

## CONTENIDOS

### **1. Introducción al tipo de dato ARRAY.**

Definición, Características, Declaración,  
Acceso e Inicialización.

### **2. Arrays multidimensionales**

Definición, Declaración, Acceso e Inicialización.

# 1

## **Introducción al tipo de dato ARRAY**

## EL TIPO COMPUESTO BASICO: Los Arrays

### Introducción

Hasta ahora hemos visto los tipos de datos simples de C++, y sus características .

- **int** ( Números enteros )
- **float, double** ( Números reales )
- **bool** ( Valores lógicos )
- **char** ( Caracteres y cualquier cantidad de 8 bits )
- **void** ( Nada. Sirve para indicar que una función no devuelve valores )

También se ha visto cómo declarar y utilizar constantes utilizando **const** y el tipo enumerado **enum** .

Ahora vamos a introducir un tipo de dato definido por el programador:  
el tipo **array**

## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Definición

- ★ Un *Array* es un conjunto finito de valores del mismo tipo.
- ★ Un *Array* es una estructura de datos que almacena bajo el mismo nombre (variable) a una colección de datos del mismo tipo.
- ★ Un *Array* es un conjunto de variables del mismo tipo que tienen el mismo nombre y se diferencian en el índice.

Letras = 

'a'	'z'	'r'	'j'
-----	-----	-----	-----

- A cada dato almacenado se le denomina *elemento del array* o *ítem*.
- Cada elemento del array está numerado, y a éste número se le denomina *índice*.

## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Características

- ◆ Los elementos del array se numeran consecutivamente comenzando con 0, 1, 2, ...

letras = 

'a'	'z'	'r'	'j'
-----	-----	-----	-----

  
          0      1      2      3

índice

- ◆ Estos números, los índices, permiten localizar cada elemento del array.

letras[0] es el elemento que está en la posición 0,

letras[1] es el elemento que está en la posición 1, etc ...

array de 4  
elementos

- ◆ Los elementos almacenados en el array pueden ser de cualquier tipo: tipos simples como: **int**, **char**, **bool**, **float** ó tipos definidos por el programador como por ejemplo *estructuras* (hablaremos de ellas más tarde).

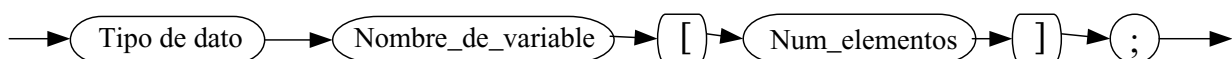
## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Declaración de un array

- Se declara de forma similar a cualquier otro tipo de datos, solo que hay que indicarle al compilador el número de elementos que forma el array: *tamaño ó longitud del array*.

<tipo\_de\_dato> <nombre\_de\_variable> [numero\_de\_elementos];

Sintaxis



Ejemplos:

```
char letras[4]; // variable letras de tipo array de 4 elementos de tipo char  
int edades[10]; // variable edades de tipo array de 10 elementos de tipo entero.
```

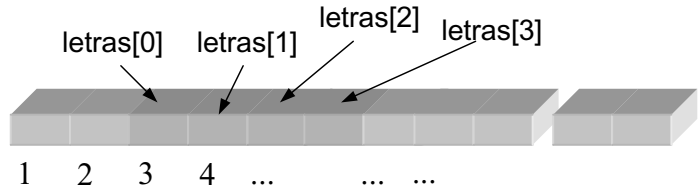
## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Declaración de un array

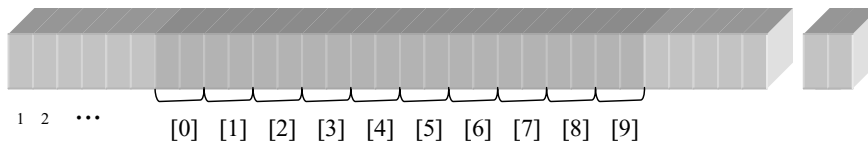
- El compilador reserva espacio en memoria en posiciones contiguas.

```
char letras[4];  
int edades[10];
```

El array `letras` ocupa 4 bytes de memoria.



Cuando declaramos el array `edades` lo que hacemos es reservar un espacio de memoria para 10 variables de tipo `int`.



El array `edades` ocupa 20 bytes consecutivos de memoria (2 bytes por elemento).

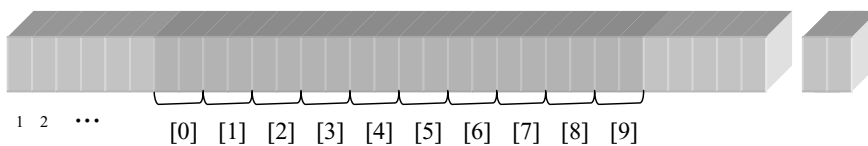
## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Declaración de un array

- Si queremos saber el número de bytes que necesitamos para almacenar un array en memoria, podemos utilizar la función `sizeof()`.

En nuestro caso, `sizeof (edades) = 20`.

```
int edades[10];
```



↑  
Cabecera `stdio.h`

## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Acceso a los elementos de un array

Para acceder a los elementos de un array hay que poner el nombre y el índice del elemento al que queremos acceder ó referenciar.

```
...  
int primera = 1, k = 3;  
char letras[4];
```

```
...  
cout << letras[0]; //visualiza la letra 'a'  
cout << letras[2]; //visualiza la letra 'r'
```

```
...  
cin >> letras[0];  
letras[2] = 'R';
```

```
...  
letras[0] = letras[primera+2];  
cout << letras[k-2]; //visualiza la letra 'z'
```

```
..
```

```
letras = 

|     |     |     |     |
|-----|-----|-----|-----|
| 'a' | 'z' | 'r' | 'j' |
| 0   | 1   | 2   | 3   |


```

Podemos visualizar el valor de los elementos de un array

Modificar el valor de los elementos de un array

Acceder a los elementos utilizando fórmulas

## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Acceso a los elementos de un array

Si necesitamos acceder a todos los elementos, utilizaremos un bucle de tipo **for**:

```
for ( int j = 0; j < num_elementos ; j++ )
```

```
    procesar_elemento [j];
```

Sintaxis

Variable de control **j** para recorrer cada uno de los elementos

```
...  
char letras[4];
```

```
...  
for ( int j = 0 ; j < 4 ; j++ )
```

```
{  
    cout << "Introduzca el elemento : " << j+1 ;  
    cin >> letras[j];  
    cout << endl;  
}
```

```
...
```

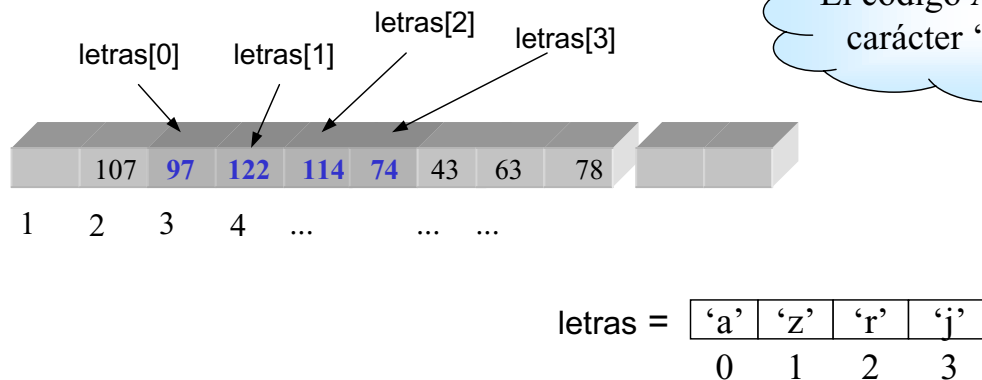
Pasada	j	Salida por pantalla
1	0	Introduzca el elemento: 1
2	1	Introduzca el elemento: 2
3	2	Introduzca el elemento: 3
4	3	Introduzca el elemento: 4

## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Acceso a los elementos de un array

C++ no comprueba que los índices del array estén dentro del rango permitido.

Si escribimos `cout << letras[4];` no tendremos error de compilación, pero el programa no funcionará correctamente.



11

## EL TIPO COMPUESTO BASICO: Los Arrays

### El tipo Array: Inicialización de un array

Antes de empezar a utilizar una variable de tipo array hay que asignar valores a cada uno de sus elementos. Tenemos varias formas de inicializar un array:

#### 1. Inicialización en la declaración:

```
...  
char letras[4] = { 'a', 'z', 't', 'j' };  
...
```

Los valores se encierran entre llaves y se separan por comas.

```
...  
int edades[] = { 10, 20, 30, 40, 50, 60 };  
...
```

C++ permite omitir el tamaño del array cuando se inicializa. El compilador reserva memoria para un array de enteros de tamaño 6.

```
char saludo[] = { 'h', 'o', 'l', 'a' };  
...
```

```
char saludo[] = { "hola" };  
...
```

Los array de caracteres se pueden inicializar de éstas dos formas.

12

## El tipo Array: Inicialización de un array

### 2. Inicialización elemento a elemento en el cuerpo del programa:

```
letras[0] = 'a';  
letras[1] = 'z';  
letras[2] = 'r';  
letras[3] = 'j';
```

Este método no es muy práctico cuando tenemos muchos elementos.

### 3. Inicialización mediante una sentencia FOR:

```
int edades[ 6 ];  
int valor = 10;  
for (int i = 0; i<6 ; i++ )  
{  
    edades[i] = 10;  
    valor = valor + 10 ;  
}
```

```
int temperatura[ 10 ];  
for (int i = 0; i<10; i++)  
    edades[i] = 0;
```

Temperatura = 

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

edades = 

10	20	30	40	50	60
----	----	----	----	----	----

## 2

## ARRAYS multidimensionales

### Arrays multidimensionales: Definición

Los arrays que hemos visto anteriormente se conocen como **arrays unidimensionales** y se caracterizan por tener un solo índice. También se conocen como **listas de elementos** y se corresponden con el concepto de **vector**.

Los **arrays multidimensionales** son aquellos que tienen más de una dimensión y por tanto tienen más de un índice. Los más utilizados son los de dos dimensiones, conocidos con el nombre de **tablas**. Se corresponden con el concepto de **matriz**.

C++ permite trabajar con arrays de de tantas dimensiones como requieran las aplicaciones ( 3, 4 ó más dimensiones).

### Arrays bidimensionales: Definición

- Un array de dos dimensiones se corresponde con una tabla con varias filas y varias columnas.

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

índice para las filas

índice para las columnas

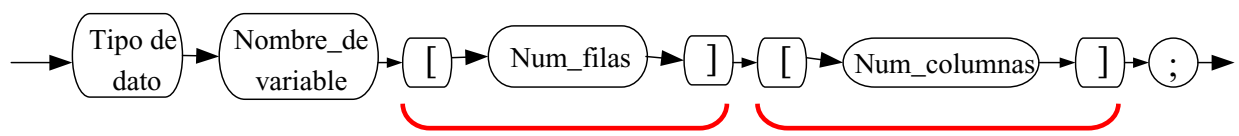
- Cada elemento almacenado en el array está identificado por dos índices, sus coordenadas, la fila y la columna en la que se encuentra dicho elemento.
- Ambos índices se numeran consecutivamente comenzando con 0, 1, 2, ...

## Arrays bidimensionales: Declaración

- Se declara de forma similar al tipo de dato array de una dimensión, solo que hay que indicarle al compilador el tamaño de cada uno de los índices, es decir, el número de filas y el número de columnas.

Sintaxis

```
<tipo_de_dato> <nombre_de_variable> [num_filas] [num_columnas];
```

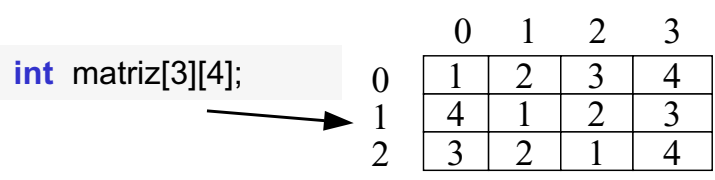


Ejemplos:

```
char tablero[8][8]; // variable llamada tablero de tipo array de dos dimensiones (8 filas y 8 columnas). Almacena 64 elementos de tipo char.
int matriz[3][4]; // variable llamada matriz de tipo array de dos dimensiones (3 filas y 4 columnas). Almacena 12 elementos de tipo entero.
```

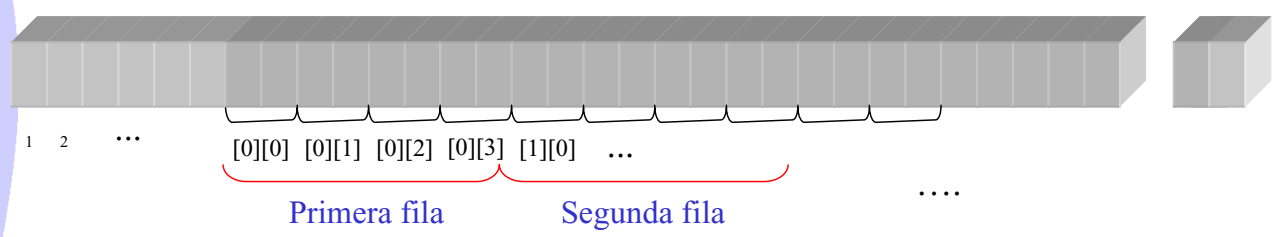
## Arrays bidimensionales: Declaración

- Al igual que en los arrays de una dimensión, el compilador reserva espacio en memoria en posiciones contiguas. Primero se almacenan los datos de la primera fila, luego los de la segunda, ...



El array **matriz** ocupa 24 bytes consecutivos de memoria.

Cuando declaramos el array **matriz** lo que hacemos es reservar un espacio de memoria para 12 variables de tipo **int**.



## EL TIPO COMPUESTO BASICO: Los Arrays

### Arrays bidimensionales: Acceso a los elementos

Se puede acceder a los elementos de un array bidimensional de forma similar a como lo hacíamos para arrays de una dimensión. Hay que poner el nombre y los índices (fila y columna) del elemento al que queremos acceder ó referenciar.

```
int j = 0;
char valor ;

int matriz[3][4];
char cuadro[3][3];

cout <<matriz[0][3];
...
cin >> matriz[1][ j+1 ];
cuadro[2][2] = 'R';
...
valor = cuadro[2][1];
...
```

Podemos visualizar el valor de los elementos de un array

Modificar el valor de los elementos de un array

Acceder a los elementos para extraer valores

19

## EL TIPO COMPUESTO BASICO: Los Arrays

### Arrays bidimensionales: Acceso a los elementos

Si necesitamos acceder a todos los elementos de un array bidimensional, utilizaremos un doble bucle for.

```
for ( int i = 0; i < num_filas ; i++ )
    for ( int j = 0 ; j < num_columnas ; j ++ )
        procesar_elemento [i][j];
```

Variable de control **i** para recorrer las filas

Variable de control **j** para recorrer las columnas

Sintaxis

```
int matriz[3][4];
...
for ( int i = 0 ; i < 3 ; i++ )
    for ( int j = 0 ; j < 4 ; j++ )
    {
        cout << "El elemento " << i+1 << j+1 << "es: ";
        cout << matriz [i] [j] << endl;
    }
...
```

20

## Arrays bidimensionales: Inicialización

Al igual que los arrays de una dimensión, tenemos varias formas de inicializar un array:

### 1. Inicialización en la declaración:

Los valores se encierran entre llaves y se separan por comas.

```

...
int matriz[3][4] = { 1, 2, 3, 4, 4,1,2, 3, 3, 2,1,4 };
int valores[2][3] = { {10, 20, 30}, {0,1,2} };
char cuadro[3][3] = {
    { 'B', 'N', 'N' },
    { 'N', 'N', 'N' },
    { 'B', 'B', 'B' }
};
...
    
```

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

	0	1	2
0	10	20	30
1	0	1	2

	0	1	2
0	B	N	N
1	N	N	N
2	B	B	B

## Arrays bidimensionales: Inicialización

### 2. Inicialización elemento a elemento en el cuerpo del programa:

```

matriz[0][0] = 1;
matriz[0][1] = 2;
matriz[0][2] = 3;
...
matriz[2][3] = 4;
    
```

Poco práctico como ocurría en arrays de una dimensión.

### 3. Inicialización mediante una doble sentencia FOR:

```

int valores[2][3]
int dato = 10;
for (int i = 0; i<2 ; i++)
    for (int j = 0; j<3 ; j++)
    {
        valores[i][j] = dato;
        dato = dato + 10 ;
    }
    
```

Variable de control **i** para recorrer las filas

Variable de control **j** para recorrer las columnas

	0	1	2
0	10	20	30
1	40	50	60

## EL TIPO COMPUESTO BASICO: Los Arrays

### Más sobre inicialización de Arrays:

- ➔ Si un array se declara globalmente, y no se inicializa en dicha declaración, el compilador se encarga de inicializarlo automáticamente con un valor por defecto.
- ➔ Si no declaran globalmente, pero se inicializan uno o más elementos pero no todos, el compilador inicializa automáticamente el resto de elementos con un valor por defecto.

Si el array almacena elementos de tipo entero o real:

El valor por defecto es 0.

Si el array almacena elementos de tipo carácter:

El valor por defecto es el carácter nulo '\0'.

## EL TIPO COMPUESTO BASICO: Los Arrays

### Más sobre inicialización de Arrays:

#### Ejemplo:

```
#include <iostream.h>
```

```
int matriz[3][4];  
char cuadro[3][3];
```

```
void main()  
{  
    float ventas[4] = {3.2};  
    int otrocuadro[3][3] = { 7, 6 };  
    ...  
}
```

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

matriz

	0	1	2
0	'\0'	'\0'	'\0'
1	'\0'	'\0'	'\0'
2	'\0'	'\0'	'\0'

cuadro

	0	1	2	3
ventas	3.2	0	0	0

	0	1	2
0	7	6	0
1	0	0	0
2	0	0	0

otrocuadro