

## 4.ARRAYS

## ÍNDICE

<b>1</b>	<b>Introducción .....</b>	<b>2</b>
1.1	Características de los arrays .....	2
<b>2</b>	<b>Declaración de Arrays. ....</b>	<b>3</b>
<b>3</b>	<b>Operaciones con Arrays en C++:.....</b>	<b>4</b>
<b>4</b>	<b>Ejemplo con Arrays en C++:.....</b>	<b>6</b>
<b>5</b>	<b>Arrays multidimensionales C++: .....</b>	<b>8</b>
<b>6</b>	<b>Arrays como parámetros:.....</b>	<b>9</b>
<b>7</b>	<b>Búsqueda de elementos:.....</b>	<b>10</b>
<b>8</b>	<b>Tipos enumerados: .....</b>	<b>13</b>
<b>9</b>	<b>Registros:.....</b>	<b>15</b>
<b>10</b>	<b>Arrays de caracteres (cadena de caracteres):.....</b>	<b>18</b>

## 1 Introducción .

En C++, **un array es una colección contigua de elementos del mismo tipo**. Los arrays tienen un tamaño fijo que se especifica en el momento de la declaración. Los elementos dentro de un array se almacenan en posiciones consecutivas de memoria.

### 1.1 Características de los arrays

- **Homogéneos: Colección de Elementos del Mismo Tipo:** Un array en C++ es una estructura de datos que contiene un número fijo de elementos del mismo tipo de datos (por ejemplo, enteros, caracteres, objetos, etc.).
- **Ordenados:** Significa que hay un primer elemento, un segundo elemento, y así sucesivamente. Pueden ser seleccionados de forma directa, indicando la posición que ocupan dentro de la estructura.
- **Almacenamiento contiguo:** Los elementos de un array se almacenan en posiciones de memoria contiguas. Esto significa que los elementos están uno al lado del otro en la memoria, lo que facilita el acceso secuencial a los elementos mediante índices.
- **Índices:** Los elementos dentro de un array se acceden mediante un índice entero. El primer elemento tiene un índice de **0**, el segundo tiene un índice de **1** y así sucesivamente. El último elemento tiene un índice de **tamaño - 1**.
- **Finita:** Significa que hay también un último elemento.
- **Tamaño fijo.** El tamaño de un array se determina en el momento de su declaración y no puede cambiar durante la ejecución del programa. Esto significa que el número de elementos en el array es fijo y conocido en tiempo de compilación.
- **Nombre:** El nombre del array especifica la dirección de memoria del primer elemento del mismo.

## 2 Declaración de Arrays.

- **Sintaxis:**

La sintaxis para declarar un array en C++ es:

```
cpp Copy code  
  
tipo_de_dato nombre_array[tamaño];
```

- ``tipo_de_dato``: especifica el tipo de datos de los elementos del array.
- ``nombre_array``: es el nombre que le das al array.
- ``tamaño``: es el número de elementos que puede contener el array.

Por ejemplo, para declarar un array de enteros con 5 elementos:

```
cpp Copy code  
  
int numeros[5];
```

- **Definición de Tipos y Sentencia typedef:**

La sentencia ``typedef`` se utiliza para crear alias (nombres alternativos) para tipos de datos existentes en C++. Esto es útil para hacer el código más legible y para definir tipos personalizados.

Sintaxis ``typedef``

```
cpp Copy code  
  
typedef tipo_de_dato alias;
```

Por ejemplo, para crear un alias para un array de enteros:

```
cpp Copy code  
  
typedef int EnteroArray[5];
```

En este caso, ``EnteroArray`` es un alias para un array de 5 enteros.

### 3 Operaciones con Arrays en C++:

- **Acceso a Elementos:**

Los elementos de un array se acceden mediante su índice, que va desde `0` hasta `tamaño - 1`.

```
cpp Copy code  
  
int numeros[5] = {10, 20, 30, 40, 50};  
cout << numeros[2]; // Accede al tercer elemento (índice 2) -> Imprime 30
```

- **Asignación e igualdad:**

Los arrays no pueden ser asignados directamente unos a otros en C++. Para copiar el contenido de un array en otro, se deben usar bucles u otras técnicas.

Para comparar dos arrays, debes comparar elemento por elemento.

```
cpp Copy code  
  
int arr1[3] = {1, 2, 3};  
int arr2[3] = {1, 2, 3};  
  
// Comparación de arrays  
bool iguales = true;  
for (int i = 0; i < 3; ++i) {  
    if (arr1[i] != arr2[i]) {  
        iguales = false;  
        break;  
    }  
}  
  
if (iguales) {  
    cout << "Los arrays son iguales";  
} else {  
    cout << "Los arrays son diferentes";  
}
```

- **Lectura y Escritura:**

Para leer y escribir elementos en un array, se puede usar un bucle `for` para recorrer el array.

```
cpp Copy code  
  
int numeros[5];  
  
// Leer elementos del usuario  
for (int i = 0; i < 5; ++i) {  
    cout << "Ingrese el elemento " << i + 1 << ": ";  
    cin >> numeros[i];  
}  
  
// Mostrar elementos del array  
cout << "Elementos del array:";  
for (int i = 0; i < 5; ++i) {  
    cout << " " << numeros[i];  
}  
cout << endl;
```

- **Recorrido de Arrays:**

Puedes recorrer un array usando bucles `for` o `while` para realizar operaciones en cada elemento.

```
cpp Copy code  
  
int numeros[5] = {10, 20, 30, 40, 50};  
  
// Recorrer e imprimir elementos del array  
for (int i = 0; i < 5; ++i) {  
    cout << numeros[i] << " ";  
}  
cout << endl;  
  
// Duplicar cada elemento del array  
for (int i = 0; i < 5; ++i) {  
    numeros[i] *= 2;  
}  
  
// Mostrar elementos duplicados  
for (int i = 0; i < 5; ++i) {  
    cout << numeros[i] << " ";  
}  
cout << endl;
```

#### 4 Ejemplo con Arrays en C++:

Un ejemplo básico de cómo trabajar con arrays en C++. En este caso, crearemos un programa que declara, inicializa y luego imprime los elementos de un array de enteros.

```
cpp Copy code

#include <iostream>

using namespace std;

int main() {
    // Declaración e inicialización de un array de enteros
    int numeros[5] = {10, 20, 30, 40, 50};

    // Imprimir los elementos del array
    cout << "Elementos del array numeros:" << endl;
    for (int i = 0; i < 5; ++i) {
        cout << "Elemento " << i << ": " << numeros[i] << endl;
    }

    return 0;
}
```

##### Explicación del código:

1.

##### Declaración e Inicialización del Array:

```
cpp Copy code


int numeros[5] = {10, 20, 30, 40, 50};
```

- Aquí declaramos un array llamado `numeros` que puede contener 5 enteros. Los valores `{10, 20, 30, 40, 50}` son los elementos iniciales del array.

2.

**Imprimir los Elementos del Array:**

cpp


 Copy code

```
cout << "Elementos del array numeros:" << endl;
for (int i = 0; i < 5; ++i) {
    cout << "Elemento " << i << ": " << numeros[i] << endl;
}
```

- Usamos un bucle `for` para recorrer cada elemento del array `numeros`.
- `i` representa el índice del elemento actual.
- `numeros[i]` accede al elemento en la posición `i` del array.
- Imprimimos el índice (`i`) y el valor del elemento (`numeros[i`]).

**Salida Esperada:**

yaml

 Copy code

```
Elementos del array numeros:
Elemento 0: 10
Elemento 1: 20
Elemento 2: 30
Elemento 3: 40
Elemento 4: 50
```

Este programa declara un array de enteros, lo inicializa con valores específicos y luego imprime cada elemento del array junto con su índice correspondiente. Este es un ejemplo básico para entender cómo trabajar con arrays en C++.



## 5 Arrays multidimensionales C++:

Los arrays multidimensionales en C++ son arreglos que contienen otros arreglos. Pueden ser de dos dimensiones (matrices), tres dimensiones (cubos) o más. Aquí tienes ejemplos de cómo declarar, inicializar y acceder a arrays multidimensionales en C++:

### Arrays bidimensionales (matrices)

```
cpp Copy code

#include <iostream>

using namespace std;

int main() {
    // Declaración e inicialización de una matriz 2x3
    int matriz[2][3] = {{1, 2, 3}, {4, 5, 6}};

    // Acceso a elementos individuales de la matriz
    cout << "Elemento en la fila 1, columna 2: " << matriz[0][1] << endl; // Imprimir
    cout << "Elemento en la fila 2, columna 3: " << matriz[1][2] << endl; // Imprimir

    return 0;
}
```

### Arrays tridimensionales

```
cpp Copy code

#include <iostream>

using namespace std;

int main() {
    // Declaración e inicialización de una matriz 2x2x3
    int cubo[2][2][3] = {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}};

    // Acceso a elementos individuales del cubo
    cout << "Elemento en la capa 1, fila 1, columna 2: " << cubo[0][0][1] << endl; //
    cout << "Elemento en la capa 2, fila 2, columna 3: " << cubo[1][1][2] << endl; //

    return 0;
}
```

## 6 Arrays como parámetros:

Al igual que C++ pasa siempre por referencia los arrays para evitarse el tener que hacer una copia de memoria de una estructura que generalmente va a ser muy grande, tampoco permite retornar arrays en las funciones, por lo que, la única solución es añadir un parámetro de salida (por referencia) para el resultado.

Ejemplo:

En este programa, el procedimiento **LeeMatriz** toma una matriz **m** como parámetro de salida y utiliza un bucle **for** anidado para solicitar al usuario que ingrese los valores de la matriz uno por uno. Estos valores se almacenan en la matriz **m**. Luego, en la función **main**, llamamos a este procedimiento y mostramos la matriz ingresada por el usuario.

```
#include <iostream>

using namespace std;

// Definición del tipo de matriz
const int FILAS = 3;
const int COLUMNAS = 3;
typedef int TMatriz[FILAS][COLUMNAS];

// Procedimiento para leer una matriz desde teclado
void LeeMatriz(TMatriz &m) {
    cout << "Ingrese los valores de la matriz " << FILAS << "x" << COLUMNAS << ":\n";
    for (int i = 0; i < FILAS; ++i) {
        for (int j = 0; j < COLUMNAS; ++j) {
            cout << "Ingrese el valor para la posicion [" << i << "][" << j << "]: ";
            cin >> m[i][j];
        }
    }
}
```

```
int main() {
    TMatriz matriz;

    // Llamar al procedimiento para leer la matriz desde teclado
    LeeMatriz(matriz);

    // Mostrar la matriz ingresada
    cout << "La matriz ingresada es:\n";
    for (int i = 0; i < FILAS; ++i) {
        for (int j = 0; j < COLUMNAS; ++j) {
            cout << matriz[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

## 7 Búsqueda de elementos:

En C++ existen varios métodos de búsqueda que pueden ser utilizados para encontrar elementos en un array. Algunos de los métodos comunes son:

- **Búsqueda Lineal (Sequential Search):** Este método consiste en recorrer secuencialmente el array desde el principio hasta el final, comparando cada elemento con el valor buscado hasta que se encuentre el elemento o se llegue al final del array. Es eficiente para arrays pequeños o no ordenados, pero su complejidad temporal es  $O(n)$ , donde  $n$  es el tamaño del array.
- **Búsqueda Binaria (Binary Search):** Este método requiere que el array esté ordenado previamente. Funciona dividiendo repetidamente el array en mitades y comparando el elemento buscado con el elemento en la mitad del array. Si el elemento buscado es igual al elemento en la mitad, se ha encontrado el elemento. Si el elemento buscado es mayor, se busca en la mitad derecha del array, y si es menor, se busca en la mitad izquierda. Este proceso se repite hasta que se encuentre el elemento o hasta que el array se reduzca a cero. La complejidad temporal de este algoritmo es  $O(\log n)$ , donde  $n$  es el tamaño del array.

Ejemplos:

### Búsqueda Lineal:

```
cpp Copy code

#include <iostream>

using namespace std;

// Función para buscar un elemento en un array utilizando búsqueda lineal
int busquedaLineal(int arr[], int tamano, int elemento) {
    for (int i = 0; i < tamano; ++i) {
        if (arr[i] == elemento) {
            return i; // Devolver la posición del elemento si se encuentra
        }
    }
    return -1; // Devolver -1 si el elemento no se encuentra
}

int main() {
    int numeros[] = {10, 20, 30, 40, 50};
    int tamano = sizeof(numeros) / sizeof(numeros[0]);
    int elementoBuscado = 30;

    int posicion = busquedaLineal(numeros, tamano, elementoBuscado);

    if (posicion != -1) {
        cout << "El elemento " << elementoBuscado << " se encuentra en la posición "
    } else {
        cout << "El elemento " << elementoBuscado << " no se encuentra en el array."
    }

    return 0;
}
```

## Búsqueda Binaria:

```
cpp Copy code

#include <iostream>

using namespace std;

// Función para buscar un elemento en un array ordenado utilizando búsqueda binaria
int busquedaBinaria(int arr[], int inicio, int fin, int elemento) {
    while (inicio <= fin) {
        int medio = inicio + (fin - inicio) / 2;

        if (arr[medio] == elemento) {
            return medio; // Devolver la posición del elemento si se encuentra
        }

        if (arr[medio] < elemento) {
            inicio = medio + 1;
        } else {
            fin = medio - 1;
        }
    }

    return -1; // Devolver -1 si el elemento no se encuentra
}

int main() {
    int numeros[] = {10, 20, 30, 40, 50};
    int tamano = sizeof(numeros) / sizeof(numeros[0]);
    int elementoBuscado = 30;

    int posicion = busquedaBinaria(numeros, 0, tamano - 1, elementoBuscado);

    if (posicion != -1) {
        cout << "El elemento " << elementoBuscado << " se encuentra en la posición "
    } else {
        cout << "El elemento " << elementoBuscado << " no se encuentra en el array."
    }

    return 0;
}
```

## 8 Tipos enumerados:

Los tipos enumerados en C++ son una forma de definir un conjunto de constantes enteras con nombre. Esto facilita la escritura de código más legible y mantenible al asignar nombres descriptivos a valores específicos.

Aquí tienes un ejemplo de cómo se define y se utiliza un tipo enumerado en C++:

```
cpp Copy code  
  
#include <iostream>  
  
using namespace std;  
  
// Definición de un tipo enumerado llamado DiaSemana  
typedef enum {  
    LUNES,  
    MARTES,  
    MIERCOLES,  
    JUEVES,  
    VIERNES,  
    SABADO,  
    DOMINGO  
} DiaSemana;
```

```
int main() {  
    // Declaración de una variable del tipo enumerado DiaSemana  
    DiaSemana dia = MARTES;  
  
    // Utilización del tipo enumerado en una instrucción switch  
    switch (dia) {  
        case LUNES:  
            cout << "Hoy es Lunes." << endl;  
            break;  
        case MARTES:  
            cout << "Hoy es Martes." << endl;  
            break;  
        case MIERCOLES:  
            cout << "Hoy es Miércoles." << endl;  
            break;  
        case JUEVES:  
            cout << "Hoy es Jueves." << endl;  
            break;  
        case VIERNES:  
            cout << "Hoy es Viernes." << endl;  
            break;  
  
        case SABADO:  
            cout << "Hoy es Sábado." << endl;  
            break;  
        case DOMINGO:  
            cout << "Hoy es Domingo." << endl;  
            break;  
    }  
  
    return 0;  
}
```

En este ejemplo, hemos definido un tipo enumerado llamado **DiaSemana**, que contiene los días de la semana. Cada día tiene un valor asociado (**LUNES = 0**, **MARTES = 1**, etc.). Luego, en la función **main()**, declaramos una variable **dia** del tipo **DiaSemana** y la inicializamos con el valor **MARTES**. Finalmente, utilizamos una instrucción **switch** para imprimir el nombre del día de la semana en función del valor de la variable **dia**.

El **typedef** permite crear un alias para el tipo enumerado, en este caso, **DiaSemana**.

## 9 Registros:

En C++, los registros (también conocidos como estructuras) son un tipo de dato que te permite combinar diferentes tipos de datos bajo un solo nombre. Esto resulta útil cuando necesitas almacenar información relacionada de manera organizada.

Aquí tienes ejemplo de distintos usos de los registros en C++:

### Ejemplo 1: Array como parte del registro:

```
cpp Copy code

#include <iostream>
#include <string>

using namespace std;

// Definición de un registro llamado Estudiante
struct Estudiante {
    string nombre;
    int notas[3];
};

int main() {
    // Declaración de un arreglo de estructuras Estudiante
    Estudiante estudiantes[3];

    // Asignación de valores a los miembros de cada estructura en el arreglo
    estudiantes[0].nombre = "María";
    estudiantes[0].notas[0] = 85;
    estudiantes[0].notas[1] = 90;
    estudiantes[0].notas[2] = 75;

    estudiantes[1].nombre = "Juan";
    estudiantes[1].notas[0] = 70;
    estudiantes[1].notas[1] = 80;
    estudiantes[1].notas[2] = 65;

    estudiantes[2].nombre = "Ana";
    estudiantes[2].notas[0] = 95;
    estudiantes[2].notas[1] = 85;
    estudiantes[2].notas[2] = 90;
}
```



```
// Impresión de los valores de los estudiantes
for (int i = 0; i < 3; i++) {
    cout << "Nombre: " << estudiantes[i].nombre << endl;
    cout << "Notas: ";
    for (int j = 0; j < 3; j++) {
        cout << estudiantes[i].notas[j] << " ";
    }
    cout << endl;
}

return 0;
}
```

En este ejemplo, hemos definido un arreglo de estructuras llamado **estudiantes** que contiene información sobre tres estudiantes. Cada estructura tiene un miembro **nombre** de tipo string y un miembro **notas** que es un arreglo de int con tamaño 3. Luego, asignamos valores a los miembros de cada estructura en el arreglo y finalmente imprimimos esta información en la consola.

### Ejemplo 1: Array de un registro:

```
cpp Copy code

#include <iostream>
#include <string>

using namespace std;

// Definición de un registro llamado Persona
struct Persona {
    string nombre;
    int edad;
};
```

```
int main() {  
    // Declaración de un arreglo de estructuras Persona  
    Persona personas[3];  
  
    // Asignación de valores a los miembros de cada estructura en el arreglo  
    personas[0].nombre = "Juan";  
    personas[0].edad = 25;  
  
    personas[1].nombre = "María";  
    personas[1].edad = 30;  
  
    personas[2].nombre = "Pedro";  
    personas[2].edad = 40;  
  
    // Impresión de los valores de las personas  
    for (int i = 0; i < 3; i++) {  
        cout << "Persona " << i+1 << endl;  
        cout << "Nombre: " << personas[i].nombre << endl;  
        cout << "Edad: " << personas[i].edad << endl;  
        cout << endl;  
    }  
  
    return 0;  
}
```

En este ejemplo, hemos definido una estructura llamada **Persona** que tiene dos miembros: **nombre** (de tipo string) y **edad** (de tipo int). Luego, creamos un **arreglo de estructuras Persona** llamado **personas** que contiene información sobre tres personas. Después, asignamos valores a los miembros de cada estructura en el arreglo y finalmente imprimimos esta información en la consola.

## 10 Arrays de caracteres (cadena de caracteres):

En C++, los **arrays de caracteres**, también conocidos como **cadenas de caracteres o strings**, son una secuencia de caracteres almacenados consecutivamente en la memoria. Cada carácter en la cadena se representa como un elemento del array, **y la cadena se termina con el carácter nulo '\0'**.

Un ejemplo de nivel intermedio que muestra cómo trabajar con arrays de caracteres en C++:

```
#include <iostream>
#include <cstring> // Incluir la biblioteca para trabajar con cadenas de caracteres

using namespace std;

int main() {
    // Declarar un array de caracteres para almacenar una cadena
    char nombre[50];

    // Pedir al usuario que ingrese su nombre
    cout << "Por favor, ingrese su nombre: ";
    cin.getline(nombre, 50); // Leer la cadena de caracteres (incluyendo espacios)

    // Mostrar el nombre ingresado
    cout << "¡Hola, " << nombre << "!" << endl;

    // Calcular la longitud de la cadena ingresada
    int longitud = strlen(nombre);

    // Mostrar la longitud de la cadena
    cout << "La longitud de su nombre es: " << longitud << " caracteres." << endl;

    // Cambiar la primera letra del nombre a mayúscula
    nombre[0] = toupper(nombre[0]); // La función toupper convierte un carácter a mayúscula

    // Mostrar el nombre con la primera letra en mayúscula
    cout << "Su nombre con la primera letra en mayúscula es: " << nombre << endl;

    return 0;
}
```

En este ejemplo:

- Se declara un **array de caracteres** llamado nombre con capacidad para 50 caracteres.
- Se utiliza la **función cin.getline()** para leer una línea de texto, incluyendo espacios, desde la entrada estándar (cin), y se almacena en el array nombre.
- Se calcula la longitud de la cadena ingresada utilizando la función strlen() de la biblioteca **<cstring>**.
- Se convierte la primera letra del nombre a mayúscula utilizando la **función toupper()**.
- Finalmente, se muestra el nombre ingresado, su longitud y el nombre con la primera letra en mayúscula.