CAPÍTULO 5

Introducción a la gestión de subrutinas

Índice

Una subrutina es un trozo de código independiente, que normalmente realiza una tarea auxiliar completa, que se puede llamar y ejecutar desde cualquier parte de un programa y que, una vez ejecutado, devuelve el control al programa inmediatamente después de donde se había efectuado la llamada. Siendo habitual que al llamar una subrutina se le pasen parámetros para operar con ellos o para seleccionar distintos comportamientos, y que aquélla, al terminar, devuelva valores al programa que la llamó.

Conviene tener en cuenta que dependiendo del lenguaje de programación y de sus particularidades, una subrutina puede recibir cualquiera de los siguientes nombres: *rutina*, *procedimiento*, *función* o *método*. De hecho, es probable que ya hayas oído hablar de alguno de ellos. Por

ejemplo, en Python3 y en Java, en lugar de hablar de subrutinas se habla de funciones y métodos. Aunque en realidad sí que hay diferencias de significado entre algunos de los términos anteriores, cuando en este texto utilicemos el término subrutina, nos estaremos refiriendo indistintamente a cualquiera de los anteriores.

Para ver con más detalle en qué consiste una subrutina utilizaremos el siguiente programa en Python3, en el que se puede ver un ejemplo de subrutina, llamada «`multadd`».

```
1 def multadd(x, y, a):
2     res = x*y + a
3     return res
4
5 r1 = multadd(5, 3, 2)
6 r2 = multadd(2, 4, 1)
```

La sintaxis de Python3 para declarar el inicio de una subrutina es «`def nombre(param1, param2...):`». En dicha línea se da nombre a la subrutina y se especifica el nombre de cada uno de los parámetros que espera recibir dicha subrutina. En el ejemplo anterior, la subrutina se llama «`multadd`» y espera recibir tres parámetros, a los que nombra `x`, `y` y `z` (nombres que se utilizarán por la subrutina para identificar a los parámetros recibidos). El código que haya a continuación, mientras esté convenientemente indentado con respecto a la línea anterior, se considerará que es parte de la subrutina. Por último, en el caso de que la subrutina devuelva algún resultado, deberá haber al menos una línea con la sintaxis «`return value`». En el ejemplo anterior se puede ver que la subrutina devuelve el valor de la variable `res`.

Una vez que sabemos qué trozo del código corresponde a la subrutina «`multadd`», la siguiente pregunta es qué acciones lleva a cabo dicha subrutina, que son: I) realiza la operación $res \leftarrow x \cdot y + a$ con los parámetros x, y y a que recibe al ser llamada y, una vez completada la operación, II) devuelve el control al punto siguiente desde el que fue llamada, que puede acceder al resultado calculado.

Como se puede observar, la subrutina «`multadd`» es llamada dos veces desde el programa principal. La primera de ellas, los parámetros x, y y a toman los valores 5, 3 y 2, respectivamente. Mientras que la segunda vez, toman los valores 2, 4 y 1.

Cuando se ejecuta la instrucción «`r1 = multadd(5, 3, 2)`», el control se transfiere a la subrutina «`multadd`», que calcula $5 \cdot 3 + 2$ y devuelve el control al programa principal, que, a su vez, almacena el resultado, 17, en la variable `r1`.

A continuación comienza la ejecución de la siguiente instrucción del programa, «`r2 = multadd(2, 4, 1)`». y el control se transfiere de nuevo a la subrutina «`multadd`», quien ahora calcula $2 \cdot 4 + 1$ y devuelve de

nuevo el control al programa principal, pero en esta ocasión al punto del programa en el que se almacena el resultado, 9, en la variable `r2`.

Por tanto, y como se ha podido comprobar, «`multadd`» responde efectivamente a la definición dada de subrutina: es un trozo de código independiente, que realiza una tarea auxiliar completa, que se puede llamar y ejecutar desde cualquier parte de un programa y que, una vez ejecutado, devuelve el control al programa inmediatamente después de donde se había efectuado la llamada. A «`multadd`» se le pasan parámetros para operar con ellos, y, al terminar, devuelve un valor al programa que la llamó.

El uso de subrutinas presenta varias ventajas. La primera de ellas es que permite dividir un problema largo y complejo en subproblemas más sencillos. La ventaja de esta división radica en la mayor facilidad con la que se puede escribir, depurar y probar cada uno de los subproblemas por separado. Esto es, se puede desarrollar y probar una subrutina independientemente del resto del programa y posteriormente, una vez que se ha verificado que su comportamiento es el esperado, se puede integrar dicha subrutina en el programa que la va a utilizar.

Otra ventaja de programar utilizando subrutinas es que si una misma tarea se realiza en varios puntos del programa, no es necesario escribir el mismo código una y otra vez a lo largo del programa. Si no fuera posible utilizar subrutinas se debería repetir la misma fracción de código en todas y cada una de las partes del programa en las que éste fuera necesario. Es más, si en un momento dado se descubre un error en un trozo de código que se ha repetido en varias parte del programa, sería necesario revisar todas las partes del programa en las que éste aparece para rectificar en cada una de ellas el mismo error. De igual forma, cualquier mejora de dicha parte del código implicaría revisar todas las partes del programa en las que aparece. Por el contrario, si se utiliza una subrutina y se detecta un error o se quiere mejorar su implementación, basta con modificar su código; una sola vez.

En los párrafos anteriores se ha mostrado cómo la utilización de subrutinas permite reducir tanto el tiempo de desarrollo del código como el de su depuración. Sin embargo, el uso de subrutinas tiene una ventaja de mayor calado: subproblemas que aparecen con frecuencia en el desarrollo de ciertos programas pueden ser implementados como subrutinas y agruparse en bibliotecas (*libraries* en inglés[1]). Cuando un programador requiere resolver un determinado problema ya implementado, le basta con recurrir a una determinada biblioteca y llamar la subrutina ade-

[1] La palabra inglesa *library* se suele traducir erróneamente en castellano como «librería»; la traducción correcta es «biblioteca».

cuada. Es decir, gracias a la agrupación de subrutinas en bibliotecas, el mismo código puede ser reutilizado por muchos programas.

Desde el punto de vista que nos ocupa, el de los mecanismos que proporciona un procesador para soportar determinadas tareas de programación, el uso de subrutinas implica que se deben facilitar las siguientes acciones: llamar una subrutina; pasar los parámetros con los que debe operar la subrutina; devolver los resultados; y continuar la ejecución del programa a partir de la siguiente instrucción en código máquina a la que invocó a la subrutina. Es por ello que en este capítulo se presentan los aspectos más básicos relacionados con la gestión de subrutinas en el ensamblador Thumb de ARM. Es decir, cómo llamar y retornar de una subrutina y cómo intercambiar información entre el programa invocador y la subrutina utilizando registros.

Para complementar la información mostrada en este capítulo y obtener otro punto de vista sobre este tema, se puede consultar el Apartado 3.8 «*Subroutine Call and Return*» del libro «*Computer Organization and Architecture: Themes and Variations*» de Alan Clements. Conviene tener en cuenta que en dicho apartado se utiliza el juego de instrucciones ARM de 32 bits y la sintaxis del compilador de ARM, mientras que en este libro se describe el juego de instrucciones Thumb de ARM y la sintaxis del compilador GCC.

5.1. Llamada y retorno de una subrutina

El juego de instrucciones ARM dispone de las siguientes instrucciones para gestionar la llamada y el retorno de una subrutina:

- «**`bl etiqueta`**» Se utiliza en el programa invocador para **llamar** la subrutina que comienza en la dirección de memoria indicada por la etiqueta. La ejecución de esta instrucción conlleva las siguientes acciones:
  - Se almacena la dirección de memoria de la siguiente instrucción a la que contiene la instrucción «**`bl etiqueta`**» en el registro `r14` (también llamado `lr`, por *link register*). Es decir, `lr`←`PC`+`4`[2].
  - Se lleva el control del flujo del programa a la dirección indicada en el campo «`etiqueta`». Es decir, se realiza un salto incondicional a la dirección especificada en «`etiqueta`» (`PC`←`etiqueta`).

[2]La instrucción «**`bl etiqueta`**» se codifica utilizando dos medias palabras, de ahí que al `PC` se le sume `4` para calcular la dirección de retorno.

- «`mov pc, lr`» Se utiliza en la subrutina para **retornar** al programa invocador. Esta instrucción actualiza el contador de programa con el valor del registro `lr`, lo que a efectos reales implica realizar un salto incondicional a la dirección contenida en el registro `lr`. Es decir, `PC←lr`.

La correcta utilización de estas dos instrucciones permite realizar de forma sencilla la llamada y el retorno desde una subrutina. En primer lugar, el programa invocador debe llamar la subrutina utilizando la instrucción «`bl etiqueta`». Esta instrucción almacena en `lr` la dirección de vuelta y salta a la dirección indicada por la etiqueta. Por último, cuando finalice la subrutina, para retornar al programa que la llamó, ésta debe ejecutar la instrucción «`mov pc, lr`».

Esta forma de proceder será la correcta siempre que el contenido del registro `lr` no se modificado durante la ejecución de la subrutina. Este funcionamiento queda esquematizado en la Figura 5.1.

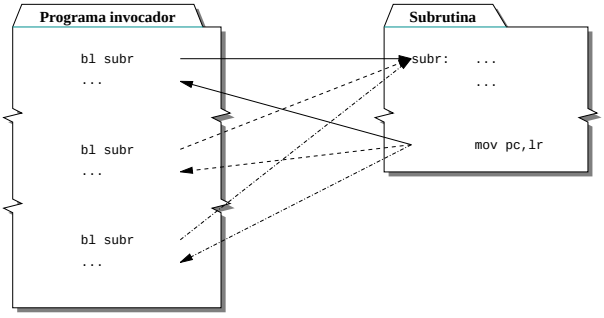

Figura 5.1: Llamada y retorno de una subrutina

Como primer ejemplo de subrutinas en ensamblador de ARM, el siguiente código muestra un programa que utiliza una subrutina llamada «`suma`». Esta subrutina suma los valores almacenados en los registros `r0` y `r1`, y devuelve la suma de ambos en el registro `r2`. Como se puede observar, la subrutina se llama desde dos puntos del programa principal (la primera línea del programa principal se ha etiquetado con «`main`»).

introsub-suma-valor.s

```
1           .data
2  datos:   .word 5, 8, 3, 4
3  suma1:   .space 4
4  suma2:   .space 4
5           .text
6
7  @ Programa invocador
8
9  main:    ldr r4, =datos
```

```
10
11          ldr r0, [r4]
12          ldr r1, [r4, #4]
13  primera: bl suma
14          ldr r5, =suma1
15          str r2, [r5]
16
17          ldr r0, [r4, #8]
18          ldr r1, [r4, #12]
19  segunda: bl suma
20          ldr r5, =suma2
21          str r2, [r5]
22
23  stop:   wfi
24
25  @ Subrutina
26
27  suma:   add r2, r1, r0
28          mov pc, lr
29
30          .end
```

Carga el programa anterior en el simulador y contesta a las siguientes preguntas mientras realizas una ejecución paso a paso.

EJERCICIOS

▶ **5.1** ¿Cuál es el contenido del `PC` y del registro `lr` antes y después de ejecutar la instrucción «`bl suma`» etiquetada como «`primera`»?

▶ **5.2** ¿Cuál es el contenido de los registros `PC` y `lr` antes y después de ejecutar la instrucción «`mov pc, lr`» la primera vez que se ejecuta la subrutina «`suma`»?

▶ **5.3** ¿Cuál es el contenido del `PC` y del registro `lr` antes y después de ejecutar la instrucción «`bl suma`» etiquetada como «`segunda`»?

▶ **5.4** ¿Cuál es el contenido de los registros `PC` y `lr` antes y después de ejecutar la instrucción «`mov pc, lr`» la segunda vez que se ejecuta la subrutina «`suma`»?

▶ **5.5** Anota el contenido de las variables «`suma1`» y «`suma2`» después de ejecutar el programa anterior.

▶ **5.6** Crea un nuevo programa a partir del anterior en el que la subrutina «`suma`» devuelva en `r2` el doble de la suma de `r1` y `r0`. Ejecuta el programa y anota el contenido de las variables «`suma1`» y «`suma2`».

5.2. Paso de parámetros

Se denomina *paso de parámetros* al mecanismo mediante el cual el programa invocador y la subrutina intercambian datos.

Los parámetros intercambiados entre el programa invocador y la subrutina pueden ser de tres tipos según la dirección en la que se transmita la información: de *entrada*, de *salida* o de *entrada/salida*. Se denominan parámetros de entrada a los que proporcionan información del programa invocador a la subrutina. Se denominan parámetros de salida a los que devuelven información de la subrutina al programa invocador. Por último, los parámetros de *entrada/salida* proporcionan información del programa invocador a la subrutina y devuelven información de la subrutina al programa invocador.

Por otro lado, para realizar el paso de parámetros es necesario disponer de algún lugar físico donde se pueda almacenar y leer la información que se quiere transferir. Las dos opciones más comunes son: utilizar registros o utilizar la pila. Que se utilicen registros o la pila depende de la arquitectura en cuestión y del convenio que se siga para el paso de parámetros en dicha arquitectura.

Este apartado se va a ocupar únicamente del paso de parámetros por medio de registros. De hecho, la arquitectura ARM establece un convenio para el paso de parámetros mediante registros: para los parámetros de entrada y de salida se deben utilizar los registros `r0`, `r1`, `r2` y `r3`.

Hasta aquí se ha visto que hay parámetros de entrada, de salida y de entrada/salida. Además, también se ha visto que los parámetros pueden pasarse por medio de registros o de la pila. El último aspecto a considerar del paso de parámetros es cómo se transfiere cada uno de los parámetros. Hay dos formas de hacerlo: un parámetro puede pasarse por valor o por referencia. Se dice que un parámetro se pasa por valor cuando lo que se transfiere es el dato en sí. Un parámetro se pasa por referencia cuando lo que se transfiere es la dirección de memoria en la que se encuentra dicho dato.

Según todo lo anterior, el desarrollo de una subrutina implica determinar en primer lugar:

- El número de parámetros necesarios.
- Cuáles son de entrada, cuáles de salida y cuáles de entrada/salida.
- Si se van a utilizar registros o la pila para su transferencia.
- Qué parámetros deben pasarse por valor y qué parámetros por referencia.

Naturalmente, el programa invocador deberá ajustarse a los requerimientos que haya fijado el desarrollador de la subrutina en cuanto a cómo se debe realizar el paso de parámetros.

En los siguientes apartados se utilizarán parámetros de entrada, salida o entrada/salida según sea necesario. Además, el paso de parámetros se hará utilizando los registros fijados por el convenio ARM. El primero de los siguientes apartados muestra cómo se realiza el paso de parámetros por valor y el segundo, cómo se realiza el paso de parámetros por referencia.

5.2.1. Paso de parámetros por valor

El paso de parámetros por valor implica la siguiente secuencia de acciones (ver Figura 5.2):

1. Antes de realizar la llamada a la subrutina, el programa invocador carga el valor de los parámetros de entrada en los registros correspondientes.

2. La subrutina, finalizadas las operaciones que deba realizar y antes de devolver el control al programa invocador, carga el valor de los parámetros de salida en los registros correspondientes.

3. El programa invocador recoge los parámetros de salida de los registros correspondientes.

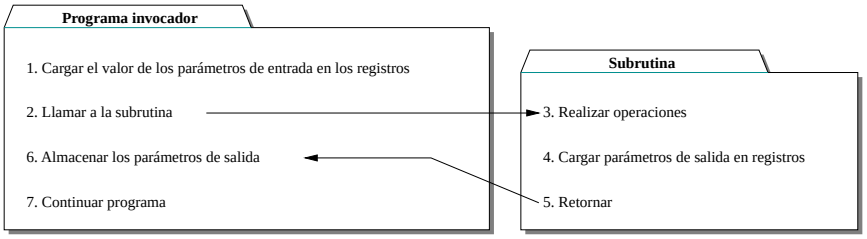

Figura 5.2: Paso de parámetros por valor

Como ejemplo de paso de parámetros por valor se presenta el siguiente código que ya se había visto con anterioridad. Éste muestra un ejemplo de paso de parámetros de entrada y de salida por valor.

introsub-suma-valor.s

```
1           .data
2 datos:    .word 5, 8, 3, 4
3 suma1:    .space 4
4 suma2:    .space 4
5           .text
```

```
6
7  @ Programa invocador
8
9  main:    ldr r4, =datos
10
11         ldr r0, [r4]
12         ldr r1, [r4, #4]
13 primera: bl suma
14         ldr r5, =suma1
15         str r2, [r5]
16
17         ldr r0, [r4, #8]
18         ldr r1, [r4, #12]
19 segunda: bl suma
20         ldr r5, =suma2
21         str r2, [r5]
22
23 stop:    wfi
24
25 @ Subrutina
26
27 suma:    add r2, r1, r0
28         mov pc, lr
29
30         .end
```

EJERCICIOS

▶ **5.7** Enumera los registros que se han utilizado para pasar los parámetros a la subrutina.

▶ **5.8** ¿Los anteriores registros se han utilizado para pasar parámetros de entrada o de salida?

▶ **5.9** Anota qué registro se ha utilizado para devolver el resultado al programa invocador.

▶ **5.10** ¿El anterior registro se ha utilizado para pasar un parámetro de entrada o de salida?

▶ **5.11** Señala qué instrucciones se corresponden a cada uno de las acciones enumeradas en la Figura 5.2. (Hazlo únicamente para la primera de las dos llamadas.)

5.2.2. Paso de parámetros por referencia

En cuanto al paso de parámetros por referencia, éste implica la siguiente secuencia de acciones (ver Figura 5.3):

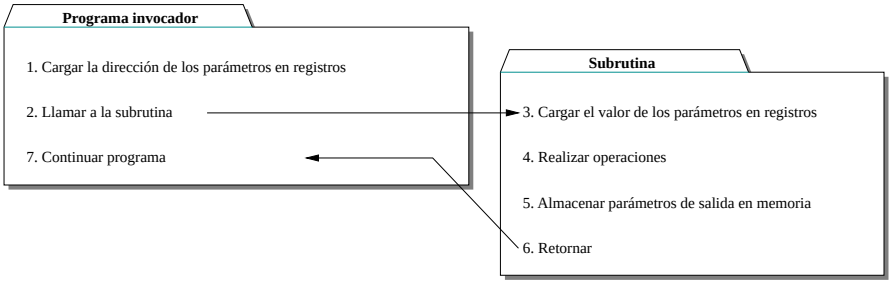

Figura 5.3: Paso de parámetros por referencia

- Antes de realizar la llamada a la subrutina, el programa invocador carga en los registros correspondientes, las direcciones de memoria en las que está almacenada la información que se quiere pasar.
- La subrutina carga en registros el contenido de las direcciones de memoria indicadas por los parámetros de entrada y opera con ellos (recuerda que ARM no puede operar directamente con datos en memoria).
- La subrutina, una vez ha finalizado y antes de devolver el control al programa principal, almacena los resultados en las direcciones de memoria proporcionadas por el programa invocador.

El siguiente programa muestra un ejemplo en el que se llama a una subrutina utilizando el paso de parámetros por referencia tanto para los parámetros de entrada como los de salida.

introsub-suma-referencia.s

```
1          .data
2  datos:  .word 5, 8, 3, 4
3  suma1:  .space 4
4  suma2:  .space 4
5          .text
6
7  @ Programa invocador
8
9  main:   ldr r0, =datos
10         ldr r1, =datos + 4
11         ldr r2, =suma1
12 primera: bl suma
13
14         ldr r0, =datos + 8
15         ldr r1, =datos + 12
16         ldr r2, =suma2
17 segunda: bl suma
18
```

```
19 stop:    wfi
20
21 @ Subrutina
22
23 suma:    ldr r4, [r0]
24          ldr r5, [r1]
25          add r6, r4, r5
26          str r6, [r2]
27          mov pc, lr
28
29          .end
```

EJERCICIOS

▶ **5.12** Enumera los registros que se han utilizado para pasar la dirección de los parámetros de entrada a la subrutina.

▶ **5.13** Anota qué registro se ha utilizado para pasar la dirección del parámetro de salida a la subrutina.

▶ **5.14** Señala qué instrucciones se corresponden con cada una de las acciones enumeradas en la Figura 5.3. (Hazlo únicamente para la primera de las dos llamadas.)

5.2.3. Paso de parámetros, ¿por valor o por referencia?

Una vez descritas las dos formas de paso de parámetros a una subrutina, queda la tarea de decidir cuál de las formas, por referencia o por valor, es más conveniente en cada caso. Los ejemplos anteriores eran un poco artificiales ya que todos los parámetros se pasaban o bien por valor o por referencia. En la práctica, se debe analizar para cada parámetro cuál es la forma idónea de realizar el paso. Es decir, generalmente, no todos los parámetros de una subrutina utilizarán la misma forma de paso.

De hecho, la decisión última de cómo se pasan los parámetros a una subrutina depende en gran medida de la estructura de datos que se quiera pasar. Si se trata de un dato de tipo estructurado (vectores, matrices, cadenas, estructuras...), el paso siempre se hará por referencia. Tanto si el parámetro en cuestión es de entrada, de salida o de entrada/salida.

En cambio, si el dato que se quiere pasar es de tipo escalar (un número entero, un número real, un carácter...), éste se puede pasar por valor o por referencia. Esta última decisión depende de la utilización que se le vaya a dar al parámetro: es decir, si se trata de un parámetro de entrada, de salida o de entrada/salida. La siguiente lista muestra las opciones disponibles según el tipo de parámetro:

- Parámetro de entrada. Un parámetro de este tipo es utilizado por la subrutina pero no debería ser modificado: es preferible pasar este tipo de parámetros por valor.
- Parámetro de salida. Un parámetro de este tipo permite a la subrutina devolver el resultado de las operaciones realizadas. Para este tipo de parámetros se puede optar por cualquiera de las opciones: que el parámetro sea devuelto por valor o por referencia.
  - Por valor: la subrutina devuelve el dato utilizando el registro reservado para ello.
  - Por referencia: la subrutina almacena en memoria el dato. La dirección de memoria la obtiene la subrutina del registro que había sido utilizado para ello.
- Parámetro de entrada/salida. Un parámetro de este tipo proporciona un valor que la subrutina necesita conocer pero en el que posteriormente devolverá el resultado. En este caso, es preferible pasarlo por referencia.

5.2.4. Un ejemplo más elaborado

A continuación se plantea el desarrollo de un programa en lenguaje ensamblador donde se aplican todos los conceptos presentados hasta ahora en este capítulo.

Dicho programa tiene por objeto calcular cuántos elementos de un vector dado tienen el valor `12`. El programa consta de una subrutina que devuelve el número de elementos de un vector que son menores a un número dado. Llamando dos veces a dicha subrutina con los valores `13` y `12` y realizando una simple resta, el programa será capaz de determinar el número de elementos que son iguales a `12`.

Se debe desarrollar dicho programa paso a paso. En primer lugar, se debe desarrollar una subrutina que contabilice cuántos elementos de un vector son menores a un valor dado. Para ello, hay que determinar qué parámetros debe recibir dicha subrutina, así como qué registros se van a utilizar para ello, y cuáles se pasarán por referencia y cuáles por valor.

Una vez desarrollada la subrutina y teniendo en cuenta los parámetros que requiere, se deberá desarrollar el programa que llame a dicha subrutina y obtenga el número de elementos del vector dado que son iguales a `12`.

Una descripción detallada del funcionamiento que debiera tener dicho programa se muestra en el siguiente listado en lenguaje Python3:

```
1 def nummenorque(vector_s, dim_s, dato_s):
2       n = 0
3       for i in range(dim_s):
```

```
4            if vector_s[i] < dato_s:
5               n = n + 1;
6       return n
7
8  vector = [5, 3, 5, 5, 8, 12, 12, 15, 12]
9  dim = 9
10 num = 12
11 res1 = nummenorque(vector, dim, num + 1)
12 res2 = nummenorque(vector, dim, num)
13 res = res1 - res2
```

EJERCICIOS

▶ **5.15** Antes de desarrollar el código de la subrutina, contesta las siguientes preguntas:

a) ¿Qué parámetros debe pasar el programa invocador a la subrutina? ¿Y la subrutina al programa invocador?

b) ¿Cuáles son de entrada y cuáles de salida?

c) ¿Cuáles se pasan por referencia y cuáles por valor?

d) ¿Qué registros se van a utilizar para cada uno de ellos?

▶ **5.16** Completa el desarrollo del fragmento de código correspondiente a la subrutina.

▶ **5.17** Desarrolla el fragmento de código correspondiente al programa invocador.

▶ **5.18** Comprueba que el programa escrito funciona correctamente. ¿Qué valor se almacena en «`res`» cuando se ejecuta? Modifica el contenido del vector «`vector`», ejecuta de nuevo el programa y comprueba que el resultado obtenido es correcto.

5.3. Problemas del capítulo

EJERCICIOS

▶ **5.19** Realiza el siguiente ejercicio:

a) Desarrolla una subrutina que calcule cuántos elementos de un vector de enteros son pares (múltiplos de 2). La subrutina debe recibir como parámetros el vector y su dimensión y devolver el número de elementos pares.

b) Implementa un programa en el que se inicialicen dos vectores de 10 elementos cada uno, «`vector1`» y «`vector2`», y que almacene en sendas variables, «`numpares1`» y «`numpares2`», el número de elementos pares de cada uno de ellos. Naturalmente, el programa deberá utilizar la subrutina desarrollada en el apartado a) de este ejercicio.

▶ **5.20** Realiza el siguiente ejercicio:

a) Implementa una subrutina que calcule la longitud de una cadena de caracteres que finalice con el carácter nulo.

b) Implementa un programa en el que se inicialicen dos cadenas de caracteres y calcule cuál de las dos cadenas es más larga. Utiliza la subrutina desarrollada en el apartado a) de este ejercicio.

▶ **5.21** Realiza el siguiente ejercicio:

a) Desarrolla una subrutina que sume los elementos de un vector de enteros de cualquier dimensión.

b) Desarrolla un programa que sume todos los elementos de una matriz de dimensión $m \times n$. Utiliza la subrutina desarrollada en el apartado a) de este ejercicio para sumar los elementos de cada fila de la matriz.

   En la versión que se implemente de este programa utiliza una matriz con $m = 5$ y $n = 3$, es decir, de dimensión 5×3 con valores aleatorios (los que se te vayan ocurriendo sobre la marcha). Se debe tener en cuenta que la matriz debería poder tener cualquier dimensión, así que se deberá utilizar un bucle para recorrer sus filas.

Bibliografía

[Adv95] Advanced RISC Machines Ltd (ARM) (1995). *ARM 7TDMI Data Sheet*.
URL http://www.ndsretro.com/download/ARM7TDMI.pdf

[Atm11] Atmel Corporation (2011). *ATmega 128: 8-bit Atmel Microcontroller with 128 Kbytes in-System Programmable Flash*.
URL http://www.atmel.com/Images/doc2467.pdf

[Atm12] Atmel Corporation (2012). *AT91SAM ARM-based Flash MCU datasheet*.
URL http://www.atmel.com/Images/doc11057.pdf

[Bar14] S. Barrachina Mir, G. León Navarro y J. V. Martí Avilés (2014). *Conceptos elementales de computadores*.
URL http://lorca.act.uji.es/docs/conceptos_elementales_de_computadores.pdf

[Cle14] A. Clements (2014). *Computer Organization and Architecture. Themes and Variations. International edition.* Editorial Cengage Learning. ISBN 978-1-111-98708-4.

[Shi13] S. Shiva (2013). *Computer Organization, Design, and Architecture, Fifth Edition*. Taylor & Francis. ISBN 9781466585546.
URL http://books.google.es/books?id=m5KlAgAAQBAJ