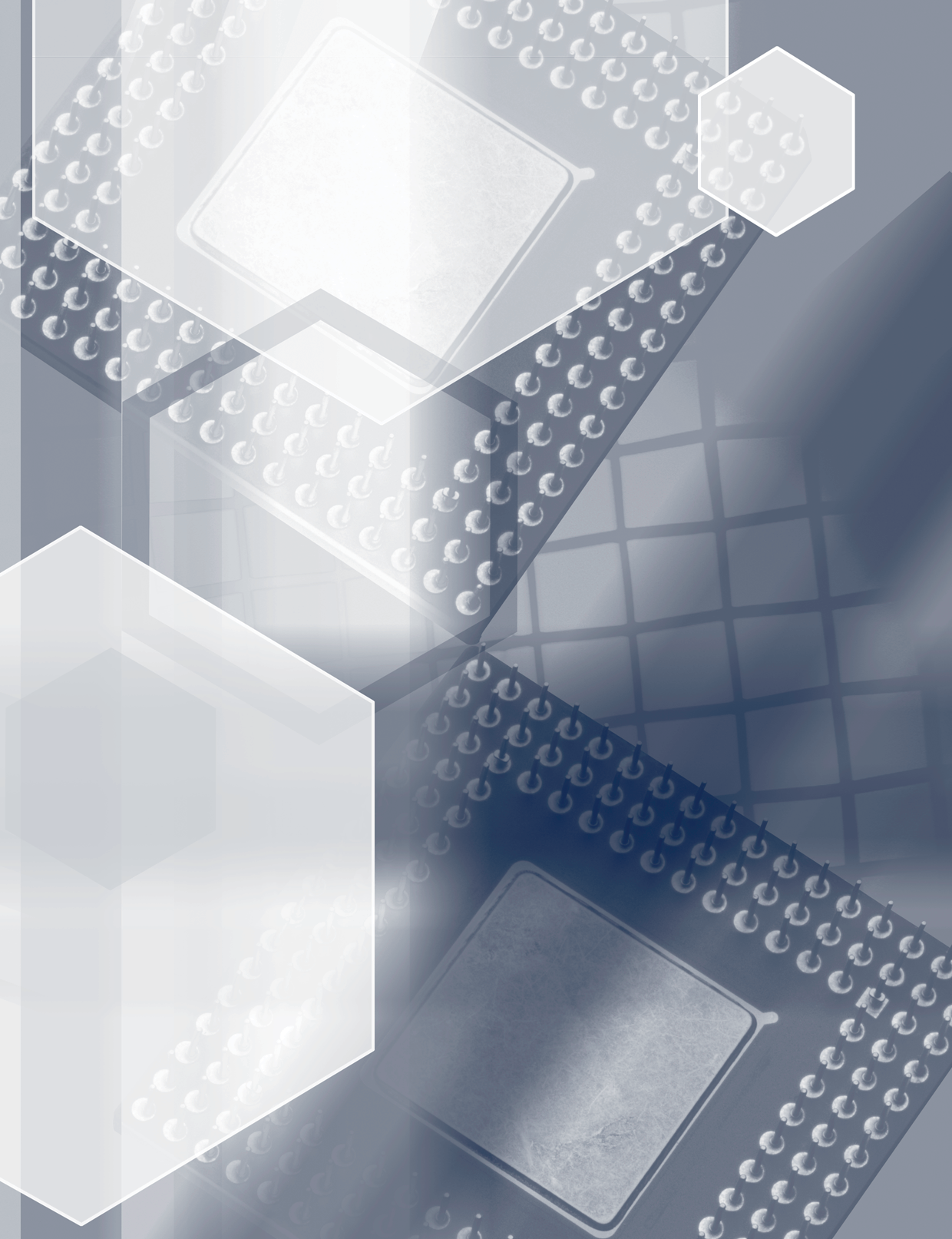
Funciones en C 03



## En esta unidad aprenderás a:

**Comprender el desarrollo de un programa utilizando funciones**

## Establecer la diferencia entre la definición y declaración de una función

**Conocer y utilizar el paso de valores entre funciones y la devolución de valores de una función**

## Establecer la diferencia entre el paso de argumentos

**por valor y por referencia**

## Conocer la clasificación

**de las variables de acuerdo a su ámbito y clase**

**de almacenamiento**



1. **Funciones en C**
   1. Introducción



# Introducción

En la unidad 2 hemos visto que una función es un fragmento de código que realiza una tarea bien definida. Por ejemplo, la función *printf* imprime por la salida estándar los argumentos que le pasamos. Al igual que esta función, existen otras funciones que rea- lizan diversas tareas ya definidas en el estándar ANSI C y que pueden ser utilizadas por el programador. Este tipo de funciones predefinidas son denominadas **funciones de biblioteca.** Sin embargo, cada programador puede definir sus propias funciones de acuerdo a sus necesidades. Las funciones que define el programador son conocidas como **funciones de usuario.**

La utilización de funciones nos permite dividir un programa extenso en pequeños seg- mentos que realizan tareas concretas. Probablemente, dentro de un mismo programa se realicen las mismas tareas varias veces, lo que se facilita mediante la utilización de fun- ciones. Sin embargo, es probable que ciertas funciones no sean reutilizables, pero al usarlas se mejora la legibilidad del programa.

La filosofía en la que se base el diseño de C es el empleo de funciones. Por esta razón, un programa en C contiene al menos una función, la función *main*. Esta función es par- ticular dado que la ejecución del programa se inicia con las instrucciones contenidas en su interior. Una vez iniciada la ejecución del programa, desde la función *main* se puede llamar a otras funciones y, posiblemente, desde estas funciones a otras. Otra particula- ridad de la función *main* es que se llama directamente desde el sistema operativo y no desde ninguna otra función. De esta manera, un programa en C sólo puede contener una función *main*.

Con el propósito de permitir un manejo eficiente de los datos, las funciones en C no se pueden anidar. En otras palabras, una función no se puede declarar dentro de otra fun- ción, por lo que todas las funciones son globales o externas, lo que hace que puedan llamarse desde cualquier parte de un programa.

Se puede acceder (llamar) a una determinada función desde cualquier parte de un pro- grama. Cuando se llama a una función, se ejecutan las instrucciones que constituyen dicha función. Una vez que se ejecutan las instrucciones de la función, se devuelve el control del programa a la siguiente instrucción (si existe) inmediatamente después de la que provocó la llamada a la función.

Cuando se accede a una función desde un determinado punto del programa, se le puede pasar información mediante unos identificadores especiales conocidos como **argumen- tos** (también denominados parámetros). Una vez que la función procesa esta informa- ción, devuelve un valor mediante la instrucción *return*.

La estructura general de una función en C es la siguiente:

tipo\_de\_retorno nombre\_de\_la\_función (lista\_de\_parámetros)

{

cuerpo\_de\_la\_función return expresión

**71**

}

Donde:

* + - **tipo\_de\_retorno:** es el tipo del valor devuelto por la función, o, en caso de que la función no devuelva valor alguno, la palabra reservada *void*.
    - **nombre\_de\_la\_función:** es el nombre o identificador asignado a la función.
    - **lista\_de\_parámetros:** es la lista de declaración de los parámetros que son pasados a la función. Éstos se separan por comas. Debemos tener en cuenta que pueden existir funciones que no utilicen parámetros.
    - **cuerpo\_de\_la\_función:** está compuesto por un conjunto de sentencias que llevan a cabo la tarea específica para la cual ha sido creada la función.
    - **return expresión:** mediante la palabra reservada *return*, se devuelve el valor de la función, en este caso representado por *expresión.*

Vamos a suponer que queremos crear un programa para calcular el precio de un producto basándose en el precio base del mismo y el impuesto aplicable. A continuación mostra- mos el código fuente de dicho programa:

#include <stdio.h>

float precio(float base, float impuesto); /\* declaración \*/

main()

{

}

float importe = 2.5; float tasa = 0.07;

printf("El precio a pagar es: %.2f\n", precio(importe, tasa)); return 0;

float precio(float base, float impuesto) /\* definición \*/

{

float calculo;

calculo = base + (base \* impuesto); return calculo;

}

El ejemplo anterior se compone de dos funciones, la función requerida *main* y la fun- ción creada por el usuario *precio,* que calcula el precio de un producto tomando como parámetros su precio base y el impuesto aplicable. La función *precio* calcula el precio de un producto sumándole el impuesto correspondiente al precio base y devuelve el valor calculado mediante la sentencia *return*.

Por otra parte, en la función *main* declaramos dos variables de tipo *float* que contie- nen el precio base del producto y el impuesto aplicable. La siguiente sentencia dentro de la función *main* es la llamada a la función de biblioteca *printf,* que recibe como pará- metro una llamada a la función *precio,* que devuelve un valor de tipo *float*. De esta manera, la función *printf* imprime por la salida estándar el valor devuelto por la fun- ción *precio*. Es importante tener en cuenta que las variables *importe* y *tasa* (argumen- tos) dentro de la función *main* tienen una correspondencia con las variables *base* e *impuesto* (parámetros) dentro de la función *precio* respectivamente.

En el ejemplo anterior, justo antes de la función *main*, hemos declarado la función *pre- cio*. La intención es que la función *main* sea capaz de reconocerla. Sin embargo, la defi- nición de dicha función aparece después de la función *main*. Las definiciones de fun- ción pueden aparecer en cualquier orden dentro de uno o más ficheros fuentes. Más adelante, en esta unidad, veremos en detalle la declaración y definición de funciones. Por otra parte, hemos añadido la sentencia *return 0* al final de la función *main*, puesto que se trata de una función como otra cualquiera y puede devolver un valor a quien le ha llamado, en este caso el entorno en el que se ejecuta el programa. Generalmente, el valor *0* implica un fin de ejecución normal, mientras que otro valor diferente implica un final de ejecución inusual o erróneo.

# Declaración de funciones

Antes de empezar a utilizar una función debemos **declararla.** La declaración de una fun- ción se conoce también como **prototipo** de la función. En el prototipo de una función se tienen que especificar los parámetros de la función, así como el tipo de dato que devuelve.

Los prototipos de las funciones que se utilizan en un programa se incluyen generalmente en la cabecera del programa y presentan la siguiente sintaxis:

tipo\_de\_retorno nombre\_de\_la\_función(lista\_de\_parámetros);

En el prototipo de una función no se especifican las sentencias que forman parte de la misma, sino sus características. Por ejemplo:

int cubo(int numero);

En este caso se declara la función *cubo* que recibe como parámetro una variable de tipo entero *(numero)* y devuelve un valor del mismo tipo. En ningún momento estamos espe- cificando qué se va a hacer con la variable *numero*, sólo declaramos las características de la función *cubo*.

Cabe señalar que el nombre de los parámetros es opcional y se utiliza para mejorar la comprensión del código fuente. De esta manera, el prototipo de la función *cubo* podría expresarse de la siguiente manera:

int cubo(int);

Los prototipos de las funciones son utilizados por el compilador para verificar que se accede a la función de la manera adecuada con respecto al número y tipo de parámetros, y al tipo de valor de retorno de la misma. Veamos algunos ejemplos de prototipos de funciones:

int potencia(int base, int exponente);

double area\_rectangulo (float base, float altura); int mayor(int, int);

**73**

struct direccion entrada(void);

Las funciones de biblioteca se declaran en lo que se conocen como **ficheros de cabe- cera** o **ficheros .h** (del inglés *headers*, cabeceras). Cuando deseamos utilizar alguna de las funciones de biblioteca, debemos especificar el fichero .h en que se encuentra decla- rada la función, al inicio de nuestro programa. Por ejemplo, si deseamos utilizar la fun- ción *printf* en nuestro programa, debemos incluir el fichero stdio.h que contiene el pro- totipo de esta función.

# Definición de funciones

Tras declarar una función, el siguiente paso es implementarla. Generalmente, este paso se conoce como **definición.** Es precisamente en la definición de una función donde se especifican las instrucciones que forman parte de la misma y que se utilizan para llevar a cabo la tarea específica de la función. La definición de una función consta de dos par- tes, el **encabezado** y el **cuerpo de la función.** En el encabezado de la función, al igual que en el prototipo de la misma, se tienen que especificar los parámetros de la función, si los utiliza y el tipo de datos que devuelve, mientras que el cuerpo se compone de las instrucciones necesarias para realizar la tarea para la cual se crea la función. La sinta- xis de la definición de una función es la siguiente:

tipo\_de\_retorno nombre\_de\_la\_función(lista\_de\_parámetros)

{

sentencias;

}

El *tipo\_de\_retorno* representa el tipo de dato del valor que devuelve la función. Este tipo debe ser uno de los tipos simples de C, un puntero a un tipo de C o bien un tipo *struct*. De forma predeterminada, se considera que toda función devuelve un tipo entero *(int)*. En otras palabras, si en la declaración o en la definición de una función no se especifica el *tipo\_de\_retorno*, el compilador asume que devuelve un valor de tipo *int*. El *nombre\_de\_la\_función* representa el nombre que se le asigna a la función.

Se recomienda que el nombre de la función esté relacionado con la tarea que lleva a cabo. En caso de que la función utilice parámetros, éstos deben estar listados entre paréntesis a continuación del nombre de la función, especificando el tipo de dato y el nombre de cada parámetro. En caso de que una función no utilice parámetros, se pue- den dejar los paréntesis vacíos o incluir la palabra *void*, que indica que la función no utiliza parámetros. Después del encabezado de la función, debe aparecer, delimitado por llaves (*{* y *}*), el cuerpo de la función compuesto por las sentencias que llevan a cabo la tarea específica de la función. Veamos la definición de la función *cubo* decla- rada en el apartado anterior:

int cubo(int base)

{

int potencia;

potencia = base \* base \* base; return potencia;

}

Como ya hemos visto, a los argumentos que recibe la función también se les suele lla- mar **parámetros.** Sin embargo, algunos autores consideran como parámetros a la lista de variables entre paréntesis utilizada en la declaración o en la definición de la función, y como argumentos los valores utilizados cuando se llama a la función. También se uti- lizan los términos **argumentos formales** y **argumentos reales,** respectivamente, para hacer esta distinción.

Cuando un programa utiliza un número elevado de funciones, se suelen separar las decla- raciones de función de las definiciones de las mismas. Al igual que con las funciones de biblioteca, las declaraciones pasan a formar parte de un fichero cabecera (extensión *.h*), mientras que las definiciones se almacenan en un fichero con el mismo nombre que el fichero .h, pero con la extensión *.c*. En algunas ocasiones, un programador no desea divulgar el código fuente de sus funciones. En estos casos, se suele proporcionar al usua- rio el fichero de cabecera, el fichero compilado de las definiciones (con extensión *.o,* de objeto) y una documentación de las mismas. De esta manera, cuando el usuario desea hacer uso de cualquiera de las funciones, sabe qué argumentos pasarle y qué tipo de datos devuelve, pero no tiene acceso a la definición de las funciones.



**Ejemplo práctico**

**1** El siguiente programa calcula el cubo de los números del 1 al 5 utilizando una función definida por el usuario.

#include <stdio.h> int cubo(int base); main()

{

int numero;

for(numero=1; numero<=5; numero++){

printf("El cubo del número %d es %d\n", numero, cubo(numero));

}

return 0;

}

int cubo(int base)

{

int potencia;

potencia = base \* base \* base; return potencia;

}

La salida es:

El cubo del número 1 es 1 El cubo del número 2 es 8 El cubo del número 3 es 27 El cubo del número 4 es 64 El cubo del número 5 es 125

**75**

# Devolución de valores

Una función en C sólo puede devolver un valor. Para devolver dicho valor, se utiliza la palabra reservada *return* cuya sintaxis es la siguiente:

return <expresión>;

Donde *<expresi*ó*n>* puede ser cualquier tipo de dato salvo un array o una función. Ade- más, el valor de la expresión debe coincidir con el tipo de dato declarado en el proto- tipo de la función. Por otro lado, existe la posibilidad de devolver múltiples valores mediante la utilización de punteros o estructuras. Dentro de una función pueden exis- tir varios *return* dado que el programa devolverá el control a la sentencia que ha llamado a la función en cuanto encuentre la primera sentencia *return*. Si no existen *return*, la ejecución de la función continúa hasta la llave del final del cuerpo de la función *(}).* Hay que tener en cuenta que existen funciones que no devuelven ningún valor. El tipo de dato devuelto por estas funciones puede ser *void*, considerado como un tipo especial de dato. En estos casos, la sentencia *return* se puede escribir como *return* o se puede omitir directamente. Por ejemplo:

void imprime\_cabecera();

{

printf("esta función sólo imprime esta línea"); return;

}

equivale a:

void imprime\_cabecera();

{

printf("esta función sólo imprime esta línea");

}

# Acceso a una función

Para que una función realice la tarea para la cual fue creada, debemos acceder o llamar a la misma. Cuando se llama a una función dentro de una expresión, el control del pro- grama se pasa a ésta y sólo regresa a la siguiente expresión de la que ha realizado la lla- mada cuando encuentra una instrucción *return* o, en su defecto, la llave de cierre al final de la función.

Generalmente, se suele llamar a las funciones desde la función *main*, lo que no implica que dentro de una función se pueda acceder a otra función.

Cuando queremos acceder a una función, debemos hacerlo mediante su nombre seguido de la lista de argumentos que utiliza dicha función encerrados entre paréntesis. En caso

de que la función a la que se quiere acceder no utilice argumentos, se deben colocar los paréntesis vacíos.

Cualquier expresión puede contener una llamada a una función. Esta llamada puede ser parte de una expresión simple, como una asignación, o puede ser uno de los operandos de una expresión más compleja. Por ejemplo:

a = cubo(2);

calculo = b + c / cubo(3);

Debemos recordar que los argumentos que utilizamos en la llamada a una función se denominan **argumentos reales.** Estos argumentos deben coincidir en el número y tipo con los **argumentos formales** o **parámetros** de la función. No olvidemos que los argu- mentos formales son los que se utilizan en la definición y/o declaración de una función. Los argumentos reales pueden ser variables, constantes o incluso expresiones más com- plejas. El valor de cada argumento real en la llamada a una función se transfiere a dicha función y se le asigna al argumento formal correspondiente.

Generalmente, cuando una función devuelve un valor, la llamada a la función suele estar dentro de una expresión de asignación, como operando de una expresión compleja o como argumento real de otra función. Sin embargo, cuando la función no devuelve nin- gún valor, la llamada a la función suele aparecer sola. Veamos un ejemplo:

z = potencia( a, b); imprime\_valores (x, y, z);



**Ejemplo práctico**

**2** Vamos a acceder a las funciones *primera* y *segunda* desde la función *main*.

#include <stdio.h>

void primera(void); void segunda(void);

main()

{

printf("La primera función llamada, main\n"); primera();

segunda();

printf("Final de la función main\n"); return 0;

}

void primera(void)

{

printf("Llamada a la función primera\n"); return;

}

**77**

void segunda(void)

{

printf("Llamada a la función segunda\n"); return;

}

La salida es:

La primera función llamada, main Llamada a la función primera Llamada a la función segunda Final de la función main

# Ámbito y clases de almacenamiento

Como hemos visto en unidades anteriores, en C, las variables se pueden clasificar de acuerdo a su tipo de dato. Por ejemplo, una variable puede ser de tipo entero *(int)* o de tipo carácter *(char)*. Sin embargo, las variables también pueden clasificarse de acuerdo a su ámbito, es decir, la parte del programa en la que la variable es reconocida. De acuerdo con su ámbito, las variables pueden ser locales o globales. Por otro lado, exis- ten los modificadores de tipo o clases de almacenamiento que permiten modificar el ámbito y la permanencia de una variable dentro de un programa. Existen cuatro modi- ficadores de tipo, automático, externo, estático y registro, que se corresponden con las palabras reservadas *auto, extern, static* y *register*, respectivamente.

# Variables locales

Cuando declaramos variables dentro de la función principal del programa, es decir, den- tro de la función *main*, están únicamente asociadas a esta función, en otras palabras, son variables locales de la función *main* y no se puede acceder a ellas a través de nin- guna otra función.

Al igual que sucede con las variables declaradas dentro de la función *main*, cualquier varia- ble que declaremos dentro de una función, es local a esa función, es decir, su ámbito esta confinado a dicha función. Esta situación permite que existan variables con el mismo nom- bre en diferentes funciones y que no mantengan ninguna relación entre sí.

Debemos tener en cuenta que cualquier variable declarada dentro de una función se con- sidera como una variable automática *(auto)* a menos que utilicemos algún modificador

de tipo. Una variable se considera **automática** porque cuando se accede a la función se le asigna espacio en la memoria automáticamente y se libera dicho espacio tan pronto se sale de la función. En otras palabras, una variable automática no conserva un valor entre dos llamadas sucesivas a la misma función. Con el propósito de garantizar el con- tenido de las variables automáticas, éstas deben inicializarse al entrar a la función para evitar que su valor sea indeterminado.

Todas las variables que hemos utilizado en los ejemplos vistos hasta ahora son **varia- bles automáticas.** La utilización de la palabra reservada *auto* es opcional, aunque nor- malmente no se utiliza, por ejemplo:

auto int contador;

equivale a

int contador;



**Ejemplo práctico**

**3** Utilización del mismo identificador de variable en diferentes funciones mostrando su localidad.

#include <stdio.h>

void imprimeValor(); main()

{

int contador = 0; contador++;

printf("El valor de contador es: %d\n", contador); imprimeValor();

printf("Ahora el valor de contador es: %d\n", contador); return 0;

}

void imprimeValor()

{

int contador = 5;

printf("El valor de contador es: %d\n", contador);

}

La salida es:

El valor de contador es: 1 El valor de contador es: 5

Ahora el valor de contador es: 1

**79**