# Tipos estructurados
## Tablas & Tuplas

**Francisco Mugica**

*CER Intelligent Data Science and Artificial Intelligence*
*Soft Computing Research Group*
*Computer Science Department*
*Universitat Politècnica de Catalunya* (UPC)



# Tipos Estructurados >> Tablas

**Ya sabemos trabajar con variables de tipo simple:**

```
//   Declararlas:
//
     char   sym = '@';
     int    A=0, B=3, C=-99, D;
     float  ancho = 2.23, largo, area;
     bool   ocupado = true;
     string impresora = "EPSON";
//
//   Realizar expresiones y componer
//   instrucciones:
//
   D = (A+C/B);
   cin >> largo; area = ancho*largo;
   cout << "El area es: " << area;

//   Controlar el flujo del programa
//   usando composición secuencial,
//   alternativa o iterativa:

   if (ocupado){D=A+B;}
   else {D=A-B;}
   while (A>C) {C++; D=D+C;}
```

Pero los tipos de datos simples (elementales o básicos) son limitados.

Necesitamos tipos de datos que nos permitan describir entidades más **complejas** y **heterogéneas** para almacenar la información de un mundo complejo.

# Tipos Estructurados >> Tablas

**Supongamos que tenemos 20 impresoras y requerimos almacenar el número de copias que realizan en un día**

```
// Una impresora:
//
    int   Copias_Imp_1;

// 20 impresoras requerirían 20 variables:
//
    int   Copias_Imp_1, Copias_Imp_2;
    int   Copias_Imp_3, Copias_Imp_4;
    int   Copias_Imp_5, Copias_Imp_6;
    int   Copias_Imp_7, Copias_Imp_8;
    int   Copias_Imp_9, Copias_Imp_10;
    int   Copias_Imp_11, Copias_Imp_12;
    int   Copias_Imp_13, Copias_Imp_14;
    int   Copias_Imp_15, Copias_Imp_16;
    int   Copias_Imp_17, Copias_Imp_18;
    int   Copias_Imp_19, Copias_Imp_20;
```

Esto además de incómodo sigue siendo limitado. ¿Y si fueran mil?

Afortunadamente contamos con **tipos de datos estructurados** que nos permiten tratar esta complejidad

# Tipos Estructurados >> Tablas

**Supongamos que tenemos 20 impresoras y requerimos almacenar el número de copias que realizan en un día**

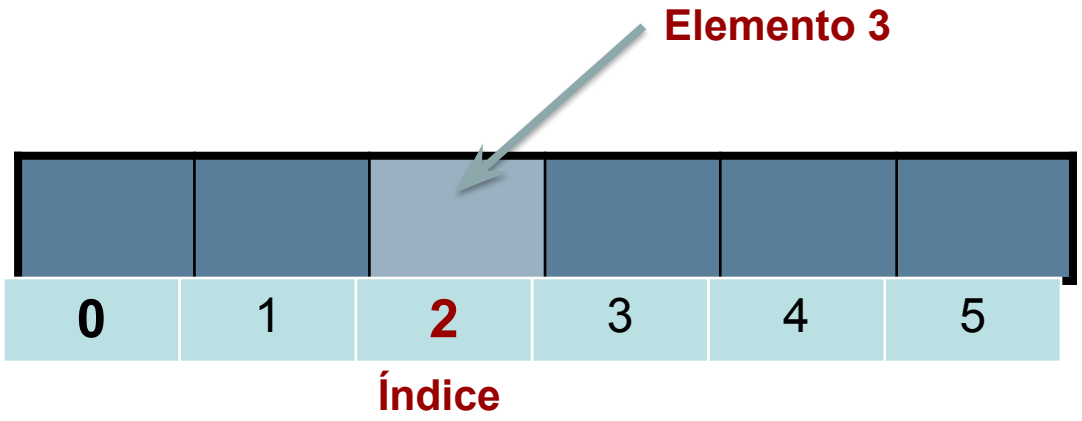
```
// Una impresora:
//
    int   Copias_Imp_1;

// 20 impresoras requerirían 20 variables:
//
    int   Copias_Imp[20];
```

¡Simplemente así!

A este tipo de datos estructurados se les conoce como: **tablas**

# Tipos Estructurados >> Tablas

- Una tabla es una **agrupación de variables del mismo tipo**:




- A cada variable individual de la tabla se le conoce como casilla, celda o elemento. Esta tabla tiene **6 elementos.**
- Cada elemento de la tabla es accesible a partir de su índice (un entero)
- **ATENCIÓN:** Los índices siempre empiezan en **cero**
- Así, el elemento **3** se accede utilizando el índice **2**

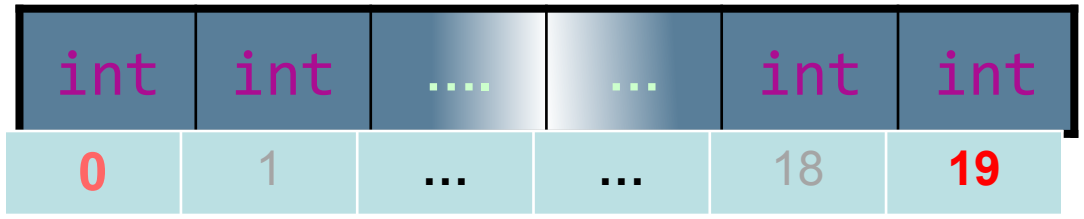
# Tipos Estructurados >> Tablas >> Declaración

- La nomenclatura en C++ para definir una tabla es:

```
<Tipo><NombreTabla> [<NumeroElementos>];
```

- Volviendo al ejemplo de las 20 impresoras, declararíamos la tabla con elementos del tipo `int` y le llamaríamos `Copias_Imp` asignándole un tamaño de `20` elementos:

```
int Copias_Imp[20];
```



# Tipos Estructurados >> Tablas >> Acceso

Para acceder a cada elemento de la tabla simplemente se usa el nombre de la tabla seguido del número de índice que le corresponde entre corchetes:

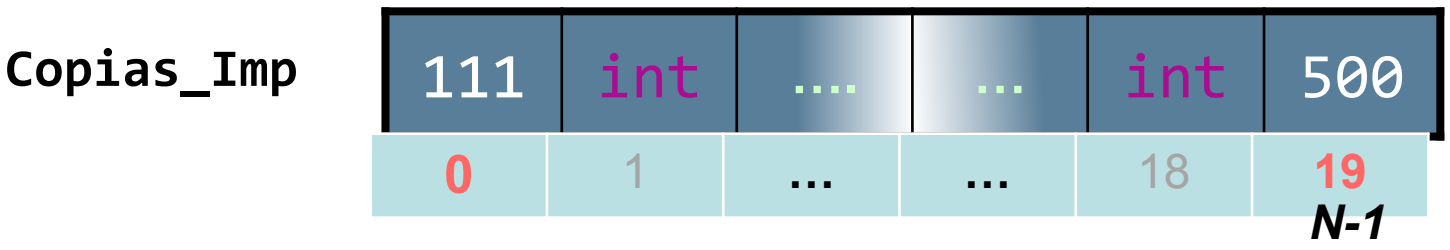
```
Nombre_tabla[<indice>]
```

Por ejemplo, en la tabla de impresoras, si se desea almacenar el dato de 111 copias a la impresora uno y 500 a la última impresora (la 20):

```
const int N = 20;
int Copias_Imp[N];
Copias_Imp[0] = 111;
Copias_Imp[19] = 500;
```

**Notar que el primer índice siempre será 0 y el último siempre N-1.** Por lo que también se puede escribir así utilizando la constante N:

```
int Copias_Imp[N-1] = 500;
```



# Tipos Estructurados >> Tablas >> Acceso

Otra forma de llenar las celdas de una tabla es mediante un **índice calculado**, es decir, utilizando una variable entera con un valor definido.

Por ejemplo, el siguiente código deposita en las celdas 1, 5 Y último de la tabla `Copias_Imp` el valor -99 usando las variables k y m;

```
const int N = 20;
int Copias_Imp[N];
int k=1, m=5;
Copias_Imp[k-1] = -99;
Copias_Imp[m-1] = -99;
Copias_Imp[N-1] = -99;
```

# Tipos Estructurados >> Tablas >> Limitaciones

- Tamaño Fijo – No se puede modificar
- Útil solo si hay un número fijo de posiciones.
- Qué hacer si no se sabe el tamaño exacto ?
- Sobredimensionar y lidiar con una tabla parcialmente llena.


```
int Tabla[6], n_elem;
```

# Tipos Estructurados >> Clase Vector >> Definición

Afortunadamente contamos con la clase vector de C++

- Tablas de dimensión variable
- Biblioteca: `#include <vector>`
- La nomenclatura en C++ para definir un vector es:

  **vector <**tipo**>** <nombre> (<elementos>, <val_ini>);

- Ejemplos:

```
vector <int> Tabla;          //tabla con 0 elementos
vector <int> Tabla(6);       //tabla con 6 elementos
vector <int> Tabla(6,0); //tabla 6 elementos
                             //inicializados a cero
```

# Tipos Estructurados >> Clase Vector >> Métodos

```
vector<int> Tabla(6,0);
```

- **size();**

```
n = Tabla.size();          // elementos actuales
```

- **push_back(), pop_back():**

```
int z=-99;
Tabla.push_back(z);        //se guarda z al final
Tabla.pop_back();          //quita el último elemento
```

0 0 0 0 0 0 -99

0 1 2 3 4 5  6

- **Subprogramas: paso por referencia**

```
int SumaTabla(vector<int> & Tabla);
```

# Tipos Estructurados >> Clase Vector >> Métodos

```
vector<int> Tabla(6,0);
```

- **clear();**

```
Tabla.clear()        //  quita todos los
                     // elementos del vector
```

- **Subprogramas: paso por referencia**

```
int SumaTabla(vector<int> & Tabla);
```

# ¡los *strings* son tablas!

- Las variables de tipo ***string*** son, de hecho, tablas de caracteres, y de longitud variable. Cuando se declara un **string** y no se inicializa, e.g.

  **string s;**

  la tabla está de hecho vacía (no "tiene casillas"). Si hacemos una asignación como …

  **s = "hola mundo";**

  la tabla que representa **s** pasa a hacerse de 10 casillas, y se llena con los 10 caracteres que hemos puesto entre comillas dobles.

## ¡los *strings* son tablas!

- Se puede acceder a las casillas de un **string** como a las de cualquier tabla. Por ejemplo, el siguiente código

```
string g = “radiohear";
g[8] = ‘d';
cout << g << endl;
```

… mostraría por pantalla: **radiohead**

Hemos, por tanto, cambiado la novena casilla de la tabla (índice 8), de ser una ‘r' a ser una ‘d'

.