Tema 3. FUNCIONES

TEMA 3. FUNCIONES

1. **INTRODUCCIÓN**
2. **DEFINICIÓN DE UNA FUNCIÓN**
3. **DECLARACIÓN DE UNA FUNCIÓN**
4. **PASO DE PARÁMETROS**
   - 4.1. PASO POR VALOR
   - 4.2. PASO POR DIRECCIÓN
5. **VARIABLES LOCALES Y GLOBALES**
6. **RECURSIVIDAD**

1. Introducción

- ✓ Programación modular: división de problemas genéricos en tareas más sencillas que finalmente podamos resolver mediante un algoritmo simple.
  - × Combinando estas tareas en una secuencia concreta obtendremos la solución al problema original.
- ✓ Subalgoritmo: un algoritmo particular que es utilizable por otros algoritmos.
- ✓ Función: es un subalgoritmo expresado en un lenguaje de programación, un módulo o segmento de código que realiza una tarea determinada.
  - × Trabajan en base a unos datos que se le suministran, denominados *argumentos* o *parámetros.*
  - × Puede devolver o no un resultado.
- ✓ Todo programa en C consta de una o más funciones. Una de ellas debe llamarse *main*.
  - × La ejecución del programa siempre comenzará por el contenido de *main*.

1. Introducción

- ✓ Ventajas del uso de subalgoritmos.
  - × Disminuye el tamaño del programa.
    - Evita duplicación de sentencias para procesos que realicen las operaciones con diferentes datos.
  - × Permite mayor claridad en los programas fuentes (legibilidad).
    - Ejemplo: si queremos ordenar una lista de números, podemos hacer una llamada a un subalgoritmo.
  - × Facilita las modificaciones posteriores de los programas.
  - × Facilita la depuración de errores.
    - Todo subalgoritmo puede ser probado independientemente del conjunto general del programa.
  - × Se pueden confeccionar bibliotecas de funciones o módulos de uso general.
    - Ayuda a la productividad y eficiencia del programador (puede utilizar código ya hecho).

2. Definición de una función

- ✓ **Sintaxis de la definición de una función:**

```
tipo_del_parámetro_de_salida  NombreFunción ( declaración_lista_parámetros )
{
      Declaración de variables
      Sentencias
      Devolución de resultados  [return(valor_de_vuelta); o simplemente return;]
}
```

  - × Se debe especificar un tipo de datos para el resultado de vuelta.
    - Se usa la palabra void para indicar que no se devuelve ningún dato.
    - Si no se especifica nada, se asume int.
  - × La lista de parámetros es una lista de identificadores de variables precedidos por la declaración de su tipo y separados por comas.
    - Los argumentos que utilicemos al efectuar la llamada han de coincidir en tipo y orden con los parámetros declarados en esta lista.
  - × La función debe devolver un valor del mismo tipo que se ha declarado en la misma.
    - Para ello se utiliza la sentencia *return.*

2. Definición de una función

EJEMPLO 1: Función que calcula el cuadrado de un número entero

```
#include <stdio.h>

int Cuadrado(int x)
{
     return(x*x);
}

main()
{
     int i, c;

     printf("Introduce el numero: ");
     scanf("%d", &i);
     c = Cuadrado(i);
     printf("El cuadrado es: %d\n", c);
}
```

Definición de la función **Cuadrado**

Llamada a la función **Cuadrado**

2. Definición de una función

EJEMPLO 2: Función que calcula el cubo de un número entero

```
#include <stdio.h>

int Cubo(int x)
{
     return(x*x*x);
}

main()
{
     int i, c;

     printf("Introduce el numero: ");
     scanf("%d", &i);
     c = Cubo(i);
     printf("El cubo es: %d\n", c);
}
```

Definición de la función **Cubo**

Llamada a la función **Cubo**

2. Definición de una función

EJEMPLO 3: Función que calcula una potencia general x^n

```
#include <stdio.h>

float Potencia(float base, int exp)
{
   int i;
   float resultado;

   resultado = 1;
   for (i=1; i <= exp; i++)
        resultado = resultado*base;
   return(resultado);
}
```

```
main()
{
   int n;
   float x, r;

   printf("Introduce la base: ");
   scanf("%f", &x);
   printf("Introduce el exponente: ");
   scanf("%d", &n);

   r = Potencia(x,n);

   printf("El resultado es: %f\n", r);
}
```


2. Definición de una función

EJEMPLO 4: Función que calcula el factorial de un número

```
#include <stdio.h>

long Factorial(int n)
{
   int i;
   long fact;

   fact = 1;
   for (i=1; i <= n; i++)
        fact = fact*i;
   return(fact);
}
```

```
main()
{
    int n;
    long r;

    printf("Introduce el numero: ");
    scanf("%d", &n);

    r = Factorial(n);

    printf("El resultado es: %ld\n", r);
}
```

2. Definición de una función

EJEMPLO 5: Secuencia de llamadas a funciones

```
#include <stdio.h>

void MiFuncion(void)
{
   printf("Esto es MiFuncion\n");
   return;
}

void MiOtraFuncion(void)
{
   printf("Esto es MiOtraFuncion\n");
   return;
}
```

```
main()
{
        printf("Esto es main\n");
        MiFuncion();
        MiOtraFuncion();
        printf("Fin de main\n");
}
```

Salida por pantalla:

```
Esto es main
Esto es MiFuncion
Esto es MiOtraFuncion
Fin de main
```


2. Definición de una función

EJEMPLO 6: Secuencia de llamadas a funciones (2)

```
#include <stdio.h>

void MiFuncion(void)
{
   printf("Esto es MiFuncion\n");
   return;
}

void MiOtraFuncion(void)
{
   printf("Esto es MiOtraFuncion\n");
   MiFuncion();
   return;
}
```

```
main()
{
        printf("Esto es main\n");
        MiFuncion();
        MiOtraFuncion();
        printf("Fin de main\n");
}
```

Salida por pantalla:

```
Esto es main
Esto es MiFuncion
Esto es MiOtraFuncion
Esto es MiFuncion
Fin de main
```

3. Declaración de una función

- ✓ En C no necesita declararse una función si la primera llamada a la misma se produce después de la definición.
  - × Puede ser necesario *declarar* una función antes de ser definida.
- ✓ Sintaxis de la declaración de una función:

  tipo_del_parámetro_de_salida NombreFunción (declaración_lista_parámetros);

  Ejemplo: float Potencia(float, int);
  - × En la lista de parámetros no es necesario especificar los identificadores.
- ✓ La declaración de una función se llama también *prototipo.*
  - × Un prototipo consiste en la *declaración* de todas las características de una función excepto el código de la misma.
  - × Los prototipos se pueden escribir en los llamados *ficheros de cabecera* (.h) y luego incluirlos en nuestro programa fuente con #include.

3. Declaración de una función

EJEMPLO 1: Función que calcula una potencia general x^n

```
#include <stdio.h>

float Potencia(float base, int exp);

main()
{
   int n;
   float x, r;

   printf("Introduce la base: ");
   scanf("%f", &x);
   printf("Introduce el exponente: ");
   scanf("%d", &n);

   r = Potencia(x,n);

   printf("El resultado es: %f\n", r);
}
```

```
float Potencia(float base, int exp)
{
   int i;
   float resultado;

   resultado = 1;
   for (i=1; i <= exp; i++)
        resultado = resultado*base;
   return(resultado);
}
```

4. Paso de parámetros

- ✓ **Paso por valor**
  - × Se pasa una copia temporal de la variable
  - × Se puede modificar el valor de la copia, pero no el valor de la variable original
  - × En todos los ejemplos anteriores el paso de parámetros es por valor
- ✓ **Paso por dirección**
  - × Se pasa la dirección en memoria de la variable
  - × Se puede modificar el valor de la variable original pasada
  - × Requiere el uso de *punteros*

4. Paso de parámetros

EJEMPLO 1:
Ejemplo de paso de parámetros por valor

```
#include <stdio.h>

int Funcion(int, int);

void main(void)
{
    int x=0, y=1, z;

    z = Funcion(x, y);
    printf("x = %d, y = %d, z = %d\n", x, y, z);

    system("PAUSE");
}

int Funcion(int var1, int var2)
{
    var1 = 10;
    var2 = -5;
    printf("var1 = %d, var2 = %d\n", var1, var2);
    return(var1-var2);
}
```

Salida por pantalla:

```
var1 = 10, var2 = -5
x = 0, y = 1, z = 15
```

5. Variables locales y globales

- ✓ Diremos que una variable es *local* cuando está declarada dentro de una función
  - × El espacio en memoria de la variable se creará en el momento en que empiece la ejecución de la función y se destruirá al transferir el control al punto en que se realizó la llamada
  - × No se podrá acceder a esta variable desde otro punto del programa que no sea dentro de la función en la que ha sido declarada
- ✓ Diremos que una variable es *global* si es accesible desde cualquier función
  - × Estas variables se declaran fuera del cuerpo de cualquier función
  - × Se suelen declarar en la zona previa a la definición de funciones, tras los #include y #define.
  - × Si existe una variable local con el mismo nombre que otra global, dentro de la función prevalece la variable local

5. Variables locales y globales

Ejemplo 1: Variables locales y globales.

```
#include <stdio.h>

int i; // variable global

void Funcion1(void)
{
    int j; // variable local

    j = 1;
    i = 2;
    printf("i = %d\n", i);
    printf("j = %d\n", j);
}

void Funcion2(void)
{
    int k;  // variable local

    k = 3;
    i++;
    printf("k = %d\n", k);
    printf("i = %d\n", i);
}

void Funcion3(void)
{
    int i;  // variable local

    i = -5;  // se refiere a la local
    printf("i = %d\n", i);
}

void main(void)
{
    int p;  // variable local

    p = 0;
    Funcion1();
    Funcion2();
    Funcion3();
    printf("p = %d\n", p);
    printf("i = %d\n", i);
}
```

Salida:

```
1
2
3
3
-5
0
3
```

6. Recursividad

- ✓ Cuando una función se llama a sí misma decimos que esa función es *recursiva*
- ✓ Ejemplo: el factorial de un número natural *n* se puede definir de forma recursiva

$$n! = \begin{cases} 1 & \text{si } n = 1 \\ n*(n-1)! & \text{si } n > 1 \end{cases}$$

- ✓ Función en C recursiva:

```
long Factorial(int n)
{
    if (n == 1)
        return(1);
    else
        return( n*Factorial(n-1));
}
```