

# FUNCIONES

Catedrático: Ing. David Rajo



# Concepto de Función

- C++ se puede utilizar como lenguaje de programación estructurada, también conocida como programación modular. Por esta razón para escribir un programa se divide este en varios módulos, en lugar de uno solo largo. El programa se divide en mucho módulos (rutinas pequeñas denominadas funciones). Que producen muchos beneficios: aislar mejor los problemas, escribir programas correctos mas rápido y producir programas que son mas fáciles de mantener.
- Un programa en C++ se compone de varias funciones, cada una de las cuales realiza una tarea principal

# Estructura de una función

Tipo\_de\_retorno nombreFuncion (listaDeParametros)

{

Cuerpo de la funcion

Return expresion

}

**Tipo\_de\_retorno:** tipo de valor devuelto por la función o la palabra reservada void si la función no devuelve ningún valor

**nombreFuncion:** identificador o nombre de la función

**listaDeParametros:** lista de declaraciones de los parámetros de la función separados por comas

**Expresion:** valor que devuelve la función



# Estructura de una Función

- Cada función realiza determinada tarea, pues una función, es sencillamente, un conjunto de sentencias que se pueden llamar desde cualquier parte del programa. Las funciones permiten al programador un grado de abstracción en la resolución de un problema.
- Las funciones no se pueden anidar. Esto significa que una función no se puede declarar dentro de otra función. En C++ todas las funciones son externas o globales, es decir pueden ser llamadas desde cualquier punto del programa
- Cuando se ejecuta return se retorna al punto en que fue llamado por el programa o función principal

# Definición de una función: suma de 2 números

```
float suma (float num1, float num2)
{
    float resp;
    resp = num1 + num2;
    return resp;
}
```

Tipo de Resultado

Lista de parámetros

Cabecera de la función

Valor devuelto

# Tipo de dato de retorno

- El tipo de dato debe ser uno de los tipos simples de C++
- Si la función no devuelve un resultado, se puede utilizar el tipo de datos void, que se considera un tipo de datos especial
- Muchas funciones no devuelven resultados la razón es que se utilizan como subrutinas para realizar una tarea concreta. A veces estas se denominan procedimientos
- Si el tipo de retorno es void, la sentencia return se puede escribir como

**return;**

sin ninguna expresión de retorno, o bien se puede omitir la sentencia return

- Al no declarar tipo devuelto el compilador supone que el tipo de datos devuelto es int

# Declaración de funciones

- C++ requiere que una función se declare o defina antes de que pueda ser llamada,
- La declaración para una función se conoce como prototipo de función. Los prototipos de una función contienen la misma cabecera de la función, con la diferencia que los prototipos terminan con un punto y coma.

Tipo\_dato\_a\_devolver nombre\_De\_funcion (lista de tipos de datos para argumentos)

```
int fmax(int, int);
```

```
double intercambio(int, char, char, double);
```

```
void desplegar(double, double);
```

- Nota importante: C++ requiere que este declarada una función si se llama a una función antes que se defina





# Prototipos de las funciones

- Un prototipo declara una función y proporciona información suficiente al compilador para verificar que la función esta siendo llamada correctamente, con respecto al numero y tipo de parámetros y el tipo devuelto por la función. Es obligatorio poner un punto y coma al final del prototipo de la función con el objeto de convertirlo en una sentencia
- Los prototipos se sitúan normalmente al principio de un programa, antes de la definición de la primera función `main()`.
- Una declaración de la función contiene solo la cabecera de la función y una vez declarada la función la definición completa de la función debe existir en algún lugar del programa

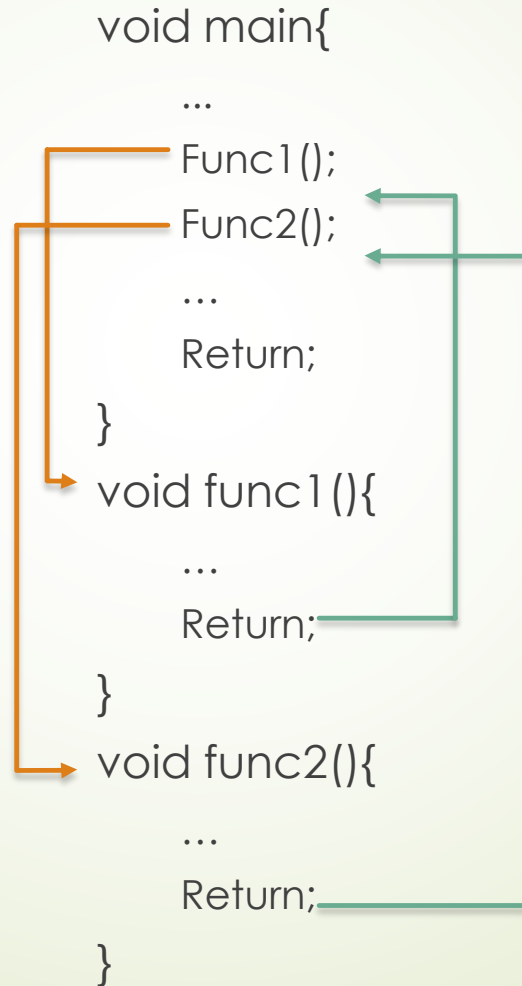




# Llamada a una función

- Las funciones para ser ejecutadas han de ser llamadas o invocadas. Cualquier expresión puede contener una llamada a una función que redirigirá el control de programa a la función nombrada. Normalmente la llamada a una función se hará desde la función principal `main()`, aunque naturalmente también se podrá hacer desde otra función.
- La función que llama a otra función se denomina función llamadora o controladora y la función controlada se denomina función llamada o invocada

# Traza de llamadas de funciones





# Ejemplo

Elaborar un programa en C++ con las siguientes características

- Debe solicitar 2 números introducidos por teclado
- Implementar un función para sumar 2 números con retorno de un numero de tipo entero
- Implementar función para encontrar el mayor de 2 números con retorno de un numero de tipo punto flotante
- Implementar función para encontrar el factorial de cada numero sin retorno de valores (subrutina o procedimiento)
- Imprimir los resultados de cada función



# Ejercicio

- Elabore un programa que simule la función `pow` de la librería `cmath` implementando estructuras iterativas.
- Crear un programa tipo calculadora que realice las operaciones de manera modular (a través de funciones) deberá contener las operaciones básicas (suma, resta, multiplicación y división) además incluir las operaciones de elevación, raíz y factorial de los números.



# PARAMETROS DE UNA FUNCIÓN

- C++ proporciona 2 métodos para pasar variables (parámetros) entre funciones.
  - Parámetros por valor
  - Parámetros por referencia,

El método por defecto de pasar parámetros es por valor.



# Paso de parámetros por valor

- Paso por valor (también llamado paso por copia) significa que cuando C++ compila la función y el código que llama la función, la función recibe una copia de los valores de los parámetros. Si se cambia el valor de un parámetro local, el cambio solo afecta a la función y no tiene efecto fuera de la función.



# Ejemplo

```
#include <iostream>
using namespace std;
int funcion(int n, int m);
int main() {
    int a, b;
    a = 10;
    b = 20;
    cout << "a,b ->" << a << ", " << b << endl;
    cout << "funcion(a,b) ->" << funcion(a, b) << endl;
    cout << "a,b ->" << a << ", " << b << endl;
    cout << "funcion(10,20) ->" << funcion(10, 20) << endl;
    return 0;
}
int funcion(int n, int m){
    n = n + 2;
    m = m - 5;
    return n+m;
}
```

# Explicación

- Empezamos haciendo  $a = 10$  y  $b = 20$ , después llamamos a la función "funcion" con los objetos  $a$  y  $b$  como parámetros. Dentro de "funcion" esos parámetros se llaman  $n$  y  $m$ , y sus valores son modificados. Sin embargo al retornar a main,  $a$  y  $b$  conservan sus valores originales. ¿Por qué?
- La respuesta es que lo que pasamos no son los objetos  $a$  y  $b$ , sino que copiamos sus valores a los objetos  $n$  y  $m$ .
- Piensa, por ejemplo, en lo que pasa cuando llamamos a la función con parámetros constantes, es lo que pasa en la segunda llamada a "funcion". Los valores de los parámetros no pueden cambiar al retornar de "funcion", ya que esos valores son constantes.
- Si los parámetros por valor no funcionasen así, no sería posible llamar a una función con valores constantes o literales.




# Paso de parámetros por referencia

- Cuando una función debe modificar el valor del parámetro pasado y devolver este parámetro modificado a la función llamadora, se ha de utilizar el método de paso de parámetros por referencia o dirección.
- En este método el compilador pasa la dirección de memoria del valor del parámetro a la función. Es decir que cuando se modifique el valor del parámetro (la variable local), este valor queda almacenado en la misma dirección de memoria.
- Para declarar una variable parámetro como paso por referencia, el símbolo & debe preceder al nombre de la variable

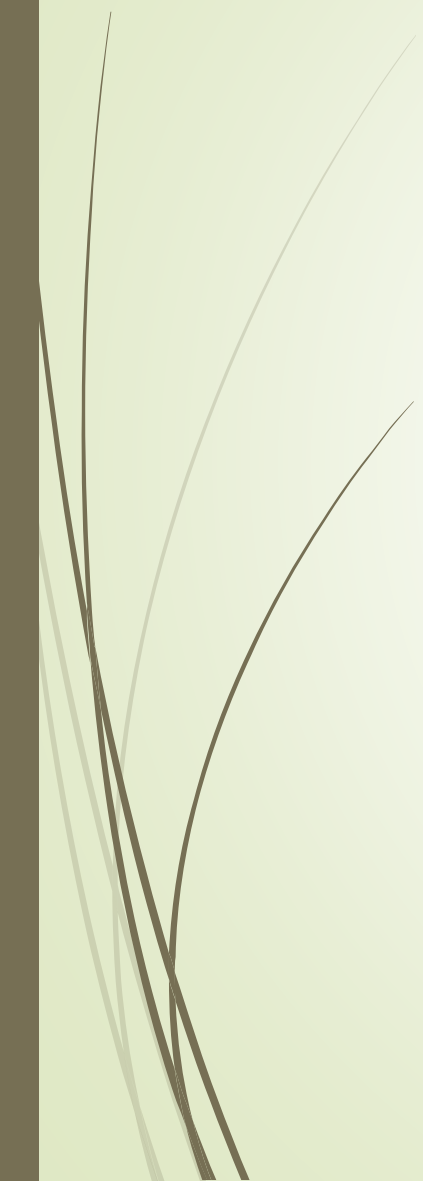
# Ejemplo

```
#include <iostream>
using namespace std;
void intercambio(int&, int&);
int main() {
    int a, b;
    a = 10;
    b = 20;
    cout << "a = " << a << "y b = " << b << endl;
    intercambio(a,b);
    cout << "a = " << a << "y b = " << b << endl;
    return 0;
}

void intercambio(int& primero, int& segundo){
    int aux;
    aux = primero;
    primero = segundo;
    segundo = aux;
}
```



# Diferencia entre los parámetros por valor y por referencia





# Argumentos por omisión

- Cuando una función tiene un numero de parámetros, normalmente el mismo numero de argumentos deben indicarse cuando se llama a la función. En C++ sin embargo, es posible omitir algún argumento.
- Una característica de C++ es que se pueden establecer parámetros por omisión o ausencia.
- Se pueden asignar valores por defecto a los parámetros de una función
- El valor por defecto deber ser una valor constante



# Ejemplo

```
void matriz (int fila, int col, char c = '*');
```

Se puede llamar de 2 formas equivalentes:

```
Matriz (4,5);
```

Ø bien

```
Matriz (4,5,'*')
```

Ø también

```
Matriz (4,5,'#');
```



# Reglas de construcción de argumentos por defecto

- Solo paso por valor
- Solo valores literales o definiciones const
- No pueden ser variables
- Los argumentos por defecto se colocan al final del prototipo, porque todos los posteriores deberán incluir valores por defecto



# Ejercicio

- Implementar el ejercicio de la impresión de una matriz implementando parámetros por defecto.



# Próxima clase

- Funciones externas
  - Funciones en línea
  - Ámbito
  - Recursividad
  - Archivos de cabecera
- 