

# Arquitectura de sistemas

**Abelardo Pardo**

University of Sydney  
School of Electrical and Information Engineering  
NSW, 2006, Australia  
*Autor principal del curso de 2009 a 2012*

**Iria Estévez Ayres**

**Damaris Fuentes Lorenzo**

**Pablo Basanta Val**

**Pedro J. Muñoz Merino**

**Hugo A. Parada**

**Derick Leony**

Universidad Carlos III de Madrid  
Departamento de Ingeniería Telemática  
Avenida Universidad 30, E28911 Leganés (Madrid), España

---

© Universidad Carlos III de Madrid | Licencia Creative Commons



## Capítulo 4. Las funciones en C

### Tabla de contenidos

- [4.1. Definición de funciones](#)
- [4.2. Paso de parámetros a una función](#)
  - [4.2.1. Tablas como parámetros a una función](#)
- [4.3. Prototipos de funciones](#)
- [4.4. Funciones estáticas](#)
- [4.5. Autoevaluación automática](#)
- [4.6. Bibliografía de apoyo](#)
- [4.7. Ejercicios](#)

El código de un programa escrito en C se divide en funciones. Aunque similares a los “métodos” de Java, las funciones no están asignadas ni a una clase ni a un objeto. Una función en C se distingue **sólo** por su nombre. Dos funciones con igual nombre y con diferente número y tipo de parámetros se considera una definición múltiple, y por tanto un error.

Las funciones suelen encapsular una operación más o menos compleja de la que se deriva un resultado. Para ejecutar esta operación, las funciones pueden precisar la invocación de otras funciones (o incluso de ellas mismas como es el caso de las funciones recursivas).

Las funciones en un programa son entidades que dado un conjunto de datos (los parámetros), se les encarga realizar una tarea muy concreta y se espera hasta obtener el resultado. Lo idóneo es dividir tareas complejas en porciones más simples que se implementan como funciones. La división y agrupación de tareas en funciones es uno de los aspectos más importantes en el diseño de un programa.

### 4.1. Definición de funciones

---

Las funciones en C tienen el siguiente formato:

```
tipo_del_resultado NOMBRE(tipo_param1 param1, tipo_param2 param2, ... )
{
    /* Cuerpo de la función */
}
```

Cuando se invoca una función se asignan valores a sus parámetros y comienza a ejecutar el cuerpo hasta que se llega al final o se encuentra la instrucción `return`. Si la función devuelve un resultado, esta instrucción debe ir seguida del dato a devolver. Por ejemplo:

```
1 int search(int table[], int size)
2 {
3     int i, j;
4     if (size == 0)
5     {
6         return 0;
7     }
8     j = 0;
9     for (i = 0; i < size; i++)
```

```

10 |     {
11 |         j += table[i];
12 |     }
13 |     return j;
14 | }

```

La ejecución de la función comienza en la línea 4. Si el parámetro `size` tiene valor cero, la función termina y devuelve el valor cero. Si no, se ejecuta el bucle y se devuelve el valor de la variable `j`. El tipo de la expresión que acompaña a la instrucción `return` debe coincidir con el tipo del resultado declarado en la línea 1.

La llamada a una función se codifica con su nombre seguido de los valores de los parámetros separados por comas y rodeados por paréntesis. Si la función devuelve un resultado, la llamada se reemplaza por su resultado en la expresión en la que se incluye. Por ejemplo:

```

1 | int addition(int a, int b)
2 | {
3 |     return (a + b);
4 | }
5 | int main()
6 | {
7 |     int c;
8 |     c = c * addition(12, 32);
9 | }

```

## 4.2. Paso de parámetros a una función

---

Los parámetros son variables locales a los que se les asigna un valor antes de comenzar la ejecución del cuerpo de una función. Su ámbito de validez, por tanto, es el propio cuerpo de la función. El mecanismo de paso de parámetros a las funciones es fundamental para comprender el comportamiento de los programas en C.

Considera el siguiente programa:

```

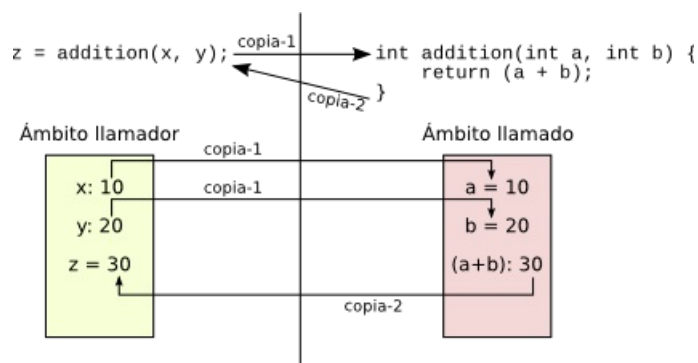
1 | int addition(int a, int b)
2 | {
3 |     return (a + b);
4 | }
5 | int main()
6 | {
7 |     int x = 10;
8 |     int y = 20;
9 |     int z;
10 |
11 |     z = addition(x, y);
12 | }

```

Los parámetros `a` y `b` declarados en la línea 1 son válidos únicamente en la expresión de la línea 3. Las variables `x`, `y` y `z`, por su lado, son válidas en el cuerpo de la función `main` (líneas 7 a 11).

El ámbito de las variables `x`, `y` y `z` (ámbito llamador), y el de las variables `a` y `b` (ámbito llamado) son totalmente diferentes. El ámbito llamador desaparece temporalmente cuando se invoca la función desde la línea 11. Durante esa ejecución, el ámbito llamado es el visible. Al terminar la función, el ámbito llamado desaparece y se recupera el ámbito llamador.

La comunicación entre estos dos ámbitos se realiza en la línea 11. Antes de comenzar la ejecución de la función, **los valores de las variables del ámbito llamador son copiadas sobre las variables del ámbito llamado**. Cuando termina la ejecución de la función, la expresión de la llamada en la línea 11 se reemplaza por el valor devuelto. En la siguiente figura se ilustra este procedimiento para el ejemplo anterior.



El paso de parámetros y la devolución de resultado en las funciones C se realiza **por valor**, es decir, **copiando** los valores entre los dos ámbitos .

Las variables locales de la función (incluidos los parámetros) desaparecen al término de la función, por lo que cualquier valor que se quiera guardar, debe ser devuelto como resultado (ver ejercicio [1](#)).

#### 4.2.1. Tablas como parámetros a una función

La copia de parámetros del ámbito llamador al llamado tiene una excepción. Cuando una función recibe como parámetro una tabla, en lugar de realizarse un duplicado se copia su dirección de memoria. Como consecuencia, si una función modifica una tabla que recibe como parámetro, estos cambios sí son visibles en el ámbito llamador. El siguiente ejemplo ilustra esta situación:

```

1  #define NUMBER 100
2
3  void fill(int table[NUMBER], int size)
4  {
5      int i;
6      for (i = 0; i < size; i++)
7      {
8          table[i] = 0;
9      }
10 }
11
12 int main()
13 {
14     int i, data[NUMBER];
15
16     for (i = 0; i < size; i++)
17     {
18         data[i] = 10;
19     }
20     fill(data, NUMBER);
21     /* Valores de data todos a cero */
22 }
  
```

Para la explicación de este comportamiento es preciso primero comprender el mecanismo de punteros en C.

## 4.3. Prototipos de funciones

Como el compilador trabaja sólo con la información contenida en un único fichero, a menudo es preciso "informar" al compilador de que en otro fichero existe una función. Esto se consigue insertando, en lugar de la definición de la función (que ya está presente en otro fichero), su prototipo. El prototipo de una función es una línea similar a la primera de su declaración: tipo del resultado, seguido del nombre de la función y de la lista de tipos de datos de los parámetros separados por comas y rodeados por paréntesis. Toda función que se invoca debe ir precedida o de su definición o de su prototipo. La definición y el prototipo de la función pueden estar presentes en el mismo fichero. El siguiente ejemplo ilustra esta situación:

```
1  /* Prototipo */
2  int addition(int, int);
3  /* Función principal */
4  int main()
5  {
6      int i, j;
7
8      i = 10;
9      j = 20;
10     /* Invocación de la función */
11     i += addition(i, j);
12 }
13 /* Definición de la función */
14 int addition(int a, int b)
15 {
16     return (a + b);
17 }
```

### Sugerencia

Copia y pega el ejemplo anterior en un fichero de texto en tu entorno de desarrollo y compila el programa con el comando **gcc -Wall -o programa fichero.c** reemplazando **fichero.c** por el nombre del fichero que hayas utilizado. Si el compilador no imprime ningún mensaje quiere decir que el programa es sintácticamente correcto. Prueba a quitar el prototipo de la línea 2 y compila de nuevo. Si mueves la definición de la función que comienza en la línea 14 al principio del fichero, ¿necesitas el prototipo? ¿qué pasa si aún así lo incluyes? ¿qué sucede si la función y el prototipo no coinciden en el tipo de los parámetros o el tipo del resultado?

## 4.4. Funciones estáticas

La definición de una función puede tener el prefijo "static". Cuando una función se declara como estática, tan sólo puede ser invocada desde el fichero en el que está definida. Este mecanismo, por tanto, puede interpretarse como una forma primitiva de restringir el acceso a una función, pero dista mucho del mecanismo de tres niveles (público, privado y protegido) presente en Java.

Cuando se desarrollan aplicaciones de gran tamaño, se suelen establecer políticas para el uso de prototipos. Por ejemplo, para poder invocar cualquier función en cualquier parte del código de un fichero, se suelen colocar al comienzo del mismo los prototipos de todas las funciones que contiene. El siguiente ejemplo muestra un ejemplo de esta política:

```
1  /* Funciones globales */
2  int function1(int p1, float p2, int table[], int size);
3  void function2();
4  struct data *function3(char *name, char* lastname, int status);
```

```
5  /* Funciones locales */
6  static void check(int table[], int size);
7  static struct data *duplicate(struct data *d1);
8
9  /* Definiciones */
10 .....
```

## 4.5. Autoevaluación automática

---

1. ¿Puede en un programa en C existir funciones con igual nombre pero con diferentes parámetros?

- Sí
- No

2. Desde un función se puede llamar a

- A otra función pero no a la propia función
- A la propia función pero no a otra función
- A otra función y a la propia función

3. Cuando una función ejecuta "return", el código que está en la función tras el return

- nunca es ejecutado en esa llamada a la función
- siempre es ejecutado en esa llamada a la función
- bajo algunas condiciones es ejecutado en esa llamada a la función

4. Un parámetro que se pasa a una función es una variable

- global
- local
- estática
- no estática

5. Con respecto a las funciones

- El prototipo de una función puede estar en varios ficheros de un mismo programa
- Si son estáticas, se pueden invocar desde otros ficheros
- Si son estáticas, deben estar en el mismo fichero de un mismo programa

## 4.6. Bibliografía de apoyo

---

- **Funciones (teoría):** "The GNU tutorial", páginas 13-17, 41-45.
- **Paso de parámetros por valor y por referencia:**

- (Teoría): "Programación en C: Metodología, algoritmos y estructura de datos", páginas 250-255.
- (Ejemplos): "El lenguaje de programación C: diseño e implementación de programas", páginas 209-215.
- **Problemas resueltos:** "Problemas resueltos de Programación en Lenguaje C", problemas 5.1-5.10.

## 4.7. Ejercicios

---

1. Un programador con poca experiencia nos manda la siguiente función que dice que intercambia el valor de sus dos parámetros. Si se llama como `swap(x, y)`, al terminar la ejecución, las variables `x` e `y` han intercambiado sus valores. ¿Es eso cierto? Comprueba tu respuesta creando un programa en tu entorno de desarrollo y ejecutándolo. Puedes imprimir la variable entera `x` con la línea `printf("%d\n", x);`.

```

1 void swap(int a, int b)
2 {
3     int tmp;
4     tmp = a;
5     a = b;
6     b = tmp;
7     return;
8 }

```

2. De nuevo este mismo programador nos manda una segunda función que dada una tabla de números enteros incrementa en uno cada uno de sus elementos. ¿Crees que esta función hace realmente esto? (Pista: la definición de la función es correcta, no tiene error alguno de sintaxis).

```

1 void increase(int table[], int size)
2 {
3     int i;
4     for (i = 0; i < size; i++)
5     {
6         table[i]++;
7     }
8 }

```

3. Una aplicación manipula datos obtenidos de un sistema de posicionamiento global a través de la siguiente estructura:

```

1 struct gps_information
2 {
3     int is_3D;
4     float latitude;
5     float longitude;
6     float height;
7 };

```

Escribe el prototipo de una función que recibe dos estructuras de este tipo y un `float`. La función devuelve un entero con valor 1 si latitud y longitud de ambas estructuras están separadas por no más del valor del tercer parámetro y cero en caso contrario. Debes utilizar el sinónimo de tipo de datos que se define.

Escribe la función con su definición completa (quizás te sea de ayuda la función de librería `fabsf`).

4. Escribe una función que recibe como parámetros dos tablas de estructuras como las del ejercicio anterior, un entero con su longitud (que es la misma para ambas), y un `float`. La función procesa las dos tablas a la vez (primero con primero, segundo con segundo, etc.) y devuelve el índice del primer par de elementos que están cerca tal y como se calcula con la función del apartado anterior utilizando el último parámetro como distancia. Si la condición no se cumple para ningún par de elementos, devuelve el valor -1.

Esta función que has implementado es la que se necesita en la aplicación, y no la anterior, que es una función auxiliar y de ámbito reducido. ¿Qué cambio harías en la definición del ejercicio anterior?