# Tipos estructurados en C++

UNIDAD 3

PROGRAMACIÓN AVANZADA

# Tipos estructurados en C++

Arreglos y Registros. Declaración. Acceso a elementos. Operaciones básicas. Combinación arreglos y registros.

# Cadenas

ARREGLO DE CARACTERES

# CADENAS

Una cadena en C++ es un conjunto de caracteres, o valores de tipo char, terminados con el carácter nulo, es decir el valor numérico 0.

La manera de definir una cadena es la siguiente:

char <identificador> [<longitud máxima>];

# CADENAS

Cuando se declara una cadena hay que tener en cuenta que tendremos que reservar una posición para almacenar el carácter nulo terminador, de modo que si queremos almacenar la cadena "HOLA", tendremos que declarar la cadena como:

```
char Saludo[5];
```

# CADENAS

Es muy importante tener presente que, en C++, los índices tomarán valores empezando siempre en cero, así el primer carácter de nuestra cadena sería Saludo[0], que es la letra 'H'.

En un programa C++, una cadena puede almacenar informaciones en forma de texto, como nombres de personas, mensajes de error, números de teléfono, etc.

La asignación directa sólo está permitida cuando se hace junto con la declaración.

El siguiente ejemplo producirá un error en el compilador, ya que una cadena definida de este modo se considera una constante.

```
char Saludo[5];
Saludo = "HOLA“;
```

# CADENAS

La manera correcta de asignar una cadena es:

```
char Saludo[5];
Saludo[0] = 'H';
Saludo[1] = 'O';
Saludo[2] = 'L';
Saludo[3] = 'A';
Saludo[4] = 0;
```

O bien:

```
char Saludo[5] = "HOLA";
```

# CADENAS

Existen muchas funciones, que permiten compararlas, copiarlas, calcular su longitud, imprimirlas, visualizarlas, etc.

En C existe una biblioteca estándar (disponible en todos los compiladores), llamada precisamente string.

En C++ también existe una biblioteca para manipular cadenas, aunque en este caso se trata de una biblioteca de clases.

# ESCRITURA DE CADENAS

**Función puts**

La función puts simplemente imprime una cadena de caracteres en la salida estándar (y produce un salto de línea). Le debemos proporcionar la dirección donde encontrar la cadena de caracteres. El código:

```
#include <stdio.h>
main() {
        puts("Bienvenido a la programación");
        puts(" en lenguaje C");
}
```

produce el resultado:

```
Bienvenido a la programación
en lenguaje C
```

# LECTURA DE CADENAS

## Función gets

La función gets simplemente toma una cadena de caracteres de la entrada estándar (cuya introducción es preciso terminar con un ENTER) y la almacena en una variable string. Supongamos este código:

```
#include <stdio.h>
main() {
        char cadena[50];
        puts(“Ingrese su nombre:");
        gets(cadena);
        puts(“Hola ");
        puts(cadena);
}
```

La declaración char cadena[50]; crea una variable llamada cadena que puede almacenar hasta 50 caracteres. Este código produce, cuando escribimos con el teclado el texto Bienvenido a la programación en lenguaje C, el resultado:

```
Ingrese su nombre:
JUAN PEREZ
Hola
JUAN PEREZ
```

# BIBLIOTECA DE MANEJO DE CADENAS 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

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

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

# CADENAS - EJEMPLO

```
#include <iostream>
#include <string.h>
#include<locale.h>
using namespace std;
int main() {
	setlocale(LC_ALL, "");
	char completo [80];
	char nombre[32] = “José Francisco";
	char apellidos [32] = “de San Martín";
	/* Construye el nombre completo */
	strcpy ( completo, nombre );		/* completo <- “Jose Francisco" */
	strcat ( completo, " ");		/* completo <- “Jose Francisco " */
	strcat ( completo, apellidos );		/* completo <- “Jose Francisco de San Martin" */
	cout<< "El nombre completo es “<< completo ;
	return 0;
}
```

# Declaración y Asignación a un tipo string.

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
//forma #1 de inicialización
//aunque prácticamente no se usa
string cad_1("Hola mundo");
//forma #2 de inicializar y asignar una cadena
string cad_2 = "Segunda forma";

//asignar/copiar cad_1 en cad_2
cad_2 = cad_1;

//asiganr un solo carcter a un tipo string
cad_1 = 'P';

return 0;
}
```

# Comparaciones entre strings.

La comparación entre objetos ***string*** se puede llevar a cabo fácilmente mediante el uso de los operadores ==, <=, >=, <, >, !=, que son los mismos que se usan para las operaciones de tipo lógicas, hay que aclarar que ***se distingue*** entre mayúsculas y minúsculas.

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main()
7  {
8  //creamos y asignamos las cadenas str1 y str2
9  string str1="abcd", str2="abcd";
10 cout<<endl;
11 //comparamos si las cadenas son iguales
12 if(str1==str2)
13 {
14 cout<<"str1 es igual a str2"<<endl<<endl;
15 }
16
17 return 0;
18 }
```

# Concatenación de Strings.

El operador ‘+’ (más) permite concatenar dos o más cadenas, entonces en el siguiente ejemplo si tengo a str1, str2 que contienen las cadenas “Julio” y “Cesar” respectivamente se puede unir (concatenar) el contenido de str2 a str1 solo poniendo str1 = str1 + str2, o cómo óptimo a la hora de escribir con el operador **+=**, entonces se tiene str1**+=**str2 y con esto, la cadena 1 debería contener ambos nombres, el código es:

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main()
7  {
8  //definimos e inicializamos ambas cadenas
9  string str1="Julio ", str2="Cesar";
10
11 //realizamos la concatenación de ambas variables
12 str1+=str2;
13
14 //mostramos el nuevo contendio de str1
15 cout<<"El nuevo contenido de str1 es: "<<str1<<endl;
16 return 0;
17 }
```

# String. Mayúsculas y Minúsculas. Longitud

toupper() //convierte a mayúscula

tolower() // convierte a minúscula,

***Inconveniente**: Lo hace carácter por carácter.*

length() // método que permite obtener la longitud de una cadena.

**Ejemplo.**

```
string  cadena="Hola Mundo";
cout<<"La longitud de cadena es: "<<cadena.length();
```

# Práctica #1

Realizar un programa que solicite 3 palabras, las convierta a mayúscula y las muestre ordenadas alfabéticamente.

# ARREGLOS

# Tipos de Arreglos

## UNIDIMENSIONAL

Son utilizados para almacenar múltiples valores en una única variable.

Este tipo de arrays (vectores), permiten almacenar muchos valores en posiciones de memoria continuas, lo cual permite acceder a un valor u otro de manera rápida y sencilla. Estos valores pueden ser números, letras o cualquier tipo de variable que deseemos incluso tipos de datos complejos.

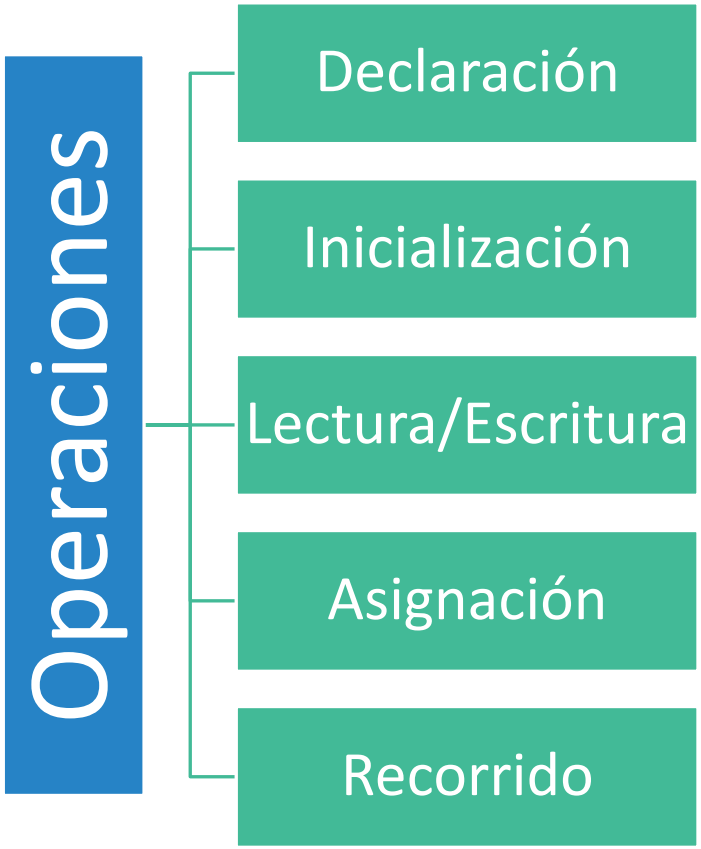
## BIDIMENSIONAL

Una matriz es una estructura conformada por filas y columnas, idealmente más de dos filas y columnas.

La intersección de una fila y una columna de la matriz son las casillas y cada una de ellas podrá poseer información, simple o compleja.

Una matriz es un vector cuyas posiciones (de la cero a la n) son, cada una de ellas, otro vector.

# Arreglos


Operaciones
Declaración
Inicialización
Lectura/Escritura
Asignación
Recorrido

# Arreglo Unidimensional

## Declaración:

La forma de declarar un arreglo unidimensional en C/C++ es:

```
tipoDato  nombreArreglo[numeroElementos];
```

**tipoDato:** Tipo de dato que tendrán los elementos del arreglo.

**nombreArreglo:** Nombre que asignamos para referirnos al arreglo.

**numeroElementos:** Valor entero que delimita el tamaño del arreglo.

# Arreglo Unidimensional

- En C/C++, los elementos de un arreglo de tamaño N están indexados a partir de cero hasta N-1.
- Para acceder a los elementos individuales de un arreglo, se escribe el nombre del arreglo seguido del subíndice del elemento entre corchetes.
- Ejemplo:

```
int a[3];
a[0] = 5;
a[1] = 8;
a[2] = a[0] * a[1];
```

# ¡Precaución!

El compilador de C/C++ **no verifica** que los subíndices de los elementos de un arreglo estén dentro de los límites correspondientes.

El programador debe tener cuidado de no sobrepasar estos límites. De lo contrario, es posible que sobreescriba la memoria asignada a otras variables.

Ejemplo:

```
int main() {
        int a[10], x[10];
        a[0] = 1;
        x[10] = 2;//ERROR
        cout << a[0] << endl;
        return 0;
}
```

# Arreglo Unidimensional

## Inicialización

- Con la declaración

  int num[3]={3,6,9};
- Después de la declaración

  int num[3];

  num[3]={3,6,9};
- Elemento por elemento (previa declaración)

  int num[3];

  num[0]=3;

  num[1]=6;

  num[2]=9;

# Arreglo Unidimensional

## Lectura

- Dependiendo el tipo de dato se utiliza la función de entrada correspondiente especificando la posición donde se recibirá el nuevo valor dentro del arreglo

  Sea int num[3];

  cin>>num[0];

## Escritura

- La muestra se debe realizar elemento por elemento. Es decir, se debe indicar la posición que ocupa el elemento a mostrar.

  cout<<num[0];

# Arreglo Unidimensional

## Asignación

- El valor asignado debe pertenecer al tipo de dato del arreglo.
- Puede realizarse
  - A una posición determinada

    num[0]=6;
  - A todo el arreglo

    vec=num; //vec debe ser del mismo tipo y tamaño que num

# Arreglo Unidimensional

## Recorrido

- Para recorrer todo el arreglo se utiliza una estructura repetitiva variando el índice desde 0 a N-1, siendo N el tamaño del arreglo.

# Arreglo Unidimensional

RECORRIDO CON WHILE

```
int main(){
int i=0,num[3]={3,6,9};
while(i<3) {
    cout<<num[i];
    i++;
}
```

RECORRIDO CON FOR

```
int main(){
int i,num[3]={3,6,9};
for(i=0;i<3;i++)
    cout<<num[i];
```

# Práctica #2

- Realizar un programa que permita cargar los N primeros elementos impares. Mostrarlos en orden inverso al generado.

# Arreglo Bidimensional

- A diferencia del arreglo unidimensional, requiere de dos subíndices, para denotar la ubicación de un elemento dentro de la matriz. El primer subíndice indicará la fila, y el segundo representará a la columna.
- La declaración de una matriz en C++ es:

```
tipoDato nombreArreglo[cantidadFilas][cantidadColumnas];
```

# Arreglo Bidimensional

int x[M][N];

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| x[0] | x[0][0] | x[0][1] | | | | | x[0][N-1] |
| x[1] | x[1][0] | x[1][1] | | | | | x[1][N-1] |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| x[M-1] | x[M-1][0] | x[M-1][1] | | | | | x[M-1][N-1] |

# Arreglo Bidimensional

- **Ejemplo:**
- Se desea implementar un arreglo de 10 x 5 donde el elemento (i,j) sea igual a (i + j).

- **Solución:**

```
int i, j;
int x[10][5];
for (i = 0; i < 10; i++)
        for (j = 0; j < 5; j++)
                x[i][j] = i + j;
```

# Práctica #3

- Escribir un programa que lea una matriz de enteros de 4 filas y 4 columnas y a continuación intercambie la fila i con la fila j, siendo i y j dos valores introducidos por teclado.