

(Versión de fecha 23 de noviembre de 2023)

Se desarrollan en este documento ejemplos sencillos de codificación y uso de funciones en el lenguaje C, con el objetivo de que los alumnos comprendan el funcionamiento de las mismas.

## 1 INTRODUCCIÓN

El código de los ejemplos que se desarrollan en este documento supone que la compilación y ejecución de los programas se realiza en la consola del ordenador. En el caso de ejecutarlos en el modo *external terminal* de VSCode, habrá que añadir una instrucción `system("pause")` al final del programa principal, para que no se cierre la consola hasta que se pulse una tecla. Además, para poder utilizar la instrucción `system()`, es necesario añadir en la cabecera del programa la instrucción `#include <stdlib.h>`, para que se incluya la librería correspondiente. El esquema sería el siguiente:

```
#include <stdlib.h>

int main() {

    // ...
    // Código del programa
    // ...

    system("pause");
    return 0;
}
```

## 2 FUNCIONES CON VALORES ESCALARES

### Problema 1.- Función sin parámetros ni valor devuelto

**Apartado 1** - Desarrolle una función en lenguaje C de nombre *saludo()*, que imprima en pantalla la frase *Hola, ¿qué tal?*; la función no requerirá ningún parámetro ni devolverá ningún resultado.

**Apartado 2** - Desarrolle un programa que utilice la función del Apartado 1 para imprimir el mensaje en pantalla.

### Solución 1: Problema 1, Apartado 1

```
void saludo() {
    printf("Hola, ¿qué tal?\n");
}
```

En la función anterior, se ha indicado el tipo *void* como valor devuelto, lo que equivale a decir que no devuelve ningún valor y, por tanto, no necesita tener ninguna sentencia *return* en el código de la función.

En la cabecera de la función se han dejado los paréntesis vacíos, para indicar que no necesita ningún parámetro. Se podría haber puesto también la cláusula *void* dentro de los paréntesis, pero no es necesario y no se suele hacer.

La instrucción *printf()* necesita que el programa que utilice la función *saludo()* incorpore la librería *stdio.h*.

### Solución 2: Problema 1, Apartado 2

```
#include <stdio.h>

void saludo() {
    printf("Hola, ¿qué tal?\n");
}

int main() {
    saludo();
    return 0;
}
```

La solución del programa anterior está pensada para compilarse y ejecutarse directamente en la consola (terminal) del ordenador. Si el código se guarda en un fichero llamado *hello.c*, la orden que habría que teclear en la consola para compilarlo y crear el programa ejecutable *hello.exe* sería la siguiente:

```
gcc hello.c -o hello.exe
```

Para ejecutar el programa resultante, hay que teclear la siguiente orden en la consola:

```
hello.exe
```

O simplemente:

```
hello
```

En un documento posterior se explicará cómo utilizar la consola para compilar y ejecutar programas. También se explicará en dicho documento como resolver el problema de mostrar en la consola la letra ñ, los caracteres acentuados y otros caracteres especiales.

### Problema 2.- Función sin parámetros que devuelve un valor

**Apartado 1** - Desarrolle una función en lenguaje C de nombre *pi()*, que devuelva un valor aproximado de  $\pi$ ; la función no requerirá ningún parámetro.

**Apartado 2** - Desarrolle un programa que solicite al usuario el valor del radio de un círculo y, utilizando la función del Apartado 1, calcule e imprima en pantalla el área del círculo y la longitud de la circunferencia.

### Solución 3: Problema 2, Apartado 1

```
double pi() {
    return 3.14159265359;
}
```

#### Solución 4: Problema 2, Apartado 2

```
#include <stdio.h>

double pi() {
    return 3.14159265359;
}

int main() {
    double R=0.0;
    printf("Radio: ");
    scanf("%lf", &R);

    double area = pi()*R*R;
    double circun = 2*pi()*R;

    printf("L=%.2f S=%.2f \n", area, circun);

    return 0;
}
```

El procedimiento mostrado para utilizar el valor de  $\pi$  en los programas es totalmente correcto, si bien hay otros procedimientos habituales para utilizar constantes en los programas. Se podría utilizar una cláusula `#define`, como se hace en el código siguiente:

```
#define PI 3.14159265359

int main() {
    // ...
    double area = PI*R*R;
    // ...
}
```

También se podría utilizar una variable global constante del programa:

```
const double PI=3.14159265359;

int main() {
    // ...
    double area = PI*R*R;
    // ...
}
```

Cada uno de los tres procedimientos descritos para utilizar constantes en los programas tiene sus ventajas y sus inconvenientes. En el caso concreto del valor de  $\pi$  se puede utilizar también el valor proporcionado por la librería *math.h*, que se llama `M_PI`:

```
#include <math.h>

int main() {
    // ...
    double area = M_PI*R*R;
    // ...
}
```

Por último, y para el caso del valor de  $\pi$ , se podría utilizar el método de la función *pi()* que se ha mostrado al principio de este apartado, pero utilizando como valor de  $\pi$  el ángulo cuyo coseno vale  $-1$ , que es el ángulo que

mide  $\pi$  radianes. De esta forma, se obtendría un valor de  $\pi$  con más decimales que el que se utilizó en el código de la Solución 3:

```
#include <math.h>

double pi() {
    return acos(-1.0);
}
```

### Problema 3.- Ejemplos de paso de parámetros por valor y por referencia

Se pretende realizar un programa que solicite al usuario un número *double* y calcule e imprima en pantalla dicho número multiplicado por 2. Para ello, se quiere utilizar una función que calcule dicho producto por 2. Se van a considerar varias soluciones.

**Apartado 1 (Paso por referencia)** - Desarrolle un programa que incluya y utilice una función en lenguaje C que reciba como parámetro un puntero a un número *double* y lo utilice para multiplicar por 2 el valor al que apunta el puntero.

**Apartado 2 (Paso por referencia)** - Desarrolle programa que incluya y utilice una función con dos parámetros: el primero es un valor *double* y el segundo es la dirección de memoria de una variable en la que la función escribirá el resultado de multiplicar por 2 el valor del primer parámetro. La función no devuelve ningún valor.

**Apartado 3 (Paso por valor)** - Desarrolle un programa que incluye y utilice una función que reciba como parámetro un número *double* y devuelva el valor de dicho número multiplicado por 2.

### Solución 5: Problema 3, Apartado 1

```
#include <stdio.h>

void producto_por_dos(double* ptr) {
    *ptr = *ptr * 2;
}

int main() {
    double x = 0.0;
    printf("x= ");
    scanf("%lf", &x);

    producto_por_dos(&x);

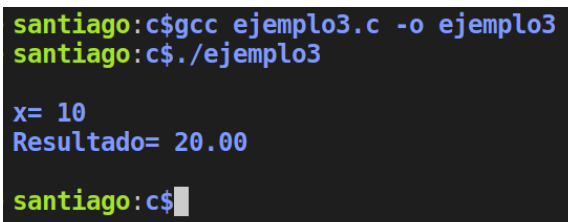
    printf("Resultado= %.2f \n", x);
    return 0;
}
```

Observe la Solución 5. La función *producto\_por\_dos()* recibe como parámetro un puntero *ptr* a cierto valor *double*, esto es, la dirección de memoria en la que está guardado dicho valor *double*. A este mecanismo de paso de parámetros se le denomina *paso por referencia*: se pasa la dirección de memoria donde está guardado un valor, no el valor en sí mismo.

Como la función conoce la posición de memoria en la que se guarda el valor, puede acceder a ella y modificar dicho valor. Es lo que hace el código de la función: el contenido de la posición de memoria apuntada por *ptr* lo hace igual al valor que tenía multiplicado por 2.

La función no necesita devolver ningún valor pues, con esta forma de proceder, la función modifica directamente el valor de una variable que *pertenece* a la función principal *main()*. Por su parte, la función *main()* hace la llamada a la función pasándole *&x*, que es la dirección de memoria de la variable *x*. Tras ejecutarse la función, el valor de la variable *x* queda multiplicado por 2.

La salida del programa es la que muestra la Figura 1.



```
santiago:c$gcc ejemplo3.c -o ejemplo3
santiago:c$./ejemplo3

x= 10
Resultado= 20.00

santiago:c$
```

Figura 1: Salida del programa de la Solución 5.

#### Solución 6: Problema 3, Apartado 2

```
#include <stdio.h>

void producto_por_dos(double num, double* ptr) {
    *ptr = num * 2;
}

int main() {
    double x = 0.0;
    double result=0.0;
    printf("\nx= ");
    scanf("%lf", &x);

    producto_por_dos(x, &result);

    printf("Resultado= %.2f \n\n", result);
    return 0;
}
```

La salida por pantalla del programa en la Solución 6 es idéntica a la de la Solución 5. En este caso, la llamada desde *main()* necesita pasar como parámetros el valor de *x* y la dirección de memoria donde guarda sus valores la variable en la que se quiere obtener el resultado, en este caso, *&result*. Una vez más, la función *producto\_por\_dos()* modifica directamente el valor de una variable que pertenece a la función *main()*.

#### Solución 7: Problema 3, Apartado 3

```
#include <stdio.h>

double producto_por_dos(double num) {
    return num*2;
}
```

```

int main() {
    double x = 0.0;
    double result=0.0;
    printf("\nx= ");
    scanf("%lf", &x);

    result = producto_por_dos(x);

    printf("Resultado= %.2f \n\n", result);
    return 0;
}

```

La salida por pantalla es idéntica a las anteriores. En este caso, la función *producto\_por\_dos()* recibe como parámetro un valor, no una referencia, y devuelve otro valor. Si se quisiera modificar el valor de la variable *x* en la función *main()*, en vez de recoger el resultado en una variable diferente, se podría asignar el resultado a la propia variable *x*:

```
x = producto_por_dos(x);
```

Las tres soluciones explicadas resuelven el problema planteado pero, siempre que sea posible, es preferible utilizar soluciones del tipo de la tercera, en la que se pasan valores y se devuelven valores, sin que la función modifique de ninguna manera variables externas a ella. Es lo que se denominan *funciones puras*. Cuando una función modifica valores de variables que están definidas en otras funciones se pueden producir *bugs* en los programas que son muy difíciles de detectar.

Lo importante es que el alumno entienda perfectamente las tres formas de resolver el problema. Luego, en cada caso concreto, deberá elegir la solución que mejor se adapte a la situación planteada.

Como se verá más adelante, cuando las funciones necesitan modificar un array, se hace necesario utilizar referencias y que la función modifique variables del tipo array que se han definido en otras funciones.

### 3 FUNCIONES PARA MANEJO DE ARRAYS

En general, en el lenguaje C todos los parámetros de las funciones se pasan por valor, excepto los arrays. Cuando el parámetro de una función es un array, no se pasa por valor, sino que lo que recibe la función es un puntero al primer elemento del array.

Como es sabido, el nombre de una variable del tipo array es equivalente a un puntero al primer elemento del array. Por ejemplo, sea el array *A* definido de la siguiente manera:

```
int A[5];
```

En este caso, las expresiones siguientes se pueden considerar equivalentes:

$$A \equiv \&A[0]$$

Para indicar a una función que un parámetro es un array, se pone el tipo de datos, un nombre de variable y una pareja de corchetes vacía. Además, hay que añadir un parámetro adicional de tipo entero que le indique a la función el número de elementos del array. Por ejemplo, la siguiente podría ser la declaración de una función que recibe como parámetro un array y el tamaño del mismo:

```
int mi_funcion(int A[], int tamano);
```

Dentro del cuerpo de la función, la variable *A* se puede utilizar como si fuera un array, pero en realidad es un puntero y la función no conoce el tamaño del array. Por eso, es necesario el segundo parámetro. No sirve de nada indicar el tamaño entre los corchetes. La función sigue recibiendo un puntero y sigue sin saber el tamaño del array.

#### Problema 4.- Arrays como parámetros

**Apartado 1** - Desarrolle una función en lenguaje C que reciba como parámetros un array de números *doubles* y un número entero con el tamaño de dicho array y devuelva la suma de los elementos del array.

**Apartado 2** - Desarrolle un programa en lenguaje C que haga uso de la función anterior.

#### Solución 8: Problema 4, Apartado 1, solución 1

```
double sum_array(double v[], int tamano) {
    double suma = 0.0;
    for(int i=0; i<tamano; i++) {
        suma = suma + v[i];
    }
    return suma;
}
```

Observe que, en el cuerpo de la función, se accede a los elementos del array utilizando la notación de índice entre corchetes, como corresponde a un array. Pero también se podría utilizar la notación de punteros, de la siguiente forma:

#### Solución 9: Problema 4, Apartado 1, solución 2

```
double sum_array(double v[], int tamano) {
    double suma = 0.0;
    for(int i=0; i<tamano; i++) {
        suma = suma + *(v+i);
    }
    return suma;
}
```

De hecho, se podría declarar el parámetro como si fuera un puntero números *double*, con la siguiente declaración:

```
double sum_array(double* v, int tamano);
```

Eso sí, siempre es necesario decirle a la función el tamaño del array al que apunta el puntero.

Independientemente de que el parámetro se declare explícitamente como array o como puntero, dentro del cuerpo de la función se puede acceder a los elementos con nomenclatura de índice o de punteros. Por tanto se dan cuatro posibilidades equivalentes:

- Parámetro array, acceso a elementos con índice.
- Parámetro array, acceso a elementos con aritmética de punteros.
- Parámetro puntero, acceso a elementos con índice.
- Parámetro puntero, acceso a elementos con aritmética de punteros.

La función definida en las Soluciones 8 y 9, en la que se ha utilizado un array como parámetro, se podrían haber hecho también declarando el parámetro como puntero, dando lugar a las soluciones 10 y 11, que también

son soluciones válidas. El lector debe observar que la diferencia entre las distintas soluciones está solo en la forma de declarar el parámetro (array o puntero) y en la forma de acceder a los elementos del array (índice o aritmética de punteros).

#### **Solución 10:** Problema 4, Apartado 1, solución 3

```
double sum_array(double* v, int tamano) {
    double suma = 0.0;
    for(int i=0; i<tamano; i++) {
        suma = suma + v[i];
    }
    return suma;
}
```

#### **Solución 11:** Problema 4, Apartado 1, solución 4

```
double sum_array(double* v, int tamano) {
    double suma = 0.0;
    for(int i=0; i<tamano; i++) {
        suma = suma + *(v+i);
    }
    return suma;
}
```

Una vez codificada la función de alguna de las cuatro formas posibles, el programa que hace uso de ella sería el mismo:

#### **Solución 12:** Problema 4, Apartado 2

```
#include <stdio.h>

double sum_array(double v[], int tamano) {
    double suma = 0.0;
    for(int i=0; i<tamano; i++) {
        suma = suma + v[i];
    }
    return suma;
}

int main() {
    double ar[] = {1,2,3};

    double s = sum_array(ar, 3);

    printf("S=%.2f\n", s);
}
```

Observe la llamada a la función `sum_array()` desde el programa principal:

```
sum_array(ar, 3);
```

Se pone simplemente el nombre de la variable del array (sin corchetes). Esta forma de llamar a la función es válida independientemente de que la función haya declarado el parámetro como array o como puntero.



### Problema 5.- Función que modifica un array

**Apartado 1** - Desarrolle una función en lenguaje C que reciba como parámetros un array de números *doubles* y un número entero con el tamaño de dicho array y modifique el array recibido de forma que los elementos con valor negativo los haga valer 0.

**Apartado 2** - Desarrolle un programa en lenguaje C que haga uso de la función anterior.

En este caso se trata de modificar el array que se recibe como parámetro. Al recibir el array por referencia, es posible modificar el valor de sus elementos y las modificaciones persistirán cuando se vuelva de la función al programa principal.

#### Solución 13: Problema 5, Apartado 1

```
void modify_array(double v[], int tamano) {
    for(int i=0; i<tamano; i++) {
        if(v[i]<0) {
            v[i] = 0;
        }
    }
}
```

Observe que, en este caso, la función no devuelve ningún valor, ya que el array hace el papel de parámetro de entrada y, una vez modificado, de parámetro de salida.

El programa que hace uso de la función podría ser el siguiente:

#### Solución 14: Problema 5, Apartado 2

```
#include <stdio.h>

void modify_array(double v[], int tamano) {
    for(int i=0; i<tamano; i++) {
        if(v[i]<0) {
            v[i] = 0;
        }
    }
}

int main() {
    double ar[] = {1,-2,3};

    modify_array(ar, 3);

    for(int i=0; i<3; i++) {
        printf("%.2f ", ar[i]);
    }
    printf("\n");
}
```

## Problema 6.- Función que construye un array

**Apartado 1** - Desarrolle una función en lenguaje C que reciba como parámetros dos arrays de números *doubles* y un número entero con el tamaño de dichos arrays; la función copiará en el segundo array los elementos del primer array en orden inverso: el primer elemento del segundo array será el último elemento del primer array y así sucesivamente.

**Apartado 2** - Desarrolle un programa en lenguaje C que haga uso de la función anterior.

En este caso, la función recibe dos arrays: el primer array no se modifica, se utiliza para realizar los cálculos necesarios para rellenar el segundo array. La función solicitada se podría codificar de la siguiente manera:

### Solución 15: Problema 6, Apartado 1

```
void invierte_array(double v[], double w[], int tamano) {
    for(int i=0; i<tamano; i++) {
        w[i] = v[tamano-1-i];
    }
}
```

Observe que la función no necesita devolver ningún valor. En este caso, el primer array hace el papel de parámetro de entrada y el segundo array hace el papel de parámetro de salida. Para que la función pueda rellenar de manera correcta el segundo array, el programa principal tiene que haberlo declarado previamente con el tamaño adecuado, como se muestra a continuación:

### Solución 16: Problema 6, Apartado 2

```
#include <stdio.h>

void invierte_array(double v[], double w[], int tamano) {
    for(int i=0; i<tamano; i++) {
        w[i] = v[tamano-1-i];
    }
}

int main() {
    double ar1[] = {1,-2,3};
    double ar2[3];

    invierte_array(ar1, ar2, 3);

    for(int i=0; i<3; i++) {
        printf("%.2f ", ar2[i]);
    }
    printf("\n");
}
```

En casos como el mostrado, en los que uno de los arrays parámetro no se va a modificar, es frecuente declarar el parámetro como constante. El funcionamiento es el mismo, pero si se intentara modificar el array declarado como constante, se produciría un error de compilación. De utilizar este criterio, el código de la Solución 16 quedaría como sigue:

#### Solución 17: Definición de parámetro constante

```
void invierte_array(const double v[], double w[], int tamano) {
    for(int i=0; i<tamano; i++) {
        w[i] = v[tamano-1-i];
    }
}
```

## 4 PROTOTIPOS DE LAS FUNCIONES

Todos los ejemplos que se han mostrado en los apartados anteriores son sencillos, con una sola función y el programa principal. En todos los ejemplos se ha dispuesto el código de la función auxiliar antes que el código de la función *main()*, que hace uso de ella. Cuando los programas constan de varias funciones auxiliares, es más frecuente poner delante de *main()* los prototipos de las funciones y el código de las mismas después del código de la función *main()*. De esta forma, es más cómodo trabajar en el programa o comprender su funcionamiento cuando se lee.

El prototipo de la función es la primera línea de la misma, donde se detallan el tipo del valor devuelto, el nombre la función y el tipo de los parámetros que recibe la función. El prototipo se termina en punto y coma. Por ejemplo, el programa de la Solución 18, quedaría de la siguiente forma utilizando un prototipo para la función *invierte\_array()*:

#### Solución 18: Ejemplo de programa con prototipo de función

```
#include <stdio.h>

void invierte_array(double v[], double w[], int tamano);

int main() {
    double ar1[] = {1,-2,3};
    double ar2[3];

    invierte_array(ar1, ar2, 3);

    for(int i=0; i<3; i++) {
        printf("%.2f ", ar2[i]);
    }
    printf("\n");
}

void invierte_array(double v[], double w[], int tamano) {
    for(int i=0; i<tamano; i++) {
        w[i] = v[tamano-1-i];
    }
}
```

En el prototipo de la función no se necesita el nombre de las variables de los parámetros, es suficiente con poner el tipo de datos. De esta forma, el prototipo de la función *invierte\_array()* se podría haber puesto:

```
void invierte_array(double[], double[], int);
```

Es habitual también que la documentación explicativa de la función se ponga en la línea del prototipo, no en el código de la función.