

# Arquitectura de Prog. Para Hardware AD22

[Tablero](#)[Mis cursos](#)[EspMecAPH](#)[Unidad 1: Firmware orientado a Objetos](#)[Libro de FOO](#)

Libro de FOO




## 6. Arreglos

### [§1 Sinopsis](#)

Desde el punto de vista del programa, una matriz (array ó vector) es una zona de almacenamiento contiguo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz [1]. Desde el punto de vista lógico podemos considerarlas como un conjunto de elementos ordenados en fila. Así pues, en principio todas las matrices

son de una dimensión, la dimensión principal, pero veremos que los elementos de esta fila pueden ser a su vez matrices (un proceso que puede ser recursivo), lo que nos permite hablar de la existencia de matrices multi-dimensionales, aunque las más fáciles de "ver" o imaginar son las de dos y tres dimensiones.


**Nota:** aunque en C/C++ los conjuntos ordenados de elementos del mismo tipo se denomina **matrices** (arreglos), la idea aparece en otros lenguajes bajo distintos nombres. Por ejemplo, *vector*, *lista* ("list") o *secuencia* ("sequence"). En cualquier caso, **no** confundirlas (las matrices) con los conjuntos de pares nombre/valor, que existen en otros lenguajes bajo los nombres de *diccionarios* ("dictionaries"); *tablas hash* ("hash tables"); *listas indexadas* ("keyed lists") o *matrices asociativas* ("associative arrays"), pero que como tales, no existen en C++; aunque la Librería Estándar sí dispone de tales estructuras ( [5.1.1a](#)).

Como advertencia para los lectores que han utilizado matrices en otros lenguajes, **señalar que quizás el aspecto más significativo del manejo de matrices en C++, es que el compilador desconoce su tamaño, de forma que el programador debe adoptar precauciones para no salir de sus límites, ya que el compilador permite referenciar elementos inexistentes, más allá del final de la matriz, con el consiguiente riesgo de error.**

## [§1 Sintaxis](#)

La declaración de matrices sigue la siguiente sintaxis:

**tipoX etiqueta [**<expr-const>**]**

- **tipoX** es el tipo de los elementos que serán almacenados en la matriz. Puede ser cualquier type-id ( [2.2](#)) válido a excepción de **void** y de funciones (no pueden existir matrices de funciones, pero sí de punteros-a-función).
- **etiqueta** es el identificador
- **<expr-const>**: una expresión cuyo resultado debe ser una constante entera positiva **n** distinta de cero, que es el número de elementos de la matriz. Los elementos están numerados desde **0** hasta **n-1**.

Ejemplos:

```
int a[10];    // declara una matriz de 10 elementos enteros
char ch[10]   // ídem de 10 elementos char
char* p[10]   // ídem de 10 elementos puntero-a-carácter
struct St mst[10] // ídem de 10 elementos estructuras tipo St
```

§4 En ciertos contextos, el primer declarador de una matriz puede no contener una <expresión> dentro de las llaves, por ejemplo:

```
int array[];
```

Este tipo de matriz, de tamaño indeterminado, resulta aceptable en situaciones donde el tamaño no es necesario para reservar espacio, pero siguen siendo necesarias las llaves para indicar que la variable es una matriz. Por ejemplo, una declaración **extern** de una matriz no necesita el tamaño exacto de la misma; alguna otra declaración tampoco la necesita, porque el tamaño está implícito en el declarador. Por ejemplo:

```
char arr[] = "AEIOU";           // declara matriz de 6 elementos [1]
char arr[] = {'A','E','I','O','U'}; // declara matriz de 5 elementos
```

[1] Respecto al porqué decimos que declara 6 elementos, y no 5 como parecería lógico, ver "Constantes literales" ([3.2.3f](#)).

### SUPER NOTA:

La exigencia de que el resultado de <expr-const> sea un valor constante, es de la mayor trascendencia para entender las limitaciones de las matrices C++. Significa que el tamaño de la matriz debe ser conocida en tiempo de compilación.

Por ejemplo, **no es posible algo como:**

```
unsigned int size;
```

```
...
```

```
char matriz[size]; // Error!!
```

en su lugar debe hacerse:

```
const unsigned int size = 10;
```

```
...
```

```
char matriz[size]; // Ok.
```

pero entonces es preferible establecer directamente:

```
char matriz[10]; // Ok.
```

o mejor aún:

```
#define ASIZE 10
```

```
...
```

```
char matriz[ASIZE]; // Ok.
```

En general, cuando se necesitan matrices que cambien de tamaño en "runtime", suele recurrirse a crearlas en el montón mediante el operador `new[]` ([4.9.20c](#)) que sí permite definir su tamaño en función de una variable. Por ejemplo:

```
unsigned int size;
```

```
...
```

```
char* mptr = new char[size]; // Ok.
```

*En este caso las matrices no son referenciadas directamente, sino a través de un puntero, y una vez creadas tampoco es posible cambiar su tamaño. El recurso utilizado cuando se necesita cambiar este, es crear otra matriz del tamaño adecuado; copiar en su caso los miembros de la antigua a la nueva; borrar la antigua (liberar el espacio asignado), y finalmente, asignar el puntero a la nueva matriz. De esta forma la ilusión del usuario es que realmente se ha cambiado el tamaño de la matriz, aunque la realidad subyacente sea muy diferente.*

Otro recurso, utilizado cuando la matriz "puede" crecer pero no se está muy seguro, es crearla con un tamaño inicial algo mayor que lo necesario (por ejemplo un 25%). En estos casos se dispone de cierto espacio de reserva antes que sea necesario proceder a un proceso de copia total.

No obstante lo anterior, en **C-99** sí es lícita la declaración de matrices de longitud variable ("variable length arrays") que es como se conoce a las que pueden definir su tamaño en runtime. Es decir, en este entorno están permitidas declaraciones del tipo

```
unsigned int size;
```

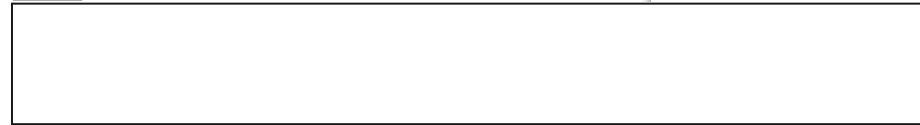
```
char matriz[size]; // Ok
```

Con objeto de garantizar la compatibilidad con el código C existente, algunos compiladores permiten la definición de estas matrices cuando son de naturaleza automática (definidas en el interior de una función). Es el caso del compilador GNU gcc <http://gcc.gnu.org/>. Sin embargo, es un recurso que debe evitarse, ya que el código resultante no resulta portable.



TECNOLÓGICO  
NACIONAL DE MÉXICO

Ads





Actividad anterior

◀ Participación Reunión Anual

Ir a...

Actividad siguiente

FORO UNIDAD 1 ▶

## Tabla de Contenidos

### [1. Sistemas Embebidos](#)

#### [1.1. Componentes de Sistemas Embebidos](#)

#### [1.2. MSP432P401R](#)

### [2. Unidad 1 "POO"](#)

### [3. Lenguaje C/C++](#)

#### [3.1. Ejercicio\\_1 Directiva #include](#)

#### [3.2. Tipo bool](#)

#### [3.3. Lectura\\_Ámbito y Visibilidad](#)

#### [3.4. Lectura\\_Espacio de Nombres](#)

#### [3.5. Ejemplo Espacios\\_de\\_Nombres](#)

### [4. Memoria](#)

#### [4.1. Ejemplo1 de Variables y Stack de Memoria](#)

#### [4.2. Especificadores](#)

#### [4.3. Ejercicio1\\_Variables](#)

### [5. Punteros](#)

#### [5.1. Ejemplo1 Punteros](#)

#### [5.2. Ejemplo2 Punteros para Ejercicio1](#)

[5.3. Ejemplo3 Punteros para Ejercicio2](#)

[5.4. Ejercicio1 Punteros](#)

[5.5. Ejercicio2 Punteros](#)

## 6. Arreglos

[6.1. Ejemplos Arreglos y Strings](#)

[7. Control de Versiones con Git](#)

---

## Mantenerse en contacto

 <https://itchihuahua.mx/>

 [contacto@itchenlinea.mx](mailto:contacto@itchenlinea.mx)



 Obtener la App Mobile