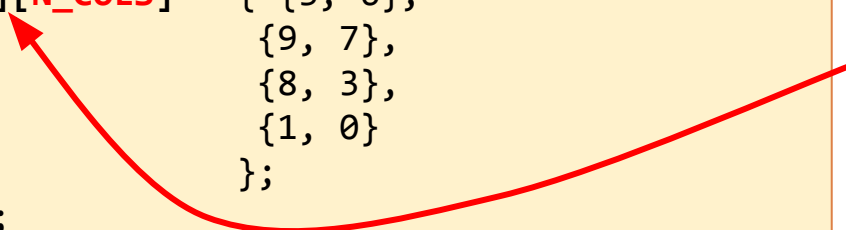
```
// ejemplo inicialización de array bidimensional
#include <stdio.h>

#define N_FILAS 4
#define N_COLS 2

int main() {
    float matDD[N_FILAS][N_COLS] = { {3.6, 6.7},
                                       {2.9, 7.6},
                                       {8.9, 9.3},
                                       {1.9, 0.2}
                                       };

    int mat[][N_COLS] = { {3, 6},
                           {9, 7},
                           {8, 3},
                           {1, 0}
                           };

    return 0;
}
```



En lenguaje C, cuando se declara un array multidimensional no es obligatorio especificar el tamaño de la primera dimensión si se inicializa en la misma declaración. El resto de dimensiones sí que hay que especificarlas.

# 4. Arrays bidimensionales

30

## Inicialización y acceso a un array bidimensional

- También podemos inicializar un array haciendo que el usuario introduzca los datos por teclado, de la siguiente forma:

```
// ejemplo inicialización de array bidimensional
#include<stdio.h>
#define N_FILAS 2
#define N_COLUMNAS 3
void inicializar(float matriz[][N_COLUMNAS]);
void imprimir(float matriz[][N_COLUMNAS]);
int main () {
    float matriz[N_FILAS][N_COLUMNAS];
    inicializar(matriz);
    imprimir(matriz);
    return 0;
}
```

En lenguaje C, no es obligatorio especificar el tamaño de la primera dimensión de un array en la declaración del módulo

```
// procedimiento para inicializar la matriz
void inicializar(float matriz[][N_COLUMNAS]){
    int i, j;
    for ( i = 0 ; i < N_FILAS ; i++ ) {
        printf("Fila %d:\n", i);
        for ( j = 0; j < N_COLUMNAS; j++) {
            printf("\tcolumna %d:", j);
            scanf("%f", &(matriz[i][j]));
        }
    }
}

// procedimiento para imprimir la matriz
void imprimir(float matriz[][N_COLUMNAS]) {
    int i, j;
    for ( i = 0 ; i < N_FILAS ; i++ ) {
        for ( j = 0; j < N_COLUMNAS; j++) {
            printf("%5.2f ", matriz[i][j]);
        }
        printf("\n");
    }
}
```

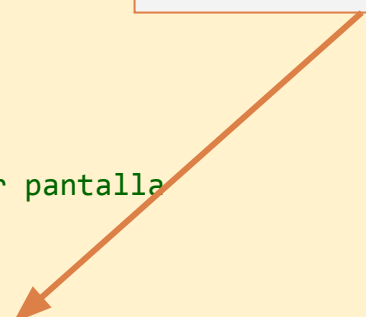
# 4. Arrays bidimensionales

31

## Ejemplo I:

Dados 25 alumnos, de los que se conocen las notas de 7 asignaturas, calcular la nota media de las asignaturas para cada uno de los alumnos e imprimirlas por pantalla.

```
#include<stdio.h>
#define N_ALUMNOS 25
#define N_ASIGNATURAS 7
void imprime_Media_Alumnos(float notas[][N_ASIGNATURAS]);
int main() {
    float notas[N_ALUMNOS][N_ASIGNATURAS];
    ...
    imprime_Media_Alumnos(notas);
    ...
    return 0;
}
// calcula la media de notas para cada alumno y las imprime por pantalla
void imprime_Media_Alumnos(float notas[][N_ASIGNATURAS]) {
    int i;
    for (i = 0; i < N_ALUMNOS; i++){
        printf("El alumno %d tiene de media %4.2f\n", i, calcula_Media(notas[i], N_ASIGNATURAS));
    }
}
```



utilizamos la función de uno de los ejemplos anteriores

# 4. Arrays bidimensionales

32

## Ejemplo II:

Dada una matriz cuadrada de enteros imprimir, en el siguiente orden, los elementos de la diagonal, los elementos del triángulo superior (por encima de la diagonal) y los del triángulo inferior (por debajo de la diagonal), todo ello con un recorrido por filas y columnas.

```
// Versión que recorre tres veces la matriz
void Imprime_Matriz_3(int matriz[][LMAX]){
    int i, j;

    // Imprimir diagonal
    for(i = 0; i < LMAX; i++) // recorrer filas
        for(j = 0; j < LMAX; j++) // recorrer columnas
            if (i == j)
                printf("%d", matriz[i][j]);
    // Imprimir triángulo superior
    for(i = 0; i < LMAX; i++)
        for(j = 0; j < LMAX; j++)
            if (j > i)
                printf("%d", matriz[i][j]);
    // Imprimir triángulo inferior
    for(i = 0; i < LMAX; i++)
        for(j = 0; j < LMAX; j++)
            if (j < i)
                printf("%d", matriz[i][j]);
}
```

```
// Versión que recorre una sola vez la matriz
void Imprime_Matriz_1(int matriz[][LMAX]){
    int i, j;

    // Imprimir diagonal
    for(i = 0; i < LMAX; i++) // recorrer filas
        printf("%d", matriz[i][i]);

    // Imprimir triángulo superior
    for(i = 0; i < LMAX - 1; i++)
        for(j = i + 1; j < LMAX; j++)
            printf("%d", matriz[i][j]);

    // Imprimir triángulo inferior
    for (i = 1; i < LMAX; i++)
        for (j = 0; j < i; j++)
            printf("%d", matriz[i][j]);
}
```