

# PROGRAMACIÓN ESTRUCTURADA

UNIDAD 6

# Subrutinas

UN CAMINO PARA DIVIDIR UN GRAN PROGRAMA EN PARTES MAS  
PEQUEÑAS

# ¿Qué son?

---

- Un grupo de instrucciones, variables, constantes y estructuras, que están diseñados con un propósito particular y tiene su nombre propio.
- Es un bloque de código que realiza alguna operación.
- Puede definir opcionalmente parámetros de entrada que permiten a los llamadores pasar argumentos a la función

# Concepto

---

- **PROCEDIMIENTOS**

- Es una subrutina que realiza una tarea específica y que no regresa valores a la subrutina que lo invocó.
- Se indica con “VOID”

- **FUNCIONES**

- Subrutina que devuelve un único resultado del procesamiento realizado.

# Funciones y procedimientos

# ¿Qué es una función?

---

- Los módulos en C se llaman funciones.
- Son bloques de código que procesan datos y devuelven un resultado
- Hemos estado utilizando funciones de la biblioteca estandar “stdio.h” como por ejemplo printf y scanf.
- Sintaxis:

```
TipoValorRetornado NombreDeLaFuncion(parámetros)
{ declaraciones locales
  Instrucciones de la función
}
```

```
int Cuadrado ( int nro)
{ int resultado;
  resultado = nro * nro;
  return (resultado) ;
}
```

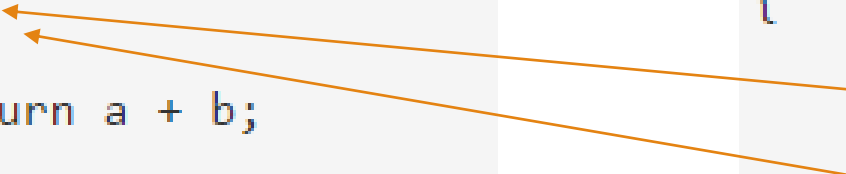
# ¿Qué es una función?

---

- La función puede invocarse, o llamarse, desde cualquier número de partes del programa.
- Una declaración de función mínima consiste en el tipo de valor devuelto, el nombre de función y la lista de parámetros (que puede estar vacía)
- Se utiliza indicando el tipo de datos que devuelve “int sum()”

```
int sum(int a, int b)
{
    return a + b;
}
```

```
int main()
{
    int i = sum(10, 32);
    int j = sum(i, 66);
}
```

Two orange arrows originate from the function calls in the main function. The first arrow starts at 'sum(10, 32)' in the line 'int i = sum(10, 32);' and points to the opening curly brace of the 'sum' function definition. The second arrow starts at 'sum(i, 66)' in the line 'int j = sum(i, 66);' and also points to the opening curly brace of the 'sum' function definition.

# ¿Qué es una función?

---

```
1  #include <iostream>
2
3  int cuadrado (int a); //prototipo de la función
4
5  int main(int argc, char** argv) {
6
7
8      int valorCuadrado = cuadrado(10); //invocación de la función
9      printf("%d", valorCuadrado);
10
11
12      return 0;
13  }
14
15  int cuadrado(int a){ //definición de la función
16
17      return a*a; // valor de retorno
18
19  }
```



# ¿Qué es un procedimiento?

---

- Similar a una función pero no devuelve ningún valor.
- Utiliza el tipo VOID.

# Parámetros

---

- Un parámetro es una variable que puede pasar su valor a una subrutina.
- Cada subrutina tiene **una firma** que indica si espera o no algún parámetro
- Los **valores que se pasan a una subrutina son los argumentos**, cuyos tipos deben ser compatibles con los tipos de parámetro de la definición de la misma

# Parámetros

---

- **POR REFERENCIA**
  - Se pasa la referencia de la variable que queremos utilizar como argumento.
  - Este método se utiliza cuando una función debe modificar el valor de un parámetro recibido y devolverlo modificado a la función llamadora
- **POR VALOR**
  - Se realiza una copia del valor de la variable y se pasa como argumento.
  - podrán ser modificados, pero cuando la función devuelve el control a la función llamadora, recuperan los valores originales que tenían antes de ser modificados por la función

# Parámetros

---

- El lenguaje C sólo posee pasaje de parámetros por copia o por valor. Por lo tanto no es posible cambiar el valor de un parámetro desde una función.
- ¿Cómo hacer entonces para permitir que un función devuelva algo a través de su lista de parámetros?
  - UTILIZANDO PUNTEROS

# Retorno de una función

---

- Hay tres formas de regresar al punto desde el cual se hizo la invocación de la función:
  - Si la función no retorna nada, el control sólo se devuelve cuando se llega a la llave derecha que termina la función.
  - Cuando se ejecuta la instrucción **return;**
  - Si la función devuelve un resultado, la instrucción **return expresion;** devuelve el valor de **expresion** al punto de llamada.

- Tipo VOID

- Es un tipo que representa que no hay ningún valor.

```
1  #include <iostream>
2
3  int saludar (); //prototipo de la función
4
5  int main(int argc, char** argv) {
6
7      saludar();
8
9      return 0;
10 }
11
12
13
14 void saludar(){
15     printf("hola mundo!")
16 }
17
18
```

# Ámbito de una variable

---

- Si una función reconoce a una variable se dice que esa variable es visible a esa función.
- El ámbito puede ser
  - Programa
    - Son globales, se definen antes de la función main y fuera de cualquier función.
  - Subrutina (función o procedimiento)
    - Son visibles únicamente para el bloque que la declara.
    - Los parámetros, son variables definidas por una subrutina que son accesibles desde el bloque y asignadas desde la función que lo invoca.

o

# Funciones recursivas

---

- Una función recursiva es aquella que se llama a si misma
  - Directa es cuando lo hace desde el propio cuerpo de la función,
  - Indirecta cuando implica a más de una función.
- Toda función recursiva debe tener una condición de terminación, porque sino seria su continuidad indefinidamente.
- Se puede utilizar una función recursiva para realizar repeticiones similares a los ciclos de repetición.
- Una función típica de recursividad es el calculo del factorial.

# Cálculo de factorial de un número

---

El factorial (!) de un número es el ese número multiplicado por ese número menos 1:

- $\text{Factorial}(n) = n * \text{factorial}(n-1)$

$$4! = 4 * 3 * 2 * 1$$

$$3! = 3 * 2 * 1$$

Entonces...

- $4! = 4 * 3!$
- $3! = 3 * 2!$
- $2! = 2 * 1!$
- $1! = 1 * 0!$
- $0! = 0 * -1!$

→ Hasta acá, todos los números dependen del factorial anterior (condición recursiva)

→ esto se vuelve infinito si no defino una condición de parada, puedo lograrlo indicando que el factorial de 0 es 1 ( $0!=1$ )



# Factorial de un número

```
1  #include <iostream>
2
3  int factorial(int n); //declaro la función recursiva
4  int factorial(int n){
5
6      if (n==0) //condición de corte
7          return 1;
8      else
9          return n * factorial(n-1);
10 }
11 int main(int argc, char** argv) {
12     int n= 4;
13     //f= 4 * 3 * 2 * 1 => f=24
14     int f = factorial(n); //invoco la función
15     printf ("El factorial de %d es %d",n,f);
16     return 0;
17 }
18
```

