



# Introducción a c++ y la programación orientada a objetos

Temperaturas	Códigos	Voltajes
95.75	Z	98
83.0	C	87
97.625	K	92
72.5	L	79
86.25		85
		72

## Arreglos y Vectores

El nombre del arreglo es c

Número de posición del elemento dentro del arreglo c	c[ 0 ]	-45	
	c[ 1 ]	6	
	c[ 2 ]	0	
	c[ 3 ]	72	
Nombre de un elemento individual del arreglo	c[ 4 ]	1543	Valor
	c[ 5 ]	-89	
	c[ 6 ]	0	
	c[ 7 ]	62	
	c[ 8 ]	-3	
	c[ 9 ]	1	
	c[ 10 ]	6453	
	c[ 11 ]	78	

```
int main()
{
    int C[10];
    for (int j =0; j<10;j++){
        C[j]=0;
    }
    cout << "elemento " << " valor" << endl;

    for (int j =0; j<10;j++){
        cout << j << C[j] << endl;
    }
    return 0;
}
```

int A[5]: // A es un arreglo de 5 enteros

**A[j]** j es el número de la posición del elemento dentro del arreglo

# Inicialización de un arreglo en una declaración

```
int temp[5] = {98, 87, 92, 79, 85};  
char codigos[6] = {'m', 'u', 'e', 's', 't', 'r', 'a'};  
double pendientes[7] = {11.96, 6.43, 2.58, .86, 5.89, 7.56, 8.22};
```

```
int galones[20] = {19, 16, 14, 19, 20, 18,  
                  12, 10, 22, 15, 18, 17,  
                  16, 14, 23, 19, 15, 18,  
                  21, 5};
```

```
int A[]={1,2,3,4,5};
```

```
int B[5]={1,2,3,4,5};
```

```
int C[7]={1,2,3,4,5,6}; //???
```

```
int main()  
{  
    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };  
  
    cout << "Elemento" << setw( 13 ) << "Valor" << endl;  
  
    for ( int i = 0; i < 10; i++ ){  
        cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;  
    }  
    return 0;  
}
```

# Tamaño arreglo con una variable constante y establecimiento de los elementos de un arreglo con cálculos

```
#include<iostream>

using namespace std;

int main()
{
    // la variable constante se puede usar para especificar el tamaño de los arreglos
    const int tamañoArreglo = 10; // debe inicializarse en la declaración

    int s[ tamañoArreglo ]; // el arreglo s tiene 10 elementos

    for ( int i = 0; i < tamañoArreglo; i++){
        s[ i ] = 2 + 2 * i; // establece los valores
    }

    cout << "Elemento" << setw( 13 ) << "Valor" << endl;

    for ( int j = 0; j < tamañoArreglo; j++ )
        cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;

    return 0;
}
```

Esto compila?

E. Código  
example1\_barras

E. Código  
example2\_contadores (datos)

E. Código  
example3\_encuesta

*"C++ no cuenta con comprobación de límites para evitar que la computadora haga referencia a un elemento que no existe."*

# Uso de arreglos tipo carácter para almacenar y manipular cadenas

```
char cadena1[] = "first";  
char cadena1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

## caracter nulo '\0'

Todas las cadenas representadas mediante arreglos de caracteres terminan con este carácter.

Sin él, este arreglo representaría tan sólo un arreglo de caracteres, no una cadena

```
char cadena2[ 20 ];  
cin >> cadena2;
```

Es responsabilidad del programador asegurar que el arreglo en el que se coloque la cadena sea capaz de contener cualquier cadena que el usuario escriba en el teclado.

```
int main(){  
    char cadena1[ 20 ];  
    char cadena2[] = "literal de cadena";  
  
    // lee la cadena del usuario y la coloca en el arreglo cadena1  
    cout << "Escriba la cadena \"hola todos\": ";  
    cin >> cadena1;  
  
    cout << "cadena1 es: " << cadena1 << "\ncadena2 es: " << cadena2;  
    cout << "\ncadena1 con espacios entre caracteres es:\n";  
  
    // imprime caracteres hasta llegar al caracter nulo  
    for ( int i = 0; cadena1[ i ] != '\0'; i++){  
        cout << cadena1[ i ] << ' ';  
    }  
  
    cin >> cadena1; // lee "todos"  
    cout << "\ncadena1 es: " << cadena1 << endl; // NOTE: cuidado con  
    la linea fantasma. aca debe usar getline(cin,string)  
  
    return 0;  
}
```

## Arreglos locales estáticos

```
void inicArregloStatic( void )    E. Codigo
{                                example5_arreglosstatic
// inicializa con 0 la primera vez que se llama a la función
    static int arreglo1[ 3 ];

    cout << "\nValores al entrar en inicArregloStatic:\n";

    for ( int i = 0; i < 3; i++ ){
        cout << "arreglo1[" << i << "] = " << arreglo1[ i ] << "
";
    }

    cout << "\nValores al salir de inicArregloStatic:\n";

    // modifica e imprime el contenido de arreglo1
    for ( int j = 0; j < 3; j++ )
        cout << "arreglo1[" << j << "] = " << ( arreglo1[ j ] +=
5 ) << " ";
    }
```

Podemos aplicar `static` a la declaración de un arreglo local, de manera que el arreglo no se cree e inicialice cada vez que el programa llame a la función, y no se destruya cada vez que termine la función en el programa. **Esto puede mejorar el rendimiento, en especial cuando se utilizan arreglos extensos.**

## Paso de arreglos a funciones

**C++ pasa los arreglos a las funciones por referencia.**

las funciones llamadas pueden modificar los valores de los elementos en los arreglos originales.

El paso de arreglos por referencia tiene sentido por cuestiones de rendimiento. Si los arreglos se pasaran por valor, se pasaría una copia de cada elemento. Para los arreglos extensos que se pasan con frecuencia, esto requeriría mucho tiempo y una cantidad considerable de almacenamiento para las copias de los elementos del arreglo.

Observe la extraña apariencia del prototipo  
**void FunA( int [], int );** //Arreglo y tamaño

**C++ ignoran los nombres de las variables en los prototipos**

**void FunArreglo( int NameA[], int VariableSizeA);**

E. Codigo example6\_modificarA

```
#include <iostream>
#include <iomanip>

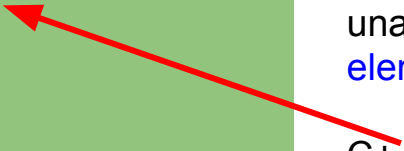
using namespace std;

void tratarDeModificarArreglo( const int [] );

int main()
{
    int a[] = { 10, 20, 30 };

    tratarDeModificarArreglo( a );
    cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
    return 0;
}

void tratarDeModificarArreglo( const int b[] )
{
    b[ 0 ] /= 2; // error de compilación
    b[ 1 ] /= 2;
    b[ 2 ] /= 2;
}
```



## Principio de menor privilegio.

Las funciones no deben recibir la capacidad de modificar un arreglo, a menos que sea absolutamente necesario

se puede encontrar con situaciones en las que una función **no tenga permitido modificar los elementos de un arreglo.**

C++ cuenta con el calificador de tipos **const**.

Cuando una función especifica un parámetro tipo arreglo al que se antepone el calificador **const**, los elementos del arreglo se hacen constantes en el cuerpo de la función

E. Código librocalicar1 (clase6)

E. Código  
example7\_busquedalineal

E. Código  
example8\_busquedainsercion