



# PROGRAMACION AVANZADA

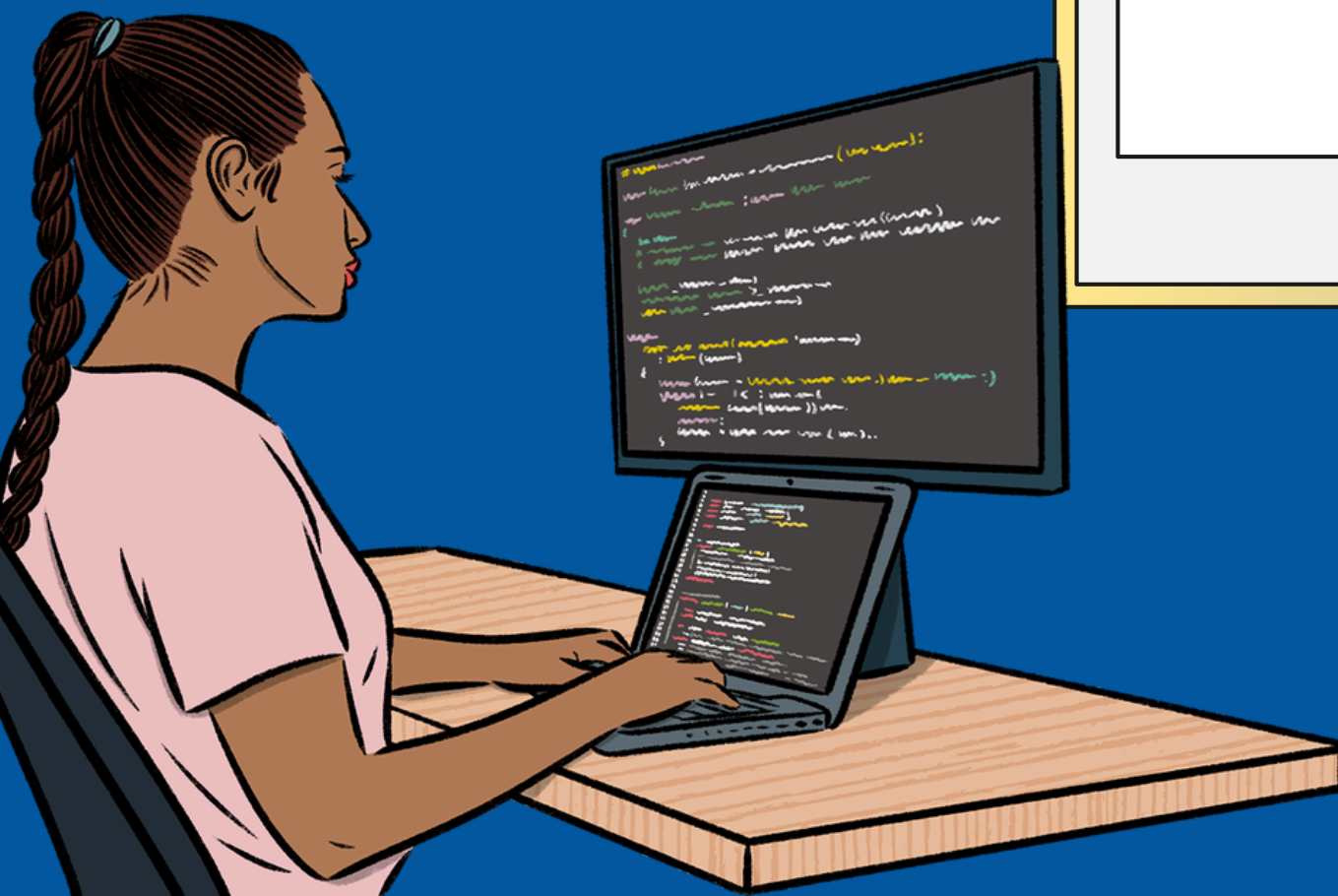
ING. INDUSTRIAL

Esp. Ing.Prof.: Cristina Delia Cruz





# TIPOS ESTRUCTURADOS EN C++





# TIPOS ESTRUCTURADOS EN C++

Los tipos de datos estructurados en C++ **son aquellos que permiten agrupar múltiples elementos de datos**, que pueden ser del mismo tipo o de tipos diferentes, en una única estructura. Estos tipos de datos son fundamentales cuando se necesita trabajar con grandes conjuntos de datos de forma organizada y eficiente. A continuación, veremos algunos de los principales tipos de datos estructurados en C++.



# DATOS SIMPLES Y COMPUESTOS

Las variables de tipos de datos simples permiten almacenar un valor específico.



Las variables de tipos de datos compuestos permiten almacenar un grupo de valores.







# ARREGLOS EN C++

**Un Arreglo es una colección de elementos del mismo tipo almacenados de manera contigua en la memoria.** En C++, los arreglos permiten acceder a cada uno de sus elementos mediante índices. Los arreglos son útiles para trabajar con múltiples valores relacionados, pero todos los elementos deben ser del mismo tipo.





# ARREGLOS

## UNIDIMENSIONALES EN C++

**Un arreglo unidimensional** (también conocido como vector) **es una colección de elementos del mismo tipo de datos que se almacenan en ubicaciones contiguas de la memoria.** En términos simples, **es una estructura de datos que contiene una lista de valores accesibles a través de un índice.** Los arreglos son ampliamente utilizados en C++ y otros lenguajes de programación para almacenar y gestionar grandes cantidades de datos de manera organizada y eficiente.



# ARREGLOS

## UNIDIMENSIONALES EN C++

Elementos homogéneos  
Todos los elementos dentro del arreglo son del mismo tipo de datos (por ejemplo, todos int, todos float, etc.).





# ARREGLOS UNIDIMENSIONALES EN C++

Acceso mediante índices  
Cada elemento del arreglo  
tiene una posición o índice que  
**comienza desde 0.**



# ARREGLOS UNIDIMENSIONALES EN C++

## Tamaño fijo

El tamaño del arreglo se define al declararlo y no puede cambiar durante la ejecución del programa (en caso de usar arreglos estáticos).

Sin embargo, en C++, podemos usar la biblioteca estándar (`std::vector`) para tener arreglos dinámicos que pueden redimensionarse.





# DECLARACIÓN DE ARREGLOS UNIDIMENSIONALES

En C++, un arreglo unidimensional se declara especificando el tipo de datos y el número de elementos que contendrá.

**tipo nombreArreglo[tamaño];**



# DECLARACIÓN DE ARREGLOS UNIDIMENSIONALES

**tipo nombreArreglo[tamaño];**

- **tipo**: Es el tipo de datos que almacenará el arreglo (por ejemplo, **int**, **float**, **char**, etc.).
- **nombreArreglo**: Es el nombre del arreglo.
- **tamaño**: Es el número de elementos que contendrá el arreglo. Debe ser un valor constante o definido durante la declaración.



# EJEMPLO DE DECLARACIÓN DE ARREGLO UNIDIMENSIONAL

```
int numeros[5];
```

En este ejemplo, hemos declarado un arreglo llamado numeros que puede almacenar 5 enteros. **Los elementos se acceden mediante sus índices, que van desde 0 hasta 4.**





# INICIALIZACIÓN DE ARREGLOS UNIDIMENSIONALES

**Un arreglo puede ser inicializado durante su declaración, asignando valores a cada elemento.**

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

**Si no se especifican todos los valores, los elementos restantes se inicializan a 0 de forma predeterminada (en el caso de los enteros).**



# INICIALIZACIÓN DE ARREGLOS UNIDIMENSIONALES

Para acceder a un elemento de un arreglo, se utiliza el nombre del arreglo seguido del índice entre corchetes “[ ]”.

```
int numeros[5] = {10, 20, 30, 40, 50};  
int primerElemento = numeros[0];  
int tercerElemento = numeros[2];
```



# MODIFICACIÓN DE ELEMENTOS DE UN ARREGLO

Podemos cambiar el valor de un elemento del arreglo usando su índice.

```
numeros[1] = 25;
```

Cambia el segundo elemento de 20 a 25





# RECORRER UN ARREGLO

Es común utilizar bucles para recorrer los elementos de un arreglo y realizar operaciones sobre ellos. El bucle más utilizado es for.



# RECORRER UN ARREGLO

## Ejemplo

```
1  #include <iostream>
2  using namespace std;
3  main() {
4      int numeros[5] = {10, 20, 30, 40, 50};
5
6      for (int i = 0; i < 5; i++) {
7          cout << numeros[i] << " ";
8      }
9      return 0;
10 }
```

En este ejemplo, el bucle recorre todos los elementos del arreglo y los imprime.



# LÍMITES DE LOS ARREGLOS

**Es importante recordar que los arreglos tienen un tamaño fijo, y acceder a un índice fuera de sus límites puede causar errores o comportamientos inesperados. Este es un error común conocido como desbordamiento de arreglo.**



# DESVENTAJAS DE LOS ARREGLOS

## **Tamaño fijo**

Una vez que se declara el tamaño del arreglo, no se puede cambiar. Si el tamaño es demasiado grande, se desperdicia memoria, y si es muy pequeño, puede ser insuficiente.

## **Acceso fuera de límites**

C++ no verifica los límites del arreglo en tiempo de ejecución, lo que puede conducir a errores si se intenta acceder a índices no válidos.





# CONCLUSION DE LOS ARREGLOS UNIDEMENSIONALES

Los arreglos unidimensionales son una estructura de datos básica y fundamental en C++. Son útiles cuando se requiere almacenar múltiples valores del mismo tipo de manera organizada. Sin embargo, es crucial utilizarlos con cuidado para evitar problemas como el acceso fuera de límites o el mal manejo del tamaño del arreglo. Para más flexibilidad, se puede optar por tipos de datos dinámicos como los vectores.



# ARREGLOS BIDIMENSIONALES EN C++

**Un arreglo bidimensional en C++ es una estructura de datos que permite almacenar datos en una matriz de filas y columnas.** Es una extensión del arreglo unidimensional y puede visualizarse como una **tabla de datos** (como una hoja de cálculo o una tabla de Excel). Es útil cuando se quiere trabajar con relaciones de datos organizados en dos dimensiones, como por ejemplo matrices, tablas de valores o datos tabulares.



# DECLARACIÓN DE ARREGLOS BIDIMENSIONALES

Para declarar un arreglo bidimensional en C++, se utiliza la siguiente sintaxis:

```
tipo nombreArreglo[filas][columnas];
```



# DECLARACIÓN DE ARREGLOS BIDIMENSIONALES

**tipo nombreArreglo[filas][columnas];**

- **tipo:** Tipo de dato que almacenará el arreglo (por ejemplo, int, double, char).
- **nombreArreglo:** Nombre del arreglo.
- **filas:** Número de filas del arreglo.
- **columnas:** Número de columnas del arreglo.



# EJEMPLO DE DECLARACIÓN

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

Este arreglo contiene 12 elementos (3 filas \* 4 columnas). Cada posición en este arreglo puede ser identificada por dos índices: el índice de la fila y el índice de la columna.





# ACCESO A LOS ELEMENTOS DEL ARREGLO

Para acceder a un elemento específico del arreglo bidimensional, se utiliza la siguiente notación:

`nombreArreglo[indiceFila][indiceColumna];`

Por ejemplo, si tenemos un arreglo de 3 filas y 4 columnas llamado `matriz` y queremos acceder al valor en la segunda fila y tercera columna, lo haríamos así:

```
int valor = matriz[1][2];
```



# INICIALIZACIÓN DE UN ARREGLO BIDIMENSIONAL

Podemos inicializar un arreglo bidimensional de forma manual en el momento de la declaración:

```
int matriz[2][3] = {  
    {1, 2, 3},  
    {4, 5, 6}  
};
```



# INICIALIZACIÓN DE UN ARREGLO BIDIMENSIONAL

En este caso, tenemos un arreglo con 2 filas y 3 columnas. Los valores se almacenan en el arreglo de la siguiente manera:

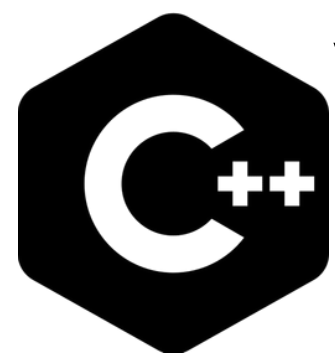
|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |



# RECORRIDO DE UN ARREGLO BIDIMENSIONAL

Para recorrer todos los elementos de un arreglo bidimensional, se utilizan dos bucles 'for' anidados: uno para recorrer las filas y otro para las columnas.

```
for (int i = 0; i < filas; i++) {  
    for (int j = 0; j < columnas; j++) {  
        cout << matriz[i][j] << " ";  
    }  
    cout << endl;  
}
```



# VENTAJAS DE LOS ARREGLOS BIDIMENSIONALES

1. **Organización de datos:** Permiten organizar datos en una estructura tabular, lo que facilita su manejo y comprensión.
2. **Acceso directo:** Permiten el acceso rápido a cualquier elemento usando índices de filas y columnas.
3. **Facilidad para realizar operaciones matemáticas:** En campos como la Ingeniería Industrial, las matrices son esenciales para cálculos de optimización, costos, y operaciones en general.





# LIMITACIONES

1. **Tamaño fijo:** Una vez que se declara un arreglo bidimensional, su tamaño es fijo y no se puede cambiar durante la ejecución del programa.
2. **Uso de memoria:** Los arreglos bidimensionales ocupan más memoria, lo que puede ser un problema en programas que manejan grandes cantidades de datos.



# CONCLUSION

Los arreglos bidimensionales son muy versátiles y se utilizan ampliamente en problemas donde se necesita almacenar y manipular datos en dos dimensiones.



# ARREGLOS DE CARACTERES

En C++, un arreglo de caracteres es una secuencia de caracteres almacenados en posiciones consecutivas en memoria. Los arreglos de caracteres se usan comúnmente para representar cadenas de texto. Aquí están algunas operaciones comunes:



# ARREGLOS DE CARACTERES

Puedes definir e inicializar un arreglo de caracteres de esta forma:

```
char arreglo[] = "Hola Mundo"; // El tamaño del arreglo se ajusta automáticamente
```

O especificar el tamaño:

```
char arreglo[20] = "Hola Mundo"; // El tamaño es 20, suficiente para "Hola Mundo" y el carácter nulo
```

Acceso a Elementos: Puedes acceder a elementos individuales del arreglo usando el índice:

```
char arreglo[] = "Hola";
```

```
char primerCaracter = arreglo[0]; // 'H'
```

Modificación de Elementos: Puedes modificar caracteres específicos del arreglo:

```
char arreglo[] = "Hola";
```

```
arreglo[0] = 'h'; // Cambia 'H' a 'h'
```



# ARREGLOS DE CARACTERES

Longitud de una Cadena: La longitud de un arreglo de caracteres se puede encontrar usando la función `strlen()` de `<cstring>`:

```
#include <iostream>
#include <cstring>

int main() {
    char arreglo[] = "Hola Mundo";
    int longitud = strlen(arreglo); // 10
    cout << "Longitud: " << longitud << endl;
    return 0;
}
```





# ARREGLOS DE CARACTERES

Comparación de Cadenas: Para comparar dos arreglos de caracteres (cadenas), puedes usar `strcmp()` de `<cstring>`:

```
#include <iostream>
#include <cstring>

int main() {
    char cadena1[] = "Hola";
    char cadena2[] = "Hola";
    if (strcmp(cadena1, cadena2) == 0) {
        cout << "Las cadenas son iguales." << endl;
    } else {
        cout << "Las cadenas son diferentes." << endl;
    }
    return 0;
}
```





# ARREGLOS DE CARACTERES

Copia de Cadenas: Para copiar el contenido de una cadena a otra, usa strcpy():

```
#include <iostream>
#include <cstring>

int main() {
    char origen[] = "Hola Mundo";
    char destino[20];
    strcpy(destino, origen); // Copia "Hola Mundo" a destino
    cout << "Destino: " << destino << endl;
    return 0;
}
```



# ARREGLOS CON CADENAS

En C++, también puedes tener arreglos de cadenas (arreglos de arreglos de caracteres). Esto es útil para almacenar varias cadenas de texto.

Definición e Inicialización: Puedes definir un arreglo de cadenas de esta manera:

```
char cadenas[3][20] = { "Primera", "Segunda", "Tercera" };
```



# ARREGLOS CON CADENAS

Acceso a Elementos: Puedes acceder a cadenas específicas e incluso a caracteres dentro de esas cadenas:

```
#include <iostream>

int main() {
    char cadenas[3][20] = { "Primera", "Segunda", "Tercera" };
    cout << "Primera cadena: " << cadenas[0] << endl;
    cout << "Caracter en la segunda cadena: " << cadenas[1][2] << endl;
    return 0;
}
```



# ARREGLOS CON CADENAS

Modificación de Elementos: Puedes modificar caracteres específicos en un arreglo de cadenas:

```
#include <iostream>

int main() {
    char cadenas[3][20] = { "Primera", "Segunda", "Tercera" };
    cadenas[0][0] = 'p'; // Cambia 'P' a 'p' en la primera cadena
    cout << "Primera cadena modificada: " << cadenas[0] << endl;
    return 0;
}
```



# ARREGLOS CON CADENAS

## Lectura de cadenas

El objeto **cin** del flujo de entrada incluye entre otras las funciones de entrada:

**cin.getline( )**: permite a cin leer la cadena completa, incluyendo cualquier espacio en blanco

**cin.get( )**: se utiliza para leer carácter a carácter

**cin.ignore( )**: lee uno o más caracteres del flujo de entrada cin sin procesar

Nota: también puede utilizarse **fflush(stdin)** para limpiar el **buffer de entrada**





# ARREGLOS CON CADENAS

Para trabajar con estas funciones, al comienzo del programa hay que escribir

```
#include <string.h>
```

La biblioteca `<string.h>` contiene un conjunto de funciones para manipular cadenas: copiar, cambiar caracteres, comparar cadenas, etc.

Las funciones más elementales son:

**strcpy ( c1, c2 );** Copia c2 en c1

**strcat ( c1, c2 );** Añade c2 al final de c1

**strlen ( cadena );** Devuelve la longitud de la cadena

**strcmp ( c1, c2 );** Devuelve cero si c1 es igual a c2; no-cero en caso contrario

**strlwr(c1);** Convierte todos los caracteres de la cadena c1 a minúscula

**strupr(c1);** Convierte todos los caracteres de la cadena c1 a mayúsculas





# EJEMPLO 1: CONTROL Y REGISTRO DE STOCK BÁSICO

Desarrollar un programa en C++ que permita el almacenamiento, carga y visualización de un conjunto homogéneo de datos numéricos mediante un arreglo unidimensional estático.

Requisitos:

1. Declarar un arreglo unidimensional de tipo entero con una capacidad fija de 5 elementos.
2. Implementar un bucle interactivo (for) que solicite al usuario el ingreso secuencial de cada valor numérico por consola.
3. Utilizar un segundo bucle para recorrer el vector y mostrar en pantalla cada elemento junto con su respectivo índice de posición (desde 0 hasta 4).



# EJEMPLO 2: PROCESAMIENTO Y MODIFICACIÓN DE CALIFICACIONES

Aplicar la inicialización directa de un arreglo unidimensional y manipular el acceso a posiciones específicas para realizar cálculos estadísticos básicos (suma y promedio).

Requisitos:

1. Inicializar de forma estática un arreglo de tipo float con 4 calificaciones predefinidas (por ejemplo: 8.5, 7.0, 9.2, 6.5).
2. Modificar explícitamente el valor almacenado en la segunda posición del arreglo (índice 1) para actualizarlo a un nuevo valor (7.5).
3. Recorrer la estructura mediante un bucle para acumular la sumatoria total de los elementos y calcular el promedio aritmético, imprimiendo ambos resultados por consola.



# EJEMPLO 3: REPRESENTACIÓN Y RECORRIDO DE UNA MATRIZ ESTÁTICA

Comprender la estructura lógica de los arreglos bidimensionales (matrices) como tablas de filas y columnas, visualizando su contenido mediante bucles anidados.

Requisitos:

1. Declarar e inicializar estáticamente una matriz de enteros de 2 filas por 3 columnas con valores numéricos enteros.
2. Implementar una estructura de dos bucles for anidados (el bucle externo para las filas  $i$  y el interno para las columnas  $j$ ).
3. Mostrar los elementos en pantalla imitando una disposición tabular limpia, utilizando saltos de línea al finalizar cada fila.



# EJEMPLO 4: CARGA INTERACTIVA Y DESPLIEGUE DE DATOS TABULARES

Diseñar un sistema de entrada de datos bidimensional donde el usuario sea quien alimente la matriz en tiempo de ejecución.

Requisitos:

1. Declarar una matriz bidimensional de 3 filas y 2 columnas con tipo de dato entero sin inicializar.
2. Utilizar bucles anidados combinados con la instrucción cin para solicitar dinámicamente al usuario el valor correspondiente a cada celda específica [fila][columna].
3. Volver a recorrer la matriz con otra estructura de bucles anidados para imprimir la tabla completa resultante en la consola.



# EJEMPLO 5: MANIPULACIÓN A NIVEL DE CARACTER EN CADENAS TRADICIONALES

Analizar las cadenas de texto en C++ como arreglos unidimensionales de tipo char que finalizan en un carácter nulo, permitiendo la modificación de caracteres individuales.

Requisitos:

1. Declarar e inicializar un arreglo de caracteres (char) con una palabra de longitud fija (por ejemplo, "Programacion" con capacidad para 20 caracteres).
2. Imprimir el arreglo completo y acceder explícitamente a su primer elemento utilizando su índice ([0]) para mostrarlo de forma aislada.
3. Modificar dicho carácter inicial asignándole una letra minúscula ('p') y mostrar el texto resultante para evidenciar el cambio.





# EJEMPLO 6: CAPTURA SEGURA DE CADENAS CON ESPACIOS EN BLANCO

Resolver las limitaciones de lectura estándar implementando funciones de entrada avanzadas para capturar oraciones completas que incluyan espacios.

Requisitos:

1. Declarar un arreglo de caracteres de tipo char con capacidad suficiente para albergar un nombre completo (ej. 50 caracteres).
2. Utilizar la función del flujo de entrada `cin.getline()` para permitir que el usuario ingrese una cadena de texto que contenga espacios en blanco sin interrumpir la lectura.
3. Imprimir un mensaje personalizado por pantalla utilizando la cadena ingresada de forma íntegra.





# EJEMPLO 7: OPERACIONES AVANZADAS CON LA BIBLIOTECA <STRING.H>

Utilizar las funciones elementales de manipulación de cadenas proporcionadas por la biblioteca estándar de C++ para medir, copiar, concatenar y comparar textos.

Requisitos:

1. Incluir obligatoriamente la cabecera `#include <string.h>` en la estructura del código fuente.
2. Implementar la función `strlen()` para obtener y mostrar la longitud de una cadena base.
3. Utilizar `strcpy()` para duplicar el contenido de una cadena en un arreglo de destino, y `strcat()` para concatenar un texto adicional al final de la copia.
4. Evaluar la igualdad de dos cadenas mediante la función `strcmp()`, aplicando una estructura condicional (if/else) para imprimir el resultado de la comparación.





# EJEMPLO 8: MATRICES DE CADENAS Y FUNCIONES DE FORMATO DE TEXTO

Manejar arreglos bidimensionales de caracteres (arreglos de cadenas) combinándolos con funciones de conversión de mayúsculas y minúsculas.

Requisitos:

1. Declarar un arreglo bidimensional de caracteres que contenga un listado de al menos 3 palabras o asignaturas (ej. `char asignaturas[3][20]`).
2. Mostrar el listado original recorriendo las filas de la matriz mediante un bucle.
3. Aplicar la función `strupr()` a la primera cadena para transformarla enteramente a mayúsculas y la función `strlwr()` a la segunda cadena para convertirla a minúsculas, visualizando los cambios por consola.



# ¿PREGUNTAS?

```
    }  
    render() {  
      return (  
        <React.Fragment>  
          <div className="py-5">  
            <div className="container">  
              <Title name="our" title="product">  
                <div className="row">  
                  <ProductConsumer>  
                    {(value) => {  
                      console.log(value)  
                    }}  
                  </ProductConsumer>  
                </div>  
              </div>  
            </div>  
          </React.Fragment>  
        )  
      )  
    }  
  }  
}
```





MUCHAS  
**GRACIAS!!!**

*Nos Vemos en la Siguiente Clase*