



# Unidad III Creación de programas con funciones

M.C. Juan Carlos Olivares Rojas



# Agenda

- 3.1 Estructura de una función
- 3.2 Parámetros de una función
- 3.3 Variables locales y alcance
- 3.4 Sobrecarga de funciones
- 3.5 Empleo de referencias
- 3.6 Paso de parámetros
- 3.7 Empleo de constantes y macros

# 3.1 Estructura de una función

- Una función permite que los programas sean modulares haciendo con ello que sean más fáciles de leer, reutilizables y en general más rápidos al no tener que repetir tanto código.
- En C todas las funciones devuelven un valor, a las funciones que no devuelven un valor en otros lenguajes se les llama procedimientos. En C un procedimiento es una función que devuelve un valor void

# Estructura de una función

- No se pueden declarar variables del tipo void
- En general se usa el término de prototipo de función para indicar los parámetros que recibe una función sin llegar a implementarla, es útil para el desarrollo de bibliotecas. Los archivos .h

# 3.1 Estructura de una función

```
Tipo_de    dato_a_devolver    nombre_funcion  
(parametros)  
{  
    Conjunto de instrucciones que realizan una  
    determinada actividad  
    return valor;  
}
```

## 3.2 Parámetros de una función

- Los parámetros de una función se pasan al igual que en otros lenguajes listando el nombre de variables y su tipo

```
int suma(int n1, int n2)
{
    int resultado = n1 + n2;
    return resultado;
}
```

## 3.3 Variables locales y alcance

- Las variables locales son aquellas se declaran dentro de un bloque de instrucciones: funciones, ciclos.
- Las variables locales solo son visibles en donde fueron declaradas. Incluso pueden tener el mismo nombre de otras variables iguales sólo se toma en cuenta la que sea más local

## 3.4 Sobrecarga de funciones

- La sobrecarga de funciones permite definir con el mismo nombre diferentes funciones que realicen la misma actividad pero que trabajen con diferentes tipos de datos

```
float suma (float n1, float n2)
{
    return n1 + n2;
}
```

# Sobrecarga de funciones

- La característica que deben tener esas funciones radica en el hecho de que deben devolver un tipo de datos diferentes sin importar el tipo de datos de sus parámetros
- También se pueden sobrecargar otros elementos como operadores

## 3.5 Empleo de referencias

- Una referencia es un alias hacia otro tipo de variables.
- En general existen dos tipos de variables: las de valor y los punteros o referencias
- Un puntero hace referencia a una dirección de memoria

# Punteros

- Los punteros se declaran anteponiendo un \* a los tipos de datos. Por ejemplo:
- `int *a;` //Es un puntero a un tipo de datos entero
- El operador de indirección \* si se utiliza hace referencia al contenido de la variable;

# Referencias

- `a = 0xFFFF;`
- `a = b;`
  
- `**a = 5`
- `b = 5;`
  
- En C todos los arreglos son funciones. Las variables también pueden tener punteros

# Referencias

- En general los punteros se utilizan para el paso de parámetros por referencia.

```
a=5; b=3;
void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}
printf("a=%d b=%d");
```

# Referencias

- El operador de dirección & devuelve la dirección de la localidad de la memoria en donde se encuentra almacenada una variable.
- Ejemplo: `int a = &b;`
- `int &a = b;` es un alias de b, puedo operar con a y se modifica b automáticamente.

# Referencias

- `a=7; b=2;`
- `void swap(int *a, int *b)`
- `{`
- `int temp = **a;`
- `** a = **b;`
- `**b = temp;`
- `}`
- `printf("a=%d b=%", a, b);`
- `a=5; b=3;`
- `void swap(int &a, int &b)`
- `{`
- `int temp = a;`
- `a = b;`
- `b = temp;`
- `}`
- `printf("a=%d b=%d", a, b);`

## 3.6 Paso de parámetros

- Existen dos tipos de pasos de parámetros: por valor y por referencia.
- En el paso de valor por valor los parámetros no se modifican mientras que por referencia los valores si cambian.
- Es importante saber como se va a realizar el paso de parámetros para no tener problemas de inconsistencias de variables

## 3.7 Empleo de constantes y macros

- Las constantes se definen de dos tipos simbólicas y por valor
- `#define PI 3.14159`
- `const float PI 3.14159`
- No es válido `PI = 3.1;`

# Macros

- Las macros son código que se expande en el momento de realizar código. Generalmente en C van en mayúscula.
- Ejemplos de macros famosas EOF, EXIT\_SUCCESS, EXIT\_FAILURE, isalpha(), NULL, STDERR, STDIN, STDOUT

# ¿Preguntas?

