

**Alfredo Vellido : [www.lsi.upc.edu/~avellido](http://www.lsi.upc.edu/~avellido)**

# **Fonaments d'Informàtica**

**Semana 5, p2. Tipos estructurados // tuplas+tablas**

## RECAP: Tipos estructurados, Tuplas

### Un inciso: tuplas. Copia de tuplas y tablas

- Una de las **diferencias** importantes entre tuplas y tablas es la posibilidad de copiarlas haciendo una asignación. Es decir, si tenemos:

```
struct tTupla {  
    int i;  
    char c;  
    string s;  
};  
typedef bool tVector[100];
```

y tenemos también dos variables de cada tipo:

```
tTupla A, B; tVector X, Y;
```

nos interesan las asignaciones:

```
A = B; // SE PUEDE HACER, igual que: A.i=B.i; A.c=B.c; A.s=B.s;
```

```
X = Y; // NO SE PUEDE HACER
```

## RECAP: Tipos estructurados. Tablas+Tuplas

### Un inciso: tuplas

Composición de tipos: tuplas de tablas

```
struct tArticulo {  
    string titulo; // título de un artículo  
    string palabras[5000]; // texto de un artículo  
    int npar; // número de palabras  
};
```

- En este caso, la tupla **tArticulo** tiene un título y una “secuencia” de palabras que es el texto del artículo. Para saber cuántas palabras contiene realmente (5,000 es tan sólo el límite superior), usamos **npar**, que indica la primera casilla de la tabla **palabras** que está vacía.

## RECAP: Tipos estructurados. Tablas+Tuplas

### Un inciso: tuplas

#### Composición de tipos: tuplas de tablas

```
typedef string texto[5000];  
struct tArticulo {  
    string titulo; // título de un artículo  
    texto palabras; // texto de un artículo  
    int npar; // número de palabras  
};
```

- En este caso, la tupla **tArticulo** tiene un título y una “secuencia” de palabras almacenadas en una tabla de tipo **texto**. Para saber cuántas palabras contiene realmente (5,000 es tan sólo el límite superior), usamos **npar**, que indica la primera casilla de la tabla **palabras** que está vacía.

## RECAP: Tipos estructurados. Tablas+Tuplas

### Un inciso: tuplas Composición de tipos: tablas de tuplas

- Las **tablas** pueden tener **tuplas como contenido de las casillas**. Por ejemplo:

```
struct tPunto2D { float x, y; };  
typedef tPunto2D tTablaPuntos[100];
```

- En este ejemplo, hemos creado una tabla, cada casilla de la cual es un punto bidimensional. Para acceder a sus coordenadas, hemos de usar la notación de tablas y tuplas en el orden correcto. Esto es:

```
tTablaPuntos P,Q;  
P[5].x = -3.4;  
Q[10].y = 2.7;
```

- En este caso, ya que **P** es una tabla, hemos de acceder a la casilla 5 con **P[4]**. Pero la casilla **P[4]** es una tupla y, por tanto, para acceder a la coordenada **x** pondremos **P[4].x**.

## Tipos estructurados. Tablas+Tuplas

### Un inciso: tuplas: **Uso de constantes**

- El problema con poner un tamaño determinado a una tabla es que, si en algún momento lo queremos cambiar, hemos de hacerlo en todos los lugares del programa en los que aparece.
- Para evitarlo, se declara una **constante global** (en la misma parte del programa en la que se declaran los tipos), y se usa así:

```
const int TAM_CODI = 12;
typedef char tCodi[TAM_CODI];
void codi_llegeix(tCodi C)
{for (int i = 0; i < TAM_CODI; i++)
    cin >> C[i];
}
```

## Tipos estructurados. Tablas, matrices

### Tablas bidimensionales. Matrices / Declaración

- La **declaración de matrices** es sencilla, se trata de declarar una tabla con doble índice (dos dimensiones) entre corchetes:

```
int imagen[3][5];
```

- Esta declaración es, en concreto, de una **matriz con 3 filas y 5 columnas**. Para C++, lo que importa es el orden de los índices y el que no sobrepasen los límites de rango. Estos límites, como en las tablas simples, **van de 0 a N-1**. La representación de los índices de la matriz sería, p.ej:

	0	1	2	3	4
0	0,0	0,1	0,2	0,3	0,4
1	1,0	1,1	1,2	1,3	1,4
2	2,0	2,1	2,2	2,3	2,4

## Tipos estructurados. Tablas, matrices

### Tablas bidimensionales. Matrices / Declaración de tipo

- Las matrices permiten declaraciones de tipo, tal como las tablas simples, y tienen la misma forma. Para declarar un tipo **tImagen** que contenga 768 filas y 1024 columnas de enteros:

```
typedef int tImagen[768][1024];
```



## Tipos estructurados. Tablas, matrices

### Tablas bidimensionales. Matrices / Acceso a elementos

- El acceso a las casillas de una matriz es similar al acceso en tablas simples, pero usando dos pares de corchetes.
- Por ejemplo, las siguientes expresiones acceden a diferentes casillas de la matriz imagen

`imagen[0][0]` // 1ª fila, 1ª columna

`imagen[767][0]` // última fila, 1ª columna

`imagen[0][1023]` // 1ª fila, última columna

`imagen[767][1023]` // última fila, última columna

ya sea para consultar su valor o para modificarlo. **NOTA:** Un error típico en el acceso a matriz es el de intercambiar los límites de las dimensiones ...

# Tipos estructurados. Tablas, matrices

## Tablas bidimensionales. Matrices

### Acceso a elementos

- **Propuesta de ejercicio 3** Declara una matriz de caracteres con 3 filas y 4 columnas y añádele los caracteres necesarios (sin usar bucles **iterativos**, sino por **asignación directa**) para que contenga lo que muestra el siguiente dibujo:

```
+---+---+---+---+
| X | . | . | X |
+---+---+---+---+
| . | X | X | . |
+---+---+---+---+
| X | X | X | X |
+---+---+---+---+
```

# Tipos estructurados. Tablas, matrices

## Tablas bidimensionales. Matrices / Recorrido de matrices

- Las iteraciones típicas de las tablas unidimensionales suelen hacer uso de un solo bucle, p.ej.:

```
int T[10]; // declaramos tabla de 10 enteros
// Llenamos la tabla con ceros
for (int k = 0; k < 10; k++) { T[k] = 0;}
```

### Propuesta de ejercicio 4

Supongamos la declaración de la variable tabla:

```
int M[10][20];
```

Escribir código para:

- Llenar la primera fila de la matriz con el valor -1.
- Llenar la última columna de la matriz con el valor 5.

## Tipos estructurados. Tablas, matrices

### Tablas bidimensionales. Matrices: Recorrido de matrices

- Las iteraciones típicas de las tablas unidimensionales suelen hacer uso de un solo bucle, p.ej.:

```
int T[10]; // declaramos tabla de 10 enteros
// Llenamos la tabla con ceros
for (int k = 0; k < 10; k++) { T[k] = 0; }
```

- Propuesta de ejercicio 5**

Escribir una **acción** que reciba **3 parámetros**: una **matriz** de 8 x 12 enteros; un entero **k** (que será una fila); y un valor **val** (un entero). La acción ha de llenar la fila **k** de la matriz con el valor **val**.

## Tipos estructurados. Tablas, matrices

### Tablas bidimensionales. Matrices: **Recorrido de matrices**

- ... Sin embargo, si hacemos uso de una sola iteración con las matrices no podremos recorrer todas sus casillas. El siguiente código:

```
int M[10][10];  
for (int k = 0; k < 10; k++) {M[k][k] = 0;}
```

... no llena toda la matriz!!!.

- **Propuesta de ejercicio 6:** ¿Qué parte de la matriz llena, realmente?

## Tipos estructurados. Tablas, matrices

### Tablas bidimensionales. Matrices: **Recorrido de matrices**

- La existencia de filas y columnas hace necesaria la definición de **una iteración dentro de otra** para poder recorrer todas las casillas de una matriz. Por ejemplo, el siguiente código llena la matriz **M** de ceros:

```
for (int i=0; i < 10; i++) // itera las filas
{ for (int j=0; j < 10; j++) // para cada fila, itera columnas
    M[i][j] = 0;
}
```

- Este tipo de “doble bucle” es **MUY** común en matrices.