Fundamentos de Programación

Piensa en C

Osvaldo Cairó

PEARSON
Prentice Hall

Fundamentos de programación. Piensa en C

Fundamentos de programación. Piensa en C

Osvaldo Cairó Battistutti
Profesor-Investigador del
Instituto Tecnológico Autónomo de México (ITAM)
Director del Laboratorio KAMET
Miembro del Sistema Nacional de Investigadores (SNI), Nivel 1

REVISIÓN TÉCNICA:

M. en C. Fabiola Ocampo Botello
Escuela Superior de Computación
Instituto Politécnico Nacional

PEARSON Educación ®

México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

Datos de catalogación bibliográfica

CAIRÓ, OSVALDO
Fundamentos de programación. Piensa en C

PEARSON EDUCACIÓN,
México, 2006

ISBN: 970-26-0810-4
Área: Computación

Formato: 18.5 × 23.5 cm Páginas: 392

Editor: Pablo Miguel Guerrero Rosas
e-mail: pablo.guerrero@pearsoned.com
Editor de desarrollo: Miguel B. Gutiérrez Hernández
Supervisor de producción: Rodrigo Romero Villalobos
Diseño de portada: Lisandro P. Lazzaroni Battistutti

PRIMERA EDICIÓN, 2006

D.R. © 2006 por Pearson Educación de México, S.A. de C.V.
Atlacomulco 500-5o. Piso
Industrial Atoto
53519, Naucalpan de Juárez, Edo. de México
E-mail: editorial.universidades@pearsoned.com

Cámara Nacional de la Industria Editorial Mexicana. Reg. Núm. 1031

Prentice-Hall es una marca registrada de Pearson Educación de México, S.A. de C.V.

Reservados todos los derechos. Ni la totalidad ni parte de esta publicación pueden reproducirse, registrarse o transmitirse, por un sistema de recuperación de información, en ninguna forma ni por ningún medio, sea electrónico, mecánico, fotoquímico, magnético o electroóptico, por fotocopia, grabación o cualquier otro, sin permiso previo por escrito del editor.

El préstamo, alquiler o cualquier otra forma de cesión de uso de este ejemplar requerirá también la autorización por escrito del editor o de sus representantes.

ISBN 970-26-0810-4

Impreso en México. *Printed in Mexico.*

1 2 3 4 5 6 7 8 9 0 - 09 08 07 06

CONTENIDO

Notas del autor **xi**

Presentación **xiii**

Agradecimientos **xv**

Capítulo 1 Algoritmos, diagramas de flujo y programas en C **1**

1.1 Problemas y algoritmos 1
1.2 Diagramas de flujo 5
1.2.1. Reglas para la construcción de diagramas de flujo 7
1.3 Tipos de datos 8
1.3.1. Identificadores 9
1.3.2. Constantes 9
1.3.3. Variables 10
1.4 Operadores 12
1.4.1. Operadores aritméticos 12
1.4.2. Operadores aritméticos simplificados 13
1.4.3. Operadores de incremento y decremento 14
1.4.4. Expresiones lógicas 15
1.4.5. Operadores relacionales 15
1.4.6. Operadores lógicos 16
1.4.7. El operador coma 16
1.4.8. Prioridades de los operadores 17

1.5. Construcción de diagramas de flujo 18
1.6. Programas 22
1.6.1 Caracteres de control 23
1.6.2. Formato de variables 25
Problemas resueltos 29
Problemas suplementarios 40

Capítulo 2 Estructuras algorítmicas selectivas 49

2.1. Introducción 49
2.2. La estructura selectiva simple `if` 50
2.3. La estructura selectiva doble `if-else` 54
2.4. La estructura selectiva múltiple `switch` 58
2.5. Estructuras selectivas en cascada 64
Problemas resueltos 69
Problemas suplementarios 84

Capítulo 3 Estructuras algorítmicas repetitivas 89

3.1. Introducción 89
3.2. La estructura repetitiva `for` 90
3.3. La estructura repetitiva `while` 97
3.4. La estructura repetitiva `do-while` 102
Problemas resueltos 109
Problemas suplementarios 128

Capítulo 4 Funciones 137

4.1. Introducción 137
4.2. Variables locales, globales y estáticas 138
4.2.1. Conflicto entre los nombres de las variables 143
4.3. Parámetros por valor y por referencia 146
4.4. Paso de funciones como parámetros 152
Problemas resueltos 153
Problemas suplementarios 168

Capítulo 5 Arreglos unidimensionales 175

5.1. Introducción 175
5.2. Arreglos unidimensionales 176
5.3. Declaración de arreglos unidimensionales 177
5.4. Apuntadores y arreglos 181
5.5. Arreglos y funciones 187
Problemas resueltos 190
Problemas suplementarios 209

Capítulo 6 Arreglos multidimensionales 213

6.1. Introducción 213
6.2. Arreglos bidimensionales 214
6.3. Declaración de arreglos bidimensionales 215
6.4. Arreglos de más de dos dimensiones 220
6.5. Declaración de arreglos tridimensionales 221
Problemas resueltos 225
Problemas suplementarios 247

Capítulo 7 Caracteres y cadenas de caracteres 253

7.1. Introducción 253
7.2. Caracteres 254
7.3. Cadenas de caracteres 257
7.4. Cadenas de caracteres y arreglos 266
Problemas resueltos 268
Problemas suplementarios 281

Capítulo 8 Estructuras y uniones 287

8.1. Introducción 287
8.2. Estructuras 288
8.2.1. Declaración de estructuras 289
8.2.2. Creación de sinónimos o alias 293
8.2.3. Estructuras anidadas 295
8.2.4. Estructuras con arreglos 298
8.3. Uniones 301
8.3.1. Declaración de uniones 301
Problemas resueltos 304
Problemas suplementarios 326

Capítulo 9 Archivos de datos 333

9.1. Introducción 333
9.2. Archivos de texto y método de acceso secuencial 334
9.3. Archivos de acceso directo 343
Problemas resueltos 351
Problemas suplementarios 370

A Facundo, Silvia, María y José

PRESENTACIÓN

Este libro está dedicado a todas aquellas personas que necesitan aprender a resolver problemas y plantear su solución en un lenguaje de programación, en este caso con el lenguaje *C*. Los textos no comerciales sobre este tema –cómo resolver problemas– son pocos, y los libros sobre la materia se enfocan en presentar un lenguaje de programación, algunos de manera didáctica, otros no tanto, pero no explican cómo resolver un problema. Ésta es la principal razón de este libro. Esta característica es fundamental, sobre todo desde el punto de vista académico, porque trata de enseñar, de hacer entender, de *hacer ver*, al lector, cómo resolver un problema, y luego cómo programar esa solución en un lenguaje de programación de alto nivel. En general, aprender a usar una herramienta es sencillo, la mayoría de los libros se enfoca en ello; pero saber utilizar una herramienta no resuelve el problema: saber manejar una máquina de escribir, por ejemplo, no lo hace a uno escritor.

El libro se compone de nueve capítulos. El primero explica qué es un algoritmo, cómo se construye un diagrama de flujo y cómo escribir un programa en *C*. Los dos siguientes presentan las *estructuras algorítmicas selectivas y repetitivas.* El siguiente capítulo presenta el tema de *funciones*, asociado siempre al concepto de *reducción de problemas*. Los capítulos 5 y 6 presentan los *arreglos unidimensionales* y *multidimensionales*, respectivamente. El capítulo 7 ofrece un panorama sobre los *caracteres y cadenas de caracteres*, y el 8 sobre *estructuras y uniones*. Finalmente, el último capítulo está dedicado al estudio de *archivos de datos*.

Es importante destacar que el nivel de complejidad de los temas aumenta en forma gradual; y cada uno se expone con amplitud y claridad. Para reafirmar lo aprendido se ofrece gran cantidad de ejercicios diseñados expresamente como elementos de ayuda para el análisis, razonamiento, práctica y comprensión de los conceptos analizados. Al final de cada capítulo encontrará dos secciones: una con problemas resueltos sobre el tema de estudio y otra con problemas para resolver.

AGRADECIMIENTOS

Muchas personas contribuyeron, de una forma o de otra, en la realización de este proyecto; algunos con un enfoque positivo se convirtieron en agentes fundamentales para mantener siempre la motivación y los deseos por culminarlo.

Quiero agradecer tanto a aquellos que me apoyaron como a los que no lo hicieron, y que sin saberlo me enriquecieron notablemente al hacer que me esforzara por demostrarles que los conocimientos y los pensamientos de bien son los que verdaderamente iluminan el camino correcto. Caminos hay muchos, correctos pocos.

Vaya un agradecimiento muy especial al Dr. Arturo Fernández Pérez, Rector del Instituto Tecnológico Autónomo de México, y a las autoridades de la División Académica de Ingeniería, así como del Departamento Académico de Computación del ITAM.

CAPÍTULO 1

Algoritmos, diagramas de flujo y programas en C

1.1 Problemas y algoritmos

Los humanos efectuamos cotidianamente series de pasos, procedimientos o acciones que nos permiten alcanzar algún resultado o resolver algún problema. Estas series de pasos, procedimientos o acciones, comenzamos a aplicarlas desde que empieza el día, cuando, por ejemplo, decidimos bañarnos. Posteriormente, cuando tenemos que ingerir alimentos también seguimos una serie de pasos que nos permiten alcanzar un resultado específico: *tomar el desayuno*. La historia se repite innumerables veces durante el día. En realidad todo el tiempo estamos aplicando **algoritmos para resolver problemas.**

"Formalmente definimos un algoritmo como un conjunto de pasos, procedimientos o acciones que nos permiten alcanzar un resultado o resolver un problema."

Muchas veces aplicamos el algoritmo de manera inadvertida, inconsciente o automática. Esto ocurre generalmente cuando el problema al que nos enfrentamos lo hemos resuelto con anterioridad un gran número de veces.

Supongamos, por ejemplo, que tenemos que abrir una puerta. Lo hemos hecho tantas veces que difícilmente nos tomamos la molestia de enumerar los pasos para alcanzar este objetivo. Lo hacemos de manera automática. Lo mismo ocurre cuando nos subimos a un automóvil, lustramos nuestros zapatos, hablamos por teléfono, nos vestimos, cambiamos la llanta de un automóvil o simplemente cuando tomamos un vaso con agua.

EJEMPLO 1.1 Construye un algoritmo para preparar "Chiles morita rellenos con salsa de nuez".[1]

En México no sólo se rellenan los chiles poblanos. Esta deliciosa receta emplea el chile morita seco, de sabor ahumado. Es importante utilizar el chile morita, porque es difícil encontrar sustitutos que igualen su singular sabor.

Ingredientes:

150 g de chiles morita (unos 20).
2 cucharadas de aceite.
12 dientes de ajo.
1 cebolla cortada en aros finos.
2 tazas de vinagre de vino tinto.
Sal.
10 granos de pimienta negra.
$1\,^1/_2$ cucharadas de orégano seco.
185 g de piloncillo rallado.

Relleno:

1 cucharada de aceite.
$^1/_2$ cebolla finamente picada.
2 dientes de ajo finamente picados.
$^1/_2$ taza (125 g) de tomate finamente picado.

1/4 taza (30 g) de almendras peladas y picadas.
1/4 taza (30 g) de uvas pasas sin semillas.
1 pechuga entera de pollo cocida y finamente desmenuzada.
1 cucharadita de sal.
1/2cucharada de pimienta recién molida.

Salsa:

2 huevos, separadas las claras de las yemas.
3/4 taza (90 g) de harina.
Aceite para freír.
3/4 taza (90 g) de nueces.
1 taza de crema de leche espesa, no azucarada.

Algoritmo (*preparación*):

- Lava los chiles y sécalos bien. Calienta el aceite en una sartén grande y saltea los chiles, los ajos y la cebolla.
- Añade el vinagre, la sal, los granos de pimienta, el orégano y el piloncillo, y continúa salteando durante 10 minutos. Retira del fuego, deja que se enfríe la mezcla y ponla en una cazuela, preferentemente de barro, tapada. Refrigera 24 horas.
- Para preparar el relleno, calienta el aceite en una sartén y saltea la cebolla durante cinco minutos o hasta que esté transparente. Agrega los ajos, el tomate, las pasas, las almendras y dos cucharadas del vinagre en el que se cocieron los chiles. Mezcla bien y añade el pollo, la sal y la pimienta. Cuece a fuego lento durante ocho minutos, sin dejar de mover. Reserva. Muele el ajo, la pimienta y un poco de sal y úntaselos a las pechugas.
- Con unos guantes (para evitar que se irrite la piel) corta cada chile a lo largo. Quítales las semillas y desvénalos. Pon el relleno a cada chile con una cucharita. No pongas mucho para evitar que se desparrame al freír los chiles.
- Bate las claras al punto de turrón (de nieve). Agrega una a una las yemas sin agitar demasiado (para evitar que las claras pierdan volumen).
- En una sartén grande, calienta entre 2 y 3 cm de aceite y déjalo al fuego hasta que esté muy caliente. Pon la harina en un plato y revuelca en ella cada chile hasta que esté cubierto; sumérgelo en el huevo batido e inmediatamente ponlo en el aceite. Fríe cada chile hasta que se dore por un lado y luego dale vuelta para que se dore el otro lado.
- En un procesador de alimentos o similar, haz un puré con las nueces y la crema con una pizca de sal. Sirve los chiles con un poco de la crema de nuez encima de ellos (el resto se presenta en una salsera).

[1] Receta veracruzana de Susana Palazuelos. Para obtener más información sobre ésta y otras recetas, consulte: *El gran libro de la cocina mexicana. Recetas de Susana Palazuelos*. Editorial Patria, 1999, ISBN: 968-39-0758-X.

En la figura 1.1 podemos observar las etapas que debemos seguir para solucionar algún problema.

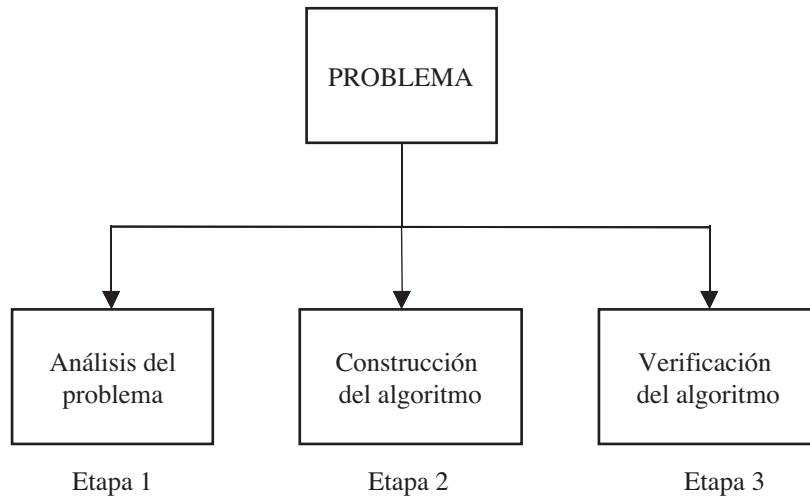


FIGURA 1.1
Etapas para solucionar un problema.

Por otra parte, las características que deben tener los algoritmos son las siguientes:

Precisión: Los pasos a seguir en el algoritmo se deben *precisar* claramente.

Determinismo: El algoritmo, dado un conjunto de datos de entrada idéntico, siempre debe arrojar los mismos resultados.

Finitud: El algoritmo, independientemente de la complejidad del mismo, siempre debe tener longitud finita.

El algoritmo consta de tres secciones o módulos principales (figura 1.2).

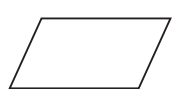
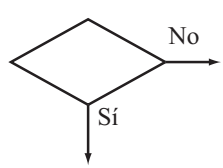
FIGURA 1.2
Módulos o secciones de un algoritmo.

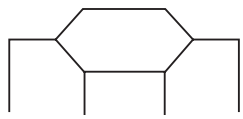
1.2 Diagramas de flujo

El diagrama de flujo representa la esquematización gráfica de un algoritmo. En realidad muestra gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema. La construcción correcta del mismo es muy importante, ya que a partir de éste se escribe el programa en un lenguaje de programación determinado. En este caso utilizaremos el lenguaje **C**, aunque cabe recordar que el diagrama de flujo se debe construir de manera independiente al lenguaje de programación. El diagrama de flujo representa la solución del problema. El programa representa la implementación en un lenguaje de programación.

A continuación, en la tabla 1.1 se presentan los símbolos que se utilizarán, junto con una explicación de los mismos. Éstos satisfacen las recomendaciones de la *International Organization for Standardization* (ISO) y el *American National Standards Institute* (ANSI).

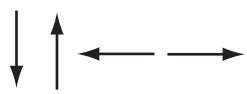
TABLA 1.1. Símbolos utilizados en los diagramas de flujo

Representación del símbolo	*Explicación del símbolo*
	Se utiliza para marcar el *inicio* y el *fin* del diagrama de flujo.
	Se utiliza para introducir los datos de entrada. Expresa *lectura*.
	Representa un *proceso*. En su interior se colocan asignaciones, operaciones aritméticas, cambios de valor de celdas en memoria, etc.
	Se utiliza para representar una *decisión*. En su interior se almacena una condición, y, dependiendo del resultado, se sigue por una de las ramas o caminos alternativos. Este símbolo se utiliza con pequeñas variaciones en las estructuras selectivas `if` e `if-else` que estudiaremos en el siguiente capítulo, así como en las estructuras repetitivas `for`, `while` y `do-while`, que analizaremos en el capítulo 3.



Se utiliza para representar una decisión múltiple, `switch`, que analizaremos en el siguiente capítulo. En su interior se almacena un selector, y, dependiendo del valor de dicho selector, se sigue por una de las ramas o caminos alternativos.

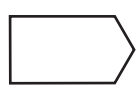
Se utiliza para representar la impresión de un resultado. Expresa *escritura*.



Expresan la dirección del flujo del diagrama.

Expresa conexión dentro de una misma página.

Representa conexión entre páginas diferentes.



Se utiliza para expresar un módulo de un problema, subproblema, que hay que resolver antes de continuar con el flujo normal del diagrama.

A continuación, en la figura 1.3 se presentan los pasos que se deben seguir en la construcción de un diagrama de flujo. El procesamiento de los datos generalmente está relacionado con el proceso de toma de decisiones. Además, es muy común repetir un conjunto de pasos.

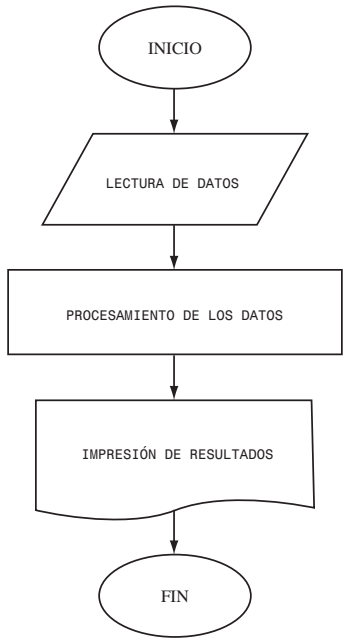


FIGURA 1.3
Etapas en la construcción de un diagrama de flujo

1.2.1. Reglas para la construcción de diagramas de flujo

El diagrama de flujo debe ilustrar gráficamente los pasos o procesos que se deben seguir para alcanzar la solución de un problema. Los símbolos presentados, colocados en los lugares adecuados, permiten crear una estructura gráfica flexible que ilustra los pasos a seguir para alcanzar un resultado específico. El diagrama de flujo facilita entonces la escritura del programa en un lenguaje de programación, **C** en este caso. A continuación se presenta el conjunto de reglas para la construcción de diagramas de flujo:

1. Todo diagrama de flujo debe tener un **inicio** y un **fin**.
2. Las líneas utilizadas para indicar la dirección del flujo del diagrama deben ser rectas: verticales u horizontales.
3. Todas las líneas utilizadas para indicar la dirección del flujo del diagrama deben estar conectadas. La conexión puede ser a un símbolo que exprese lectura, proceso, decisión, impresión, conexión o fin del diagrama.

4. El diagrama de flujo debe construirse de arriba hacia abajo (*top-down*) y de izquierda a derecha (*right to left*).
5. La notación utilizada en el diagrama de flujo debe ser independiente del lenguaje de programación. La solución presentada se puede escribir posteriormente en diferentes lenguajes de programación.
6. Al realizar una tarea compleja, es conveniente poner comentarios que expresen o ayuden a entender lo que hayamos hecho.
7. Si la construcción del diagrama de flujo requiriera más de una hoja, debemos utilizar los conectores adecuados y enumerar las páginas correspondientes.
8. No puede llegar más de una línea a un símbolo determinado.

1.3 Tipos de datos

Los datos que procesa una computadora se clasifican en **simples** y **estructurados**. La principal característica de los tipos de datos simples es que ocupan sólo una casilla de memoria. Dentro de este grupo de datos se encuentran principalmente los **enteros**, los **reales** y los **caracteres**.

TABLA 1.2. Tipos de datos simples

Tipo de datos en C	*Descripción*	*Rango*
`int`	Enteros	-32,768 a +32,767
`float`	Reales	3.4×10^{-38} a 3.4×10^{38}
`long`	Enteros de largo alcance	-2'147,483,648 a 2'147,483,647
`double`	Reales de doble precisión	1.7×10^{-308} a 1.7×10^{308}
`char`	caracter	Símbolos del abecedario, números o símbolos especiales, que van encerrados entre comillas.

Por otra parte, los datos estructurados se caracterizan por el hecho de que con un nombre se hace referencia a un grupo de casillas de memoria. Es decir, un dato estructurado tiene varios componentes. Los **arreglos**, **cadena de caracteres** y

registros representan los datos estructurados más conocidos. Éstos se estudiarán a partir del capítulo 4.

1.3.1. Identificadores

Los datos que procesará una computadora, ya sean simples o estructurados, se deben almacenar en casillas o celdas de memoria para utilizarlos posteriormente. A estas casillas o celdas de memoria se les asigna un nombre para reconocerlas: un **identificador**, el cual se forma por medio de letras, dígitos y el caracter de subrayado (_). Siempre hay que comenzar con una letra. El lenguaje de programación **C** distingue entre minúsculas y mayúsculas, por lo tanto `AUX` y `Aux` son dos identificadores diferentes. La longitud más común de un identificador es de tres caracteres, y generalmente no excede los siete caracteres. En **C,** dependiendo del compilador que se utilice, es posible generar identificadores más grandes (con más caracteres).

Cabe destacar que hay nombres que no se pueden utilizar por ser palabras reservadas del lenguaje **C**. Estos nombres prohibidos se presentan en la siguiente tabla.

TABLA 1.3. Palabras reservadas del lenguaje C

`auto`	`do`	`goto`	`signed`	`unsigned`
`break`	`double`	`if`	`sizeof`	`void`
`case`	`else`	`int`	`static`	`volatile`
`char`	`enum`	`long`	`struct`	`while`
`const`	`extern`	`register`	`switch`	
`continue`	`float`	`return`	`typedef`	
`default`	`for`	`short`	`union`	

1.3.2. Constantes

Las **constantes** son datos que no cambian durante la ejecución del programa. Para nombrar las constantes utilizamos identificadores. Existen tipos de constantes de todos los tipos de datos, por lo tanto puede haber constantes de tipo entero, real, caracter, cadena de caracteres, etc. Las constantes se deben definir antes de comenzar el programa principal, y éstas no cambiarán su valor durante la ejecución

del mismo. Existen dos formas básicas de definir las constantes:

```
const int nu1 = 20;        /* nu1 es una constante de tipo entero. */
const int nu2 = 15;        /* nu2 es una constante de tipo entero. */
const float re1 = 2.18;    /* re1 es una constante de tipo real. */
const char ca1 = 'f';      /* ca1 es una constante de tipo caracter. */
```

Otra alternativa es la siguiente:

```
#define nu1 20;            /* nu1 es una constante de tipo entero. */
#define nu2 15;            /* nu2 es una constante de tipo entero. */
#define re1 2.18;          /* re1 es una constante de tipo real. */
#define ca1 = 'f';         /* ca1 es una constante de tipo caracter. */
```

Otra forma de nombrar constantes es utilizando el método enumerador: `enum`. Los valores en este caso se asignan de manera predeterminada en incrementos unitarios, comenzando con el cero. `enum` entonces es útil cuando queremos definir constantes con valores predeterminados. A continuación se presenta la forma como se declara un `enum`:

```
enum { va0, va1, va2, va3 };       /* define cuatro constantes enteras. */
```

Esta definición es similar a realizar lo siguiente:

```
const int va0 = 0;
const int va1 = 1;
const int va2 = 2;
const int va3 = 3;
```

1.3.3. Variables

Las **variables** son objetos que pueden cambiar su valor durante la ejecución de un programa. Para nombrar las variables también se utilizan identificadores. Al igual que en el caso de las constantes, pueden existir tipos de variables de todos los tipos de datos. Por lo general, las variables se declaran en el programa principal y en las funciones (como veremos en la sección 1.6 y en el capítulo 4, respectivamente), y pueden cambiar su valor durante la ejecución del programa. Observemos a continuación la forma como se declaran:

```
void main(void)
{
...
int va1, va2;              /* Declaración de variables de tipo entero. */
float re1, re2;            /* Declaración de variables de tipo real. */
char ca1, ca2;             /* Declaración de variables de tipo caracter. */
...
}
```

Una vez que se declaran las variables, éstas reciben un valor a través de un **bloque de asignación**. La asignación es una operación destructiva. Esto significa que si la variable tenía un valor, éste se destruye al asignar el nuevo valor. El formato de la asignación es el siguiente:

```
variable = expresión o valor;
```

Donde `expresión` puede representar el valor de una expresión aritmética, constante o variable. Observa que la instrucción finaliza con punto y coma: **;**.

Analicemos a continuación el siguiente caso, donde las variables reciben un valor a través de un bloque de asignación.

```
void main(void)
{
...
int va1, va2;
float re1, re2;
char ca1, ca2;
...
va1 = 10;          /* Asignación del valor 10 a la variable va1.*/
va2 = va1 + 15;    /* Asignación del valor 25 (expresión aritmética) a va2. */
va1 = 15;          /* La variable va1 modifica su valor. */
re1 = 3.235;       /* Asignación del valor 3.235 a la variable real re1. */
re2 = re1;         /* La variable re2 toma el valor de la variable re1. */
ca1 = 't';         /* Asignación del caracter 't' a la variable ca1. */
ca2 = '?';         /* Asignación del caracter '?' a la variable ca2. */
...
}
```

Otra forma de realizar la asignación de un valor a una variable es cuando se realiza la declaración de la misma. Observemos el siguiente caso.

```
void main(void)
{
...
int va1 = 10,   va2 = 15;
float re1= 3.25,   re2 = 6.485;
char ca1 =  't',   ca2 = 's';
...
}
```

Finalmente, es importante destacar que los nombres de las variables deben ser representativos de la función que cumplen en el programa.

1.4 Operadores

Los operadores son necesarios para realizar operaciones. Distinguimos entre operadores aritméticos, relacionales y lógicos. Analizaremos también operadores aritméticos simplificados, operadores de incremento y decremento, y el operador coma.

1.4.1. Operadores aritméticos

Los operadores aritméticos nos permiten realizar operaciones entre operandos: números, constantes o variables. El resultado de una operación aritmética siempre es un número. Dado que **C** distingue entre los tipos de operandos (`int` y `float`) que se utilizan en una operación aritmética, en la tabla 1.4 se presentan los operadores aritméticos, varios ejemplos de su uso y el resultado correspondiente para cada uno de estos casos. Es importante observarlos cuidadosamente. Considera que `x` es una variable de tipo entero (`int x`) y `v` es una variable de tipo real (`float v`).

TABLA 1.4. Operadores aritméticos

Operador aritmético	*Operación*	*Ejemplos*	*Resultados*
+	Suma	`x = 4.5 + 3;`	`x = 7`
		`v = 4.5 + 3;`	`v = 7.5`
-	Resta	`x = 4.5 - 3;`	`x = 1`
		`v = 4.5 - 3;`	`v = 1.5`
*	Multiplicación	`x = 4.5 * 3;`	`x = 12`
		`v = 4.5 * 3;`	`v = 13.5`
		`v = 4 * 3;`	`v = 12.0`
/	División	`x = 4 / 3;`	`x = 1`
		`x = 4.0 / 3.0;`	`x = 1`
		`v = 4 / 3;`	`v = 1.0`
		`v = 4.0 / 3;`	`v = 1.33`
		`v = (float) 4 / 3;`	`v = 1.33`
		`v = ((float) 5 + 3) / 6;`	`v = 1.33`
%	Módulo(residuo)	`x = 15 % 2;`	`x = 1`
		`v = (15 % 2) / 2;`	`v = 0.0`
		`v = ((float) (15 % 2)) / 2;`	`v = 0.5`

Al evaluar expresiones que contienen operadores aritméticos debemos respetar la jerarquía de los operadores y aplicarlos de izquierda a derecha. Si una expresión contiene subexpresiones entre paréntesis, éstas se evalúan primero. En la tabla 1.5 se presenta la jerarquía de los operadores aritméticos de mayor a menor en orden de importancia.

TABLA 1.5. Jerarquía de los operadores aritméticos

Operador	*Operación*
`*, /, %`	Multiplicación, división, módulo
`+, -`	Suma, resta

1.4.2. Operadores aritméticos simplificados

Un aspecto importante del lenguaje **C** es la forma como se puede **simplificar** el uso de los operadores aritméticos. En la tabla 1.6 se presentan los operadores aritméticos, la forma simplificada de su uso, ejemplos de aplicación y su correspondiente equivalencia. Considere que `x` y `y` son variables de tipo entero (`int x, y`).

TABLA 1.6. Operadores aritméticos: forma simplificada de uso

Operador aritmético	*Forma simplificada de uso*	*Ejemplos*	*Equivalencia*	*Resultados*
`+`	`+=`	`x = 6;`	`x = 6;`	`x = 6`
		`y = 4;`	`y = 4;`	`y = 4`
		`x += 5;`	`x = x + 5;`	`x = 11`
		`x += y;`	`x = x + y;`	`x = 15`
`-`	`-=`	`x = 10;`	`x = 10;`	`x = 10`
		`y = 5;`	`y = 5;`	`y = 5`
		`x - = 3;`	`x = x - 3;`	`x = 7`
		`x - = y;`	`x = x - y;`	`x = 2`
`*`	`*=`	`x = 5;`	`x = 5;`	`x = 5`
		`y = 3;`	`y = 3;`	`y = 3`
		`x *= 4;`	`x = x * 4;`	`x = 20`
		`x *= y;`	`x = x * y;`	`x = 60`

continúa

TABLA 1.6. Continuación

Operador aritmético	*Forma simplificada de uso*	*Ejemplos*	*Equivalencia*	*Resultados*

/	/=	x = 25;
		y = 3;
		x /= 3;
		x /= y;
			x = 25;
			y = 3;
			x = x / 3;
			x = x / y;
				x = 25
				y = 3
				x = 8
				x = 2

%	%=	x = 20;
		y = 3;
		x %= 12;
		x %= y;
			x = 20;
			y = 3;
			x = x % 12;
			x = x % y;
				x = 20
				y = 3
				x = 8
				x = 2

1.4.3. Operadores de incremento y decremento

Los operadores de **incremento** (`++`) y **decremento** (`--`) son propios del lenguaje **C** y su aplicación es muy importante porque simplifica y clarifica la escritura de los programas. Se pueden utilizar antes o después de la variable. Los resultados son diferentes, como se puede observar en los ejemplos de la tabla 1.7. Considera que `x` y `y` son variables de tipo entero (`int x, y`).

TABLA 1.7. Operadores de incremento y decremento

Operador	*Operación*	*Ejemplos*	*Resultados*
++	Incremento	x = 7; y = x++;	x = 7 y = 7 x = 8
		x = 7; y = ++x;	x = 7 y = 8 x = 8
--	Decremento	x = 6; y = x--;	x = 6 y = 6 x = 5
		x = 6; y = --x;	x = 6 y = 5 x = 5

1.4.4. Expresiones lógicas

Las **expresiones lógicas o booleanas**, llamadas así en honor del matemático George Boole, están constituidas por números, constantes o variables y operadores lógicos o relacionales. El valor que pueden tomar estas expresiones es `1` —en caso de ser verdaderas— o `0` —en caso de ser falsas. Se utilizan frecuentemente tanto en las estructuras selectivas como en las repetitivas. En las estructuras selectivas se emplean para seleccionar un camino determinado, dependiendo del resultado de la evaluación. En las estructuras repetitivas se usan para determinar básicamente si se continúa con el ciclo o se interrumpe el mismo.

1.4.5. Operadores relacionales

Los operadores relacionales se utilizan para comparar dos operandos, que pueden ser números, caracteres, cadenas de caracteres, constantes o variables. Las constantes o variables, a su vez, pueden ser de los tipos expresados anteriormente. A continuación, en la tabla 1.8, presentamos los operadores relacionales, ejemplos de su uso y el resultado de dichos ejemplos. Considera que `res` es una variable de tipo entero (`int res`).

TABLA 1.8. Operadores relacionales

Operador relacional	*Operación*	*Ejemplos*	*Resultados*
`= =`	Igual a	`res = 'h' == 'p';`	`res = 0`
`!=`	Diferente de	`res = 'a' != 'b';`	`res = 1`
`<`	Menor que	`res = 7 < 15;`	`res = 1`
`>`	Mayor que	`res = 22 > 11;`	`res = 1`
`<=`	Menor o igual que	`res = 15 <= 2;`	`res = 0`
`>=`	Mayor o igual que	`res = 35 >= 20;`	`res = 1`

Cabe destacar que cuando se utilizan los operadores relacionales con operandos lógicos, **falso siempre es menor a verdadero**. Veamos el siguiente caso:

```
res = (7 > 8) > (9 > 6);      /*  0 > 1 (falso)      ⇒ 0   */
```

El valor de `res` es igual a `0`.

1.4.6. Operadores lógicos

Por otra parte, los **operadores lógicos**, los cuales permiten formular condiciones complejas a partir de condiciones simples, son de conjunción (**&&**), disyunción (||) y negación (**!**). En la tabla 1.9 se presentan los operadores lógicos, ejemplos de su uso y resultados de dichos ejemplos. Considera que x y y son variables de tipo entero (**int** x, y).

TABLA 1.9. Operadores lógicos

Operador lógico	*Operación*	*Ejemplos*	*Resultados*
!	Negación	`x = (!(7 > 15)); /* (!0) ⇒ 1 */` `y = (!0);`	`x = 1` `y = 1`
&&	Conjunción	`x = (35 > 20) && (20 <= 23); /* 1 && 1 */` `y = 0 && 1;`	`x = 1` `y = 0`
||	Disyunción	`x = (35 > 20) || (20 <= 18); /* 1 || 0 */` `y = 0 || 1;`	`x = 1` `y = 1`

La tabla de verdad de estos operadores se presenta a continuación.

TABLA 1.10. Tabla de verdad de los operadores lógicos

P	**Q**	***(! P)***	***(! Q)***	***(P || Q)***	***(P && Q)***
Verdadero 1	Verdadero 1	Falso 0	Falso 0	Verdadero 1	Verdadero 1
Verdadero 1	Falso 0	Falso 0	Verdadero 1	Verdadero 1	Falso 0
Falso 0	Verdadero 1	Verdadero 1	Falso 0	Verdadero 1	Falso 0
Falso 0	Falso 0	Verdadero 1	Verdadero 1	Falso 0	Falso 0

1.4.7. El operador coma

La **coma (,)** utilizada como operador sirve para encadenar diferentes expresiones. Consideremos que las variables x, v, z y v son de tipo entero (**int** x, v, z, v). Observemos a continuación diferentes casos en la siguiente tabla.

TABLA 1.11. Usos del operador coma

Expresión	*Equivalencia*	*Resultados*
`x = (v = 3, v * 5);`	`v = 3`	`v = 3`
	`x = v * 5;`	`x = 15`
`x = (v += 5, v % 3);`	`v = v + 5;`	`v = 8`
	`x = v % 3;`	`x = 2`
`x = (y = (15 > 10), z = (2 >= y), y && z);`	`y = (15 > 10);`	`y = 1`
	`z = (2 >= y);`	`z = 1`
	`x = y && z;`	`x = 1`
`x = (y = (!(7 > 15)), z = (35 > 40) && y,`	`y = (!(7 > 15));`	`y = 1`
`(!(y && z)));`	`z = (35 > 40) && y;`	`z = 0`
	`x = (!(y && z));`	`x = 1`

1.4.8. Prioridades de los operadores

Por último, y luego de haber presentado los diferentes operadores —aritméticos, relacionales y lógicos—, se muestra la tabla de jerarquía de los mismos. Cabe destacar que en **C**, las expresiones se evalúan de izquierda a derecha, pero los operadores se aplican según su prioridad.

TABLA 1.12. Jerarquía de los diferentes operadores

Operadores	*Jerarquía*
`( )`	(mayor)
`!, ++, --`	
`*, /, %`	
`+, -`	↓
`= =, !=, <, >, <=, >=`	
`&&, ‖`	
`+=, -=, *=, /=, %=`	
`,`	(menor)

El operador ( ) es asociativo y tiene la prioridad más alta en cualquier lenguaje de programación.

1.5. Construcción de diagramas de flujo

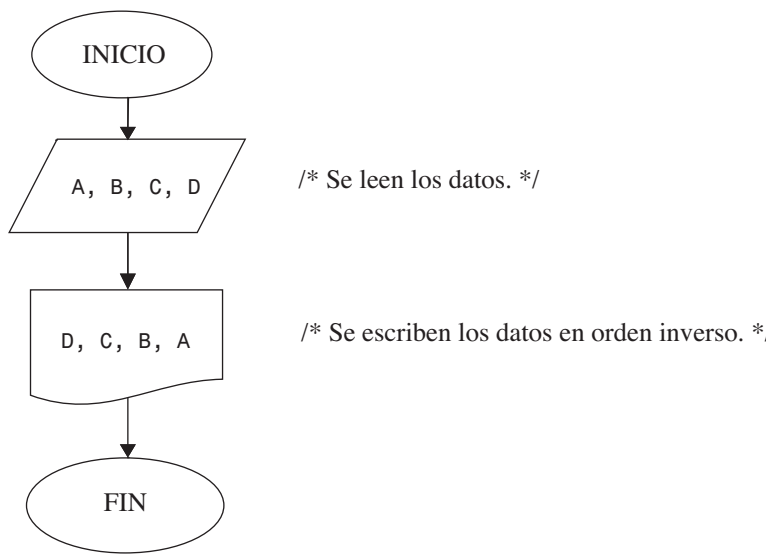
Un **diagrama de flujo** muestra, como señalamos anteriormente, la esquematización gráfica de un algoritmo. Su correcta construcción es importante, porque a partir del mismo se debe escribir el programa en un lenguaje de programación determinado. Es nuestro interés que comiences a desarrollar habilidad y una capacidad de razonamiento estructurada y flexible que te permita, en la medida que practiques, obtener la solución a los problemas planteados. A continuación se presentarán diferentes problemas y su respectiva solución por medio de diagramas de flujo.

EJEMPLO 1.2

Construye un diagrama de flujo que, al recibir los datos `A, B, C` y `D` que representan números enteros, escriba los mismos en orden inverso.

Datos: `A, B, C, D` (variables de tipo entero).

Diagrama de flujo 1.1



Observa que si se ingresan los datos: 10, 20, 30 y 40, la impresión produce lo siguiente: 40, 30, 20 y 10.

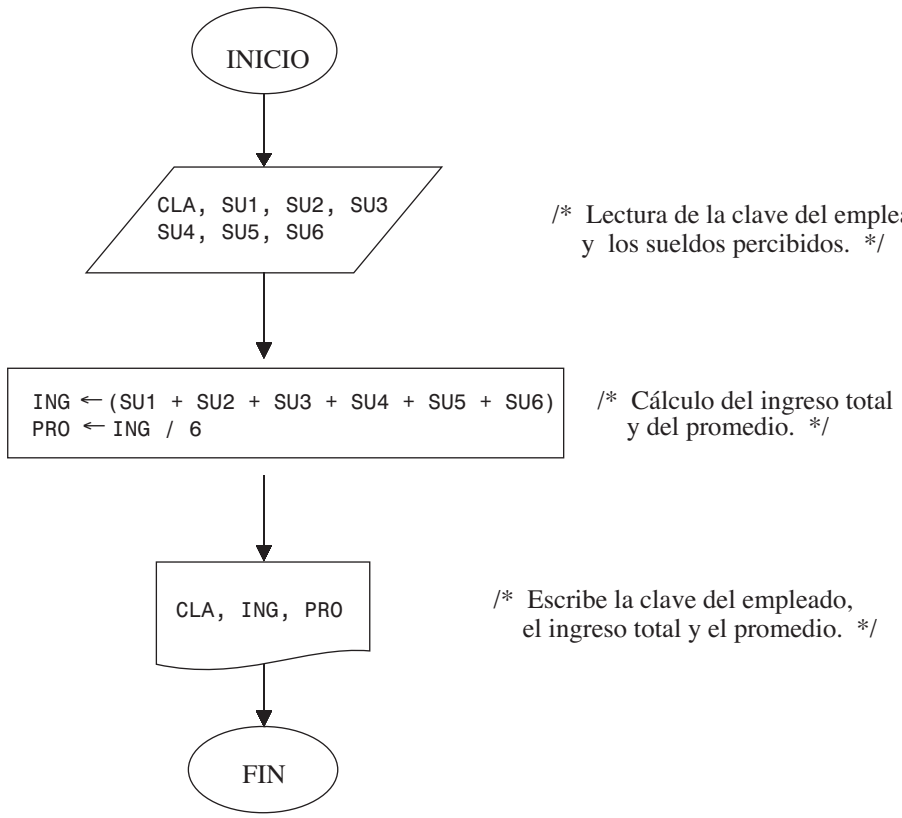
EJEMPLO 1.3

Construye un diagrama de flujo que, al recibir como datos la clave del empleado y los seis primeros sueldos del año, calcule el ingreso total semestral y el promedio mensual, e imprima la clave del empleado, el ingreso total y el promedio mensual.

Datos: `CLA, SU1, SU2, SU3, SU4, SU5, SU6`

Donde: `CLA` es una variable de tipo entero que representa la clave del empleado.
`SU1, SU2, SU3, SU4, SU5` y `SU6` son variables de tipo real que representan los seis sueldos percibidos.

Diagrama de flujo 1.2



Donde: `ING` y `PRO` son dos variables reales que almacenan el ingreso total y el promedio mensual, respectivamente.

En la tabla 1.13 puedes observar los datos y resultados para cinco corridas diferentes.

Corrida	*Datos*						*Resultados*	
	CLA	SU1	SU2	SU3	SU4	SU5	SU6	ING	PRO
1	105	12,167	14,140	13,168	12,167	21,840	12,167	85,649	14,274.83
2	105	8,750	9,745	9,745	9,745	8,750	11,190	57,925	9,654.16
3	105	21,230	18,340	19,367	19,367	18,340	22,180	118,824	19,804.00
4	105	9,645	9,645	9,645	9,800	9,645	10,280	58,660	9,776.66
5	105	11,140	10,915	12,180	15,670	11,140	12,180	73,225	12,204.16

EJEMPLO 1.4

Construye un diagrama de flujo que, al recibir como datos la base y la altura de un triángulo, calcule su superficie.

Datos: `BAS, ALT`

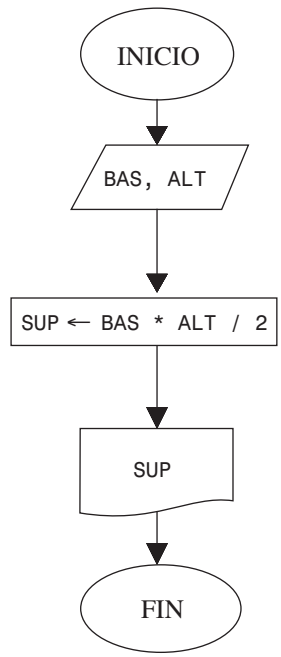
Donde: `BAS` y `ALT` son variables de tipo real que representan la base y la altura de un triángulo, respectivamente.

Recuerda que la superficie de un triángulo se calcula aplicando la siguiente fórmula:

```
Superficie = (base * altura) / 2
```

Fórmula 1.1

Diagrama de flujo 1.3



/* Lectura de datos. */

/* Cálculo de la superficie. */

/* Escritura de resultado. */

Donde: SUP es una variable de tipo real que almacena la superficie del triángulo.

En la tabla 1.14 puedes observar los datos y los respectivos resultados de cinco corridas diferentes.

TABLA 1.14.

corrida	*Datos*		*Resultado*
	BAS	ALT	SUP
1	8.5	6.2	26.35
2	7.9	13.5	60.43
3	15.18	22.0	166.98
4	12.63	7.9	49.88
5	39.40	68.5	1349.45

1.6. Programas

Un **programa**, concepto desarrollado por Von Neumann en 1946, es un conjunto de instrucciones que sigue la computadora para alcanzar un resultado específico. El programa se escribe en un **lenguaje de programación** —**C** en este caso—, a partir del diseño de un diagrama de flujo escrito con anterioridad. El lenguaje de programación está constituido por un conjunto de reglas sintácticas y semánticas. Las reglas sintácticas especifican la formación de instrucciones válidas, mientras que las semánticas especifican el significado de estas instrucciones.

C es un lenguaje de programación de tipo estructurado, que implementa por lo tanto soluciones en forma estructurada. En este enfoque la solución de los problemas se diseña de arriba hacia abajo (*top-down*), y de izquierda a derecha (*left to right*). Si la solución es correcta, el programa será fácil de entender, depurar y modificar.

La tarea intelectual, la que requiere de un pensamiento profundo, de una capacidad de razonamiento flexible y crítica, corresponde a la construcción del diagrama de flujo, que representa la solución detallada del problema. La escritura o codificación del programa, por otra parte, puede resultar una tarea sencilla si conocemos las reglas sintácticas y semánticas que constituyen el lenguaje de programación. Analicemos a continuación el primer programa escrito en el lenguaje **C**.

Programa 1.1

```
#include <stdio.h>

/* Programa 1.1
El siguiente es el primer programa escrito en el lenguaje C. */

void main (void)
{
   printf( "Mi primer programa en C" );}
```

Observa que todo programa comienza con las instrucciones que permiten incorporar las bibliotecas necesarias para correr un determinado programa en **C**. En este caso, la instrucción:

```
#include <stdio.h>
```

permite la inclusión de la biblioteca estándar `stdio` (`Standard Input Output Header`) de entrada/salida, la cual incluye las instrucciones `printf` y `scanf` necesarias para escribir y leer, respectivamente. Observa que todo lo que desees imprimir debe ir entre paréntesis ( ) y comillas " ", excepto si escribes variables, constantes o una expresión aritmética, relacional o lógica.

La siguiente instrucción del programa `/* Programa 1.1 ... */` representa la manera de escribir comentarios en el lenguaje **C**. Observa que todo comentario debe comenzar con `/*` y finalizar con `*/`.

Por otra parte, los programas se comienzan a ejecutar a partir de un determinado lugar. La instrucción:

```
void main(void)
```

indica el lugar a partir del cual se comienza a ejecutar el programa principal (`main`). El primer **`void`** indica que el programa no arrojará resultados de un tipo de datos. El segundo **`void`** especifica que el programa no tiene parámetros.

Finalmente, es importante mencionar que todas las instrucciones deben estar dentro de un bloque ({ }) y finalizar con punto y coma (;). Excepto en los casos en que las instrucciones correspondan a las estructuras selectivas, repetitivas o a nombres de funciones.

El programa 1.1 arroja el siguiente resultado:

```
Mi primer programa en C
```

1.6.1 Caracteres de control

Los **caracteres de control** producen efectos importantes en la impresión de resultados. Los diferentes caracteres de control se muestran en la siguiente tabla.

TABLA 1.15. Caracteres de control

Caracter de control	*Explicación*
`\n`	Permite pasar a una nueva línea.
`\t`	Permite tabular horizontalmente.
`\v`	Permite tabular verticalmente.
`\f`	Indica avance de página.
`\a`	Indica sonido de alerta.
`\'`	Escribe un apóstrofo.
`\"`	Escribe comillas.
`\\`	Escribe diagonal invertida.

Por ejemplo, la instrucción:

```
printf("XX \nYY \t ZZ \t RR");
```

produce el siguiente resultado:

```
XX
YY     ZZ     RR
```

y la instrucción:

```
printf("XX \tYY \n ZZ \t RR \nWW");
```

produce este otro:

```
XX     YY
ZZ      RR
WW
```

1.6.2. Formato de variables

En el lenguaje de programación **C**, el formato de lectura y escritura de las variables cambia de acuerdo con el tipo de datos que éstas puedan tener. La especificación del formato es obligatoria al escribir instrucciones de lectura (`scanf`) y escritura (`printf`). En la tabla 1.16 se presenta el formato de las variables de acuerdo con su tipo.

TABLA 1.16. Formato de escritura de las variables

Formato	*Explicación*
`%u`	Escribe enteros sin signo de 2 bytes (`unsigned int`).
`%d` `%i`	Escribe enteros de 2 bytes (`int`).
`/ld`	Imprime enteros de largo alcance (`long`).
`%f`	Escribe reales de 4 bytes (`float`).
`%lf`	Escribe reales de doble precisión, 8 bytes (`double`).
`%e`	Imprime en forma exponencial.
`%g`	Imprime en `%f` o `%e` en función del tamaño del número.
`%c`	Escribe un caracter de un byte (`char`).
`%s`	Escribe una cadena de caracteres, que termina con `'\0'`.

Por ejemplo, al definir las siguientes variables:

```
float x = 6.2555, z = 7.2576;
int y = 4, t = -5;
```

la instrucción:

```
printf(" %f  %d  %f  %d", x, y, z, t);
```

produce el siguiente resultado:

```
   6.255500   4   7.257600   -5
```

Observa que el formato de las variables va entre comillas y previo a la escritura de las mismas.

Para el mismo conjunto de variables, la siguiente instrucción:

```
printf(" %f  \n %d  \n %f  \n %d", x, y, z, t);
```

produce este otro resultado:

```
6.255500
4
7.257600
-5
```

Es importante destacar que el lenguaje **C** permite además modificaciones al símbolo `%`, con el objeto de controlar el ancho de la impresión, el número de decimales de un número real, justificar a izquierda o derecha, etc. En la siguiente tabla se presentan algunas expresiones con modificaciones en el formato y la explicación a las mismas.

TABLA 1.17. Modificaciones al símbolo %

Formato	*Explicación*
`%5d`	Escribe un entero utilizando un campo de cinco dígitos. La justificación predeterminada es a la derecha.
`%-6d`	Escribe enteros utilizando un campo de seis dígitos. La justificación es a la izquierda.
`%4.2f`	Escribe un real utilizando un campo de cuatro dígitos, dos de ellos serán utilizados para los decimales
`%-5.2f`	Escribe un real utilizando un campo de cuatro dígitos, dos de ellos serán utilizados para los decimales. La justificación es a la izquierda.

Por ejemplo, para el conjunto de variables definido anteriormente:

```
float x = 6.2555, z = 7.2576;
int y = 4, t = -5;
```

la instrucción:

```
printf("%4.2f \n %5.2e \n %5d \n %d", x, z, y, t);
```

produce el siguiente resultado:

```
6.25
7.26e+00
   4
-5
```

EJEMPLO 1.5

Observa a continuación el programa 1.2, luego de haber analizado los caracteres de control, el formato de las variables y las modificaciones al símbolo %. Cabe destacar que este programa corresponde al diagrama de flujo presentado en el ejemplo 1.2.

Programa 1.2

```
#include <stdio.h>

/*  Invierte datos
El programa, al recibir como dato un conjunto de datos de entrada, invierte el
➥orden de los mismos cuando los imprime.

A, B, C y D: variables de tipo entero. */

void main(void)
{
 int A, B, C, D;
 printf("Ingrese cuatro datos de tipo entero: ");
 scanf("%d %d %d %d", &A, &B, &C, &D);
 printf("\n %d %d %d %d ", D, C, B, A);
}
```

Observa que la instrucción de lectura `scanf` necesita del mismo formato de variables que la instrucción `printf` analizada anteriormente. La única diferencia radica en que al utilizar la instrucción de lectura se debe escribir el símbolo de dirección **&** antes de cada variable.

EJEMPLO 1.6

A continuación se presenta el programa correspondiente al diagrama de flujo del ejemplo 1.3.

Programa 1.3

```
#include <stdio.h>

/*  Promedio de sueldos.
El programa, al recibir como datos seis sueldos de un empleado, calcula tanto el
➥ingreso total como el promedio mensual.

CLA: variable de tipo entero.
SU1, SU2, SU3, SU4, SU5, SU6, ING, PRO: variables de tipo real.  */

void main (void)
{
 int CLA;
 float SU1, SU2, SU3, SU4, SU5, SU6, ING, PRO;
 printf("Ingrese la clave del empleado y los 6 sueldos: \n");
 scanf("%d %f %f %f %f %f %f", &CLA, &SU1, &SU2, &SU3, &SU4, &SU5, &SU6);
 ING = (SU1 + SU2 + SU3 + SU4 + SU5 + SU6);
 PRO = ING / 6;
 printf("\n %d  %5.2f  %5.2f", CLA, ING, PRO);
}
```

EJEMPLO 1.7

A continuación se presenta el programa correspondiente al diagrama de flujo presentado en el ejemplo 1.4.

Programa 1.4

```
 #include <stdio.h>

 /* Superficie del triángulo.
 El programa, al recibir como datos la base y la altura de un triángulo,
➥calcula su superficie.

 BAS, ALT y SUP: variables de tipo real. */

void main (void)
{
```

```
float BAS, ALT, SUP;
printf("Ingrese la base y la altura del triángulo: ");
scanf("%f %f", &BAS, &ALT);
SUP = BAS * ALT / 2;
printf("\nLa superficie del triángulo es: %5.2f", SUP);
}
```


Problemas resueltos

Problema PR1.1

Analiza cuidadosamente el siguiente programa e indica qué imprime. Si tu respuesta es correcta, felicitaciones. Si no lo es, revisa nuevamente los diferentes operadores para que refuerces lo que no hayas aprendido bien.

Programa 1.5

```
#include <stdio.h>

/* Aplicación de operadores. */

void main(void)
{
int i= 5, j = 7, k = 3, m1;
float x = 2.5, z = 1.8, t;

m1 =  ((j % k) / 2) + 1;
m1 += i;
m1 %= --i;
printf("\nEl valor de m1 es: %d", m1);

t = ((float) (j % k) / 2);
t++;
x *= ++z;
t -= (x += ++i);
printf("\nEl valor de t es: %.2f", t);
}
```

El programa genera los siguientes resultados:

```
El valor de m1 es: 2
El valor de t es: -10.50
```

Problema PR1.2

Analiza cuidadosamente el siguiente programa e indica qué imprime. Si tu respuesta es correcta, felicitaciones. Si no lo es, revisa nuevamente los diferentes operadores para que refuerces lo que no hayas aprendido bien.

Programa 1.6

```
#include <stdio.h>

/* Aplicación de operadores. */

void main(void)
{
int i = 15, j, k, l;

j = (15 > i--) > (14 < ++i);
printf("\nEl valor de j es: %d", j);

k = ! ('b' != 'd') > (!i - 1);
printf("\nEl valor de k es: %d", k);

l = (! (34 > (70 % 2)) || 0);
printf("\nEl valor de l es: %d", l);
}
```

El programa genera los siguientes resultados:

```
El valor de j es: 0
El valor de k es: 1
El valor de l es: 0
```

Problema PR1.3

Construye un diagrama de flujo que, al recibir como datos la longitud y el peso de un objeto expresados en pies y libras, imprima los datos de este objeto pero expresados en metros y kilos, respectivamente.

Datos: `PIE, LIB`

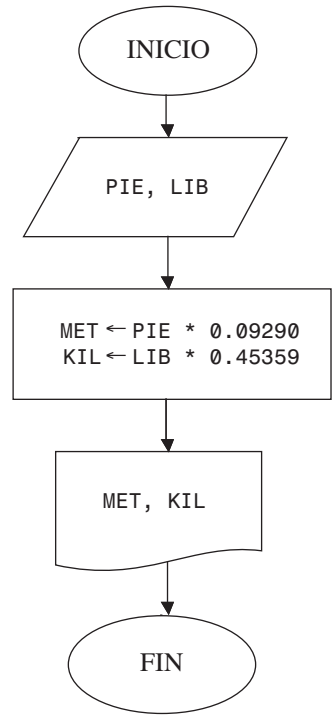
Donde: `PIE` es una variable de tipo real que representa la longitud del producto en pies.

`LIB` es una variable de tipo real que representa el peso del producto en libras.

Consideraciones:

- Un pie equivale a 0.09290 metros.
- Una libra equivale a 0.45359 kilogramos.

Diagrama de flujo 1.4



Donde: `MET` y `KIL` son variables de tipo real que almacenan los datos del objeto en metros y kilogramos, respectivamente.

A continuación se presenta el programa correspondiente.

Programa 1.7

```
   #include <stdio.h>

   /* Medidas.
    El programa, al recibir como datos la longitud y el peso de un objeto
➥expresados en pies y libras, calcula los datos de este objeto pero en
➥metros y kilogramos, respectivamente.
```

```
PIE, LIB, MET y KIL: variables de tipo real.  */

void main(void)
{
float PIE, LIB, MET, KIL;
printf("Ingrese los datos del objeto: ");
scanf("%f  %f", &PIE, &LIB);
MET = PIE * 0.09290;
KIL = LIB * 0.45359;
printf("\nDatos del objeto \nLongitud: %5.2f  \t Peso: %5.2f", MET, KIL);
}
```

Problema PR1.4

Construye un diagrama de flujo que, al recibir como datos el radio y la altura de un cilindro, calcule e imprima el área y su volumen.

Datos: `RAD, ALT`

Donde: `RAD` es una variable de tipo real que representa el radio del cilindro.
`ALT` es una variable de tipo real que representa la altura.

Consideraciones:

- El volumen de un cilindro lo calculamos aplicando la siguiente fórmula:

$$\text{Volumen} = \pi * \text{radio}^2 * \text{altura}$$

Fórmula 1.2

Donde: $\pi = 3.141592$.

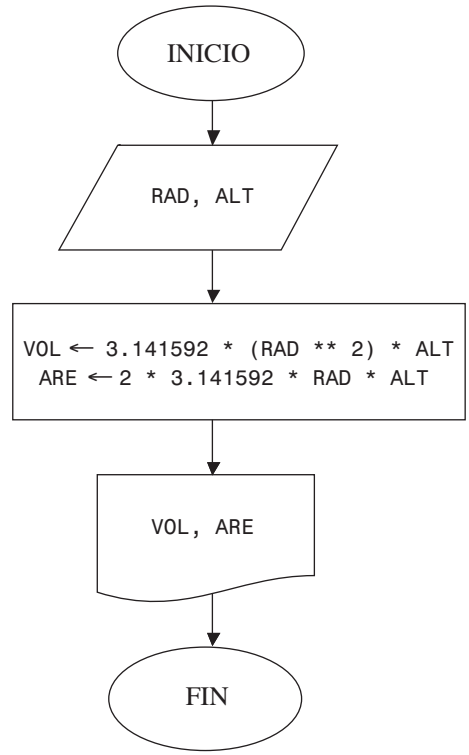
- La superficie del cilindro la calculamos como:

$$\text{Área} = 2 * \pi * \text{radio} * \text{altura}$$

Fórmula 1.3

A continuación se presenta el diagrama de flujo correspondiente.

Diagrama de flujo 1.5



Donde: VOL es una variable de tipo real que almacena el volumen del cilindro.
ARE es una variable de tipo real que almacena el área.

Programa 1.8

```
#include <stdio.h>
#include <math.h>

/*  Volumen y área del cilindro
  El programa, al recibir como datos el radio y la altura de un cilindro,
➥calcula su área y su volumen.

RAD, ALT, VOL y ARE: variables de tipo real.   */

void main(void)
{
```

```
    float RAD, ALT, VOL, ARE;
    printf("Ingrese el radio y la altura del cilindro: ");
    scanf("%f %f", &RAD, &ALT);
    /*  M_PI es una constante definida en math.h que contiene el valor de PI */
    VOL = M_PI * pow (RAD, 2) * ALT;
    ARE = 2 * M_PI * RAD * ALT;
    printf("\nEl volumen es: %6.2f  \t El área es: %6.2f", VOL, ARE);
   }
```

Observa que para realizar el programa fue necesario introducir la biblioteca `math.h`, que contiene la función `pow(x, y)` y la constante `M_PI`. En la tabla 1.18 se describen las funciones más importantes de la biblioteca `math.h`. Las variables `x` y `y` son de tipo `double` y las funciones regresan también valores de ese tipo. Cabe destacar que un error de dominio se produce si un argumento está fuera del dominio sobre el cual está definida la función.

Tabla 1.18. Funciones de la biblioteca `math.h`

Función	*Explicación*
`sin(x)`	Obtiene el seno de `x`.
`asin(x)`	Obtiene el arco seno de `x`.
`cos(x)`	Obtiene el coseno de `x`.
`acos(x)`	Obtiene el arco coseno de `x`.
`tan(x)`	Obtiene la tangente de `x`.
`atan(x)`	Obtiene el arco tangente de `x`.
`exp(x)`	Función exponencial `ex`. Eleva `e` (2.718281) a la potencia de `x`.
`fabs(x)`	Devuelve el valor absoluto de `x`.
`fmod(x1,x2)`	Obtiene el resto de `x1` entre `x2`, con el mismo signo que `x1`.
`log(x)`	Devuelve el logaritmo natural de `x, in(x), x > 0`.
`log10(x)`	Devuelve el logaritmo base 10 de `x, log10(x), x > 0`.
`pow(x, y)`	Potencia, x^y, `x > 0`, y $\geq$ `0`.
`sqrt(x)`	Obtiene la raíz cuadrada de `x, x` $\geq$ `0`.

Problema PR1.5

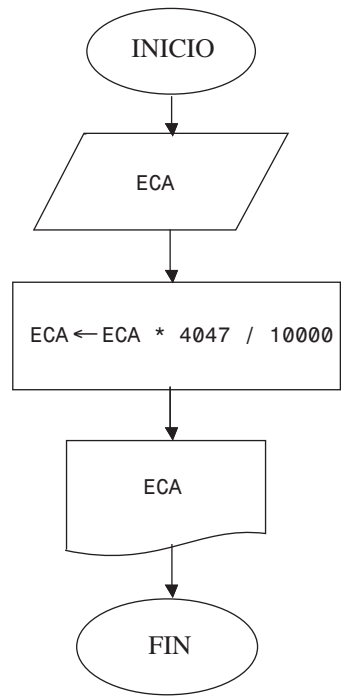
Una persona compró una estancia en un país sudamericano. La extensión de la estancia está especificada en acres. Construye un diagrama de flujo que, al recibir como dato la extensión de la estancia en *acres*, calcule e imprima la extensión de la misma en hectáreas.

Dato: `ECA` (variable de tipo real que especifica la extensión de la estancia en acres).

Consideraciones:

- 1 acre es igual a 4047 m^2.
- 1 hectárea tiene 10,000 m^2.

Diagrama de flujo 1.6



Programa 1.9

```
#include <stdio.h>

/* Estancia
 El programa, al recibir como dato la superficie de una estancia expresada
➥en acres, la convierte a hectáreas.

ECA: variable de tipo real. */

void main(void)
{
 float ECA;
 printf("Ingrese la extensión de la estancia: ");
 scanf("%f", &ECA);
 ECA = ECA * 4047 / 10000;
 printf("\nExtensión de la estancia en hectáreas: %5.2f", ECA);
}
```

Problema PR1.6

Construye un diagrama de flujo que, al recibir como datos los tres lados de un triángulo, calcule e imprima su área. Ésta se puede calcular aplicando la siguiente fórmula:

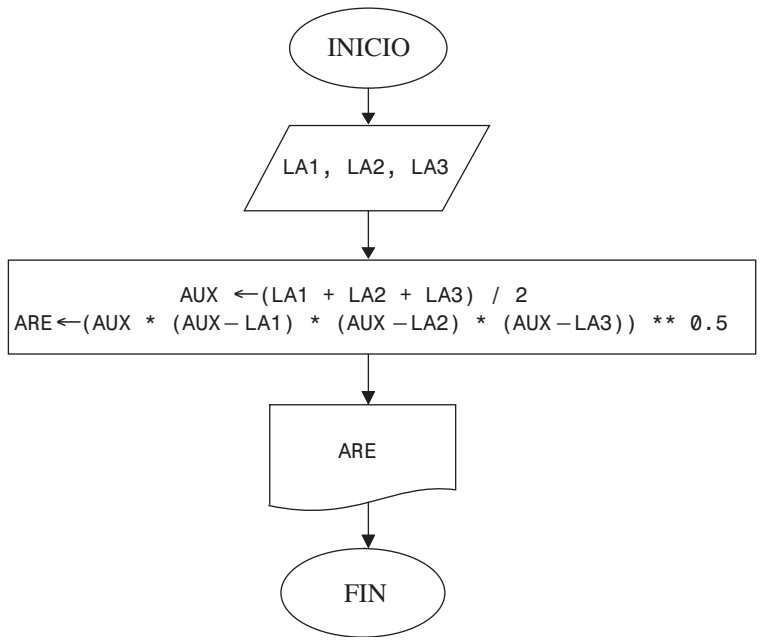
$$\text{AREA} = \sqrt{\text{AUX} * (\text{AUX} - \text{LAI}) * (\text{AUX} - \text{LA2}) * (\text{AUX} - \text{LA3})}$$

$$\text{AUX} = (\text{LA1} + \text{LA2} + \text{LA3}) / 2$$

Fórmula 1.4

Datos: `LA1`, `LA2`, `LA3` (variables de tipo real que expresan lados del triángulo).

Diagrama de flujo 1.7



Donde: AUX es una variable de tipo real, que se utiliza como auxiliar para el cálculo del área.
ARE es una variable de tipo real que almacena el área del triángulo.

Programa 1.10

```
#include <stdio.h>
#include <math.h>

/* Área del triángulo
 El programa, al recibir los tres lados de un triángulo, calcula su área.

 LA1, LA2, LA3, AUX y ARE: variables de tipo real.  */

void main(void)
{
 float LA1, LA2, LA3, AUX, ARE;
 printf("Ingrese los lados del triángulo: ");
 scanf("%f %f %f", &LA1, &LA2, &LA3);
 AUX = (LA1 + LA2 + LA3) / 2;
 ARE = sqrt (AUX * (AUX-LA1) * (AUX-LA2) * (AUX - LA3));
 printf("\nEl área del triángulo es: %6.2f", ARE);
}
```

Problema PR1.7

Construye un diagrama de flujo que, al recibir como datos las coordenadas de los puntos `P1`, `P2` y `P3` que corresponden a los vértices de un triángulo, calcule su perímetro.

Datos: `X1`, `Y1`, `X2`, `Y2`, `X3`, `Y3` (variables de tipo real que representan las coordenadas de los puntos `P1`, `P2` y `P3`).

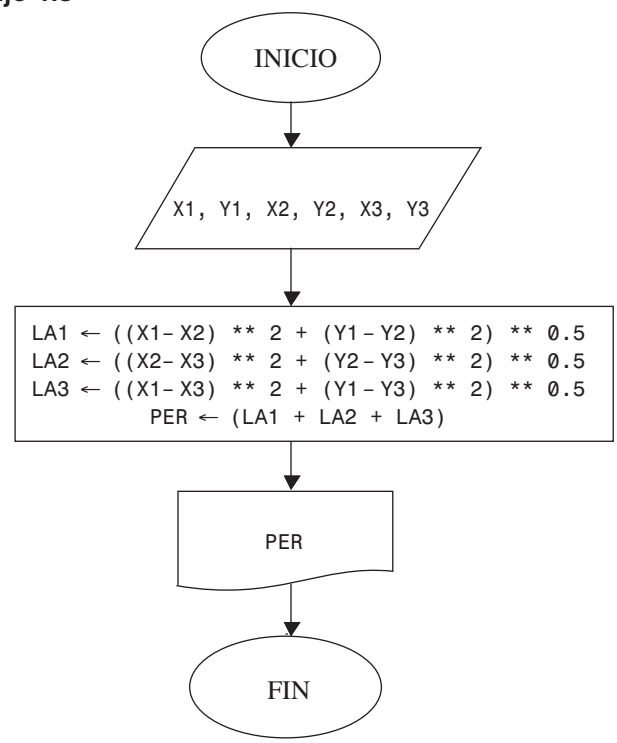
Consideraciones:

- Para calcular la distancia `DIS` entre dos puntos dados `P1` y `P2` aplicamos la siguiente fórmula:

$$\text{DIS} = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2}$$

Fórmula 1.5

Diagrama de flujo 1.8



Donde: LA1, LA2 y LA3 son variables de tipo real que se utilizan para almacenar los resultados de los lados 1, 2 y 3, respectivamente.

PER es una variable de tipo real que almacena el perímetro del triángulo.

Programa 1.11

```
#include <stdio.h>
#include <math.h>

/* Perímetro del triángulo.
El programa, al recibir las coordenadas de los puntos P1, P2 y P3 que
➥corresponden a los vértices de un triángulo, calcula su perímetro.

 X1, Y1, X2, Y2, X3, Y3, LA1, LA2, LA3 y PER: variables de tipo real. */

void main(void)
{
 float X1,Y1,X2,Y2,X3,Y3,LA1,LA2,LA3,PER;
 printf("Ingrese la coordenada del punto P1:");
 scanf("%f %f", &X1, &Y1 );
 printf("Ingrese la coordenada del punto P2:");
 scanf("%f %f", &X2, &Y2 );
 printf("Ingrese la coordenada del punto P3:");
 scanf("%f %f", &X3, &Y3 );
 LA1 = sqrt(pow(X1-X2, 2) + pow(Y1-Y2, 2));
 LA2 = sqrt(pow(X2-X3, 2) + pow(Y2-Y3, 2));
 LA3 = sqrt(pow(X1-X3, 2) + pow(Y1-Y3, 2));
 PER = LA1 + LA2 + LA3;
 printf("\nEl perímetro del triángulo es: %6.3f", PER);
}
```

Problemas suplementarios

Problema PS1.1

Analiza cuidadosamente el siguiente programa e indica qué imprime:

Programa 1.12

```
#include <stdio.h>

/* Aplicación de operadores. */

void main(void)
{
int i, j, k = 2, l = 7;

i = 9 + 3 * 2;
j = 8 % 6 + 4 * 2;
i %= j;
printf("\nEl valor de i es: %d", i);

++l;
--k -= l++ * 2;
printf("\nEl valor de k es: %d", k);

i = 5.5 - 3 * 2 % 4;
j = (i * 2 - (k = 3, --k));
printf("\nEl valor de j es: %d", j);
   }
```

Problema PS1.2

Analiza cuidadosamente el siguiente programa e indica qué imprime:

Programa 1.13

```
#include <stdio.h>

/* Aplicación de operadores. */

void main(void)
{
int i = 5, j = 4, k, l, m;
```

```
k = !i * 3 + --j * 2 - 3;
printf("\nEl valor de k es: %d", k);

l = ! (!i || 1 && 0) && 1;
printf("\nEl valor de l es: %d", l);

m = (k = (! (12 > 10)), j = (10 || 0) && k, (! (k || j)));
printf("\nEl valor de m es: %d", m);
}
```

Problema PS1.3

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos dos números reales, calcule la suma, resta y multiplicación de dichos números.

Datos: `N1`, `N2` (variables de tipo real que representan los números que se ingresan).

Problema PS1.4

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos el costo de un artículo vendido y la cantidad de dinero entregada por el cliente, calcule e imprima el cambio que se debe entregar al cliente.

Datos: `PRE`, `PAG`

Donde: `PRE` es una variable de tipo real que representa el precio del producto.
`PAG` es una variable de tipo real que representa el pago del cliente.

Problema PS1.5

Construye un diagrama de flujo y el programa correspondiente en **C**, que al recibir como dato el radio de un círculo, calcule e imprima tanto su área como la longitud de su circunferencia.

Dato: `RAD` (variable de tipo real que representa el radio del círculo).

Consideraciones:

- El área de un círculo la calculamos como:

$$\text{Área} = \pi * \text{radio}^2$$

Fórmula 1.6

- La circunferencia del círculo la calculamos de la siguiente forma:

$$\text{Circunferencia} = 2 * \pi * \text{radio}$$

Fórmula 1.7

Problema PS1.6

En una casa de cambio necesitan construir un programa tal que al dar como dato una cantidad expresada en dólares, convierta esa cantidad a pesos. Construye el diagrama de flujo y el programa correspondiente.

Dato: `CAN` (variable de tipo real que representa la cantidad en dólares).

Consideraciones:

- Toma en cuenta que el tipo de cambio actual es el siguiente: 1 dólar $\rightarrow$ 12.48 pesos.

Problema PS1.7

Escribe un programa en **C** que genere una impresión, utilizando la instrucción `printf`, como la que se muestra a continuación:

```
    XXXX
   XX
 XXX
XXX
 XXX
   XX
    XXXX
```

Problema PS1.8

En las olimpíadas de invierno el tiempo que realizan los participantes en la competencia de velocidad en pista se mide en minutos, segundos y centésimas. La distancia que recorren se expresa en metros. Construye tanto un diagrama de flujo como un programa en **C** que calcule la velocidad de los participantes en kilómetros por hora de las diferentes competencias.

Datos: `DIS`, `MIN`, `SEG`, `CEN`

Donde: `DIS` es una variable de tipo entero que indica la distancia del recorrido.
`MIN` es una variable de tipo entero que representa el número de minutos.
`SEG` es una variable de tipo entero que indica el número de segundos.
`CEN` es una variable de tipo entero que representa el número de centésimas.

Consideraciones:

- El tiempo debemos expresarlo en segundos, y para hacerlo aplicamos la siguiente fórmula:

$$\text{TSE} = \text{MIN} * 60 + \text{SEG} + \text{CEN} / 100$$

Fórmula 1.8

- Luego podemos calcular la velocidad expresada en metros sobre segundos (m/s):

$$\text{VMS} = \frac{DIS(Metros)}{TSE\ (Segundos)}$$

Fórmula 1.9

- Para obtener la velocidad en kilómetros por hora (K/h), aplicamos la siguiente fórmula:

$$\text{VKH} = \frac{VMS * 3600(Kilómetros)}{1000(hora)}$$

Fórmula 1.10

Problema PS1.9

Construye un diagrama de flujo y el correspondiente programa en **C** que calcule e imprima el número de segundos que hay en un determinado número de días.

Dato: `DIA` (variable de tipo entero que representa el número de días).

Problema PS1.10

Escribe un programa en **C** que, al recibir como dato un número de cuatro dígitos, genere una impresión como la que se muestra a continuación (el número 6352):

6
3
5
2

Problema PS1.11

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos el radio, la generatriz y la altura de un cono, calcule e imprima el área de la base, el área lateral, el área total y su volumen..

Datos: `RAD`, `ALT`, `GEN`

Donde: `RAD` es una variable de tipo real que representa el radio del cono.
`ALT` es una variable de tipo real que indica la altura.
`GEN` es una variable de tipo real que representa la generatriz.

Consideraciones:

- Un cono tiene la siguiente forma:

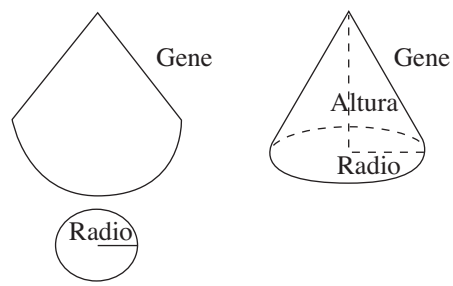


FIGURA 1.4

- El área de la base se calcula de la siguiente forma:

$$\text{Área base} = \pi * \text{radio}^2$$

Fórmula 1.11

- El área lateral se calcula así:

$$\text{Área lateral} = \pi * \text{radio} * \text{gene}$$

Fórmula 1.12

- El área total se calcula como:

$$\text{Área total} = AB + AL$$

Fórmula 1.13

- El volumen se calcula de la siguiente forma:

$$\text{Volumen} = \frac{1}{3} * AB * ALTU$$

Fórmula 1.14

Problema PS1.12

Construye un diagrama de flujo y el programa correspondiente en **C** que, al recibir como dato el radio de una esfera, calcule e imprima el área y su volumen.

Dato: RAD (variable de tipo real que representa el radio de la esfera).

Consideraciones:

- Una esfera tiene la siguiente forma:

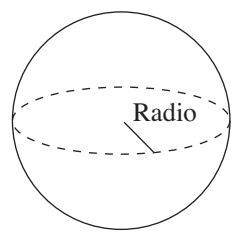


FIGURA 1.5.

- El área de una esfera la calculamos de la siguiente forma:

$$\text{Área} = 4 * \pi * \text{radio}^2$$

Fórmula 1.15

- El volumen de una esfera se calcula así:

$$\text{Volumen} = \frac{1}{3} * \pi * \text{radio}^3$$

Fórmula 1.16

Problema PS1.13

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato el lado de un hexaedro o cubo, , calcule el área de la base, el área lateral, el área total y el volumen.

Dato: LAD (variable de tipo real que representa el lado del hexaedro).

Consideraciones:

- Un hexaedro o cubo tiene la siguiente forma:

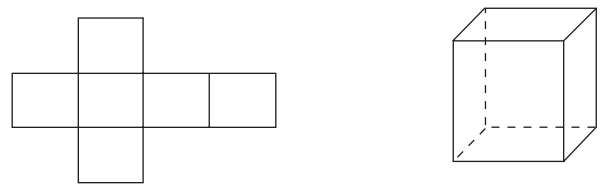


FIGURA 1.6.

- Para calcular el área de la base aplicamos la siguiente fórmula:

$$\text{Área base} = L^2$$

Fórmula 1.17

- Para calcular el área lateral utilizamos:

$$\text{Área lateral} = 4 * L^2$$

Fórmula 1.18

- Para calcular el área total utilizamos:

$$\text{Área total} = 6 * L^2$$

Fórmula 1.19

- Para calcular el volumen utilizamos:

$$\text{Volumen} = L^3$$

Fórmula 1.20

Problema PS1.14

Construye un diagrama de flujo y el respectivo programa en **C** que, al recibir como datos las coordenadas de los puntos `P1`, `P2` y `P3` que corresponden a los vértices de un triángulo, calcule su superficie.

Datos: `X1`, `Y1`, `X2`, `Y2`, `X3`, `Y3`

Donde: `X1` y `Y1` son variables de tipo real que indican las coordenadas del punto `P1`.
`X2` y `Y2` son variables de tipo real que representan las coordenadas del punto `P2`.
`X3` y `Y3` son variables de tipo real que indican las coordenadas del punto `P3`.

Consideraciones:

- Para calcular el área de un triángulo dadas las coordenadas de los vértices que la componen, podemos aplicar la siguiente fórmula:

$$\text{Área} = \frac{1}{2} * \left| X1 * (Y2 - Y3) + X2 * (Y3 - Y1) + X3 * (Y1 - Y2) \right|$$

Fórmula 1.21

CAPÍTULO 2

Estructuras algorítmicas selectivas

2.1 Introducción

Las estructuras lógicas selectivas se encuentran en la solución algorítmica de casi todo tipo de problemas. Estas estructuras se utilizan cuando se debe **tomar una decisión** en el desarrollo de la solución de un problema. La *toma de decisión* se basa en la evaluación de una o más condiciones que nos señalarán como consecuencia la rama a seguir.

Es frecuente que nos encontremos con situaciones en las que debemos tomar varias decisiones. Cuando esto ocurre, decimos que se realizan en cascada. Es decir, se toma una decisión, se señala el camino a seguir, nos encontramos con otra decisión, se marca el siguiente camino, y así sucesivamente. En estos casos prácticamente debemos construir un árbol de decisión para plantear la solución.

Las estructuras algorítmicas selectivas que estudiaremos en este capítulo son las siguientes: `if`, `if-else` y `switch`. Cabe destacar que cuando las estructuras selectivas se aplican en cascada, en realidad se utiliza una combinación de las estructuras señaladas anteriormente.

2.2 La estructura selectiva simple `if`

La estructura selectiva `if` permite que el flujo del diagrama siga por un camino específico si se cumple una condición determinada. si al evaluar la condición el resultado es verdadero, entonces se sigue por un camino específico —hacia abajo— y se ejecuta una operación o acción o un conjunto de ellas. por otra parte, si el resultado de la evaluación es falso, entonces se pasa(n) por alto esa(s) operación(es). en ambos casos se continúa con la secuencia normal del diagrama de flujo. Observemos la representación gráfica de esta estructura:

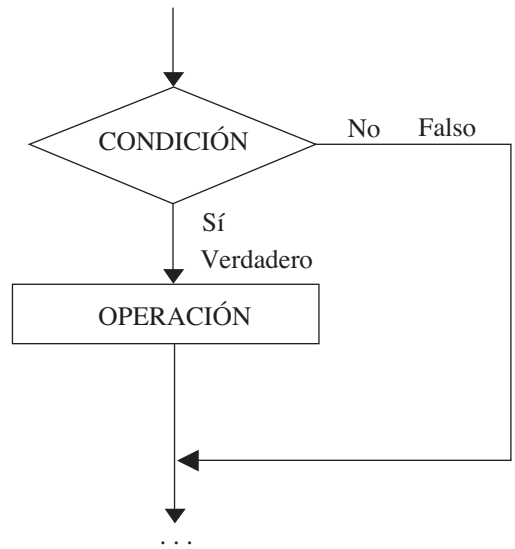


FIGURA 2.1
Estructura selectiva simple `if`

En lenguaje **C**, la estructura selectiva `if` se escribe de la siguiente forma:

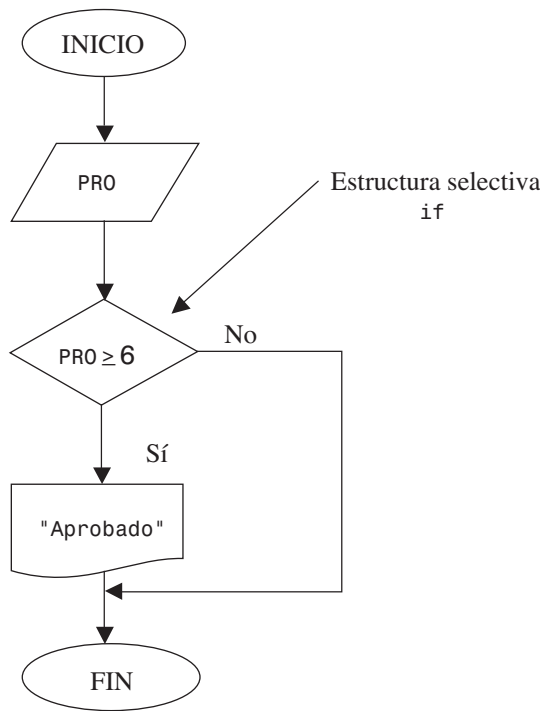
```
/* El conjunto de instrucciones muestra la sintaxis de la
➥estructura if en el lenguaje C. */
 . . .
 if  (<condición>)
        <operación>;
 . . .
```


EJEMPLO 2.1

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato el promedio de un alumno en un curso universitario, escriba `aprobado` en caso de que el promedio sea satisfactorio, es decir mayor o igual a 6.

Dato: `PRO` (variable de tipo real que representa el promedio del alumno).

Diagrama de flujo 2.1



En la tabla 2.1 podemos observar el seguimiento del diagrama de flujo para diferentes corridas.

TABLA 2.1.

Número de corrida	*Dato* PRO	*Resultado*
1	6.75	aprobado
2	5.90	
3	4.00	
4	8.80	aprobado
5	9.35	aprobado

A continuación se presenta el programa correspondiente.

Programa 2.1

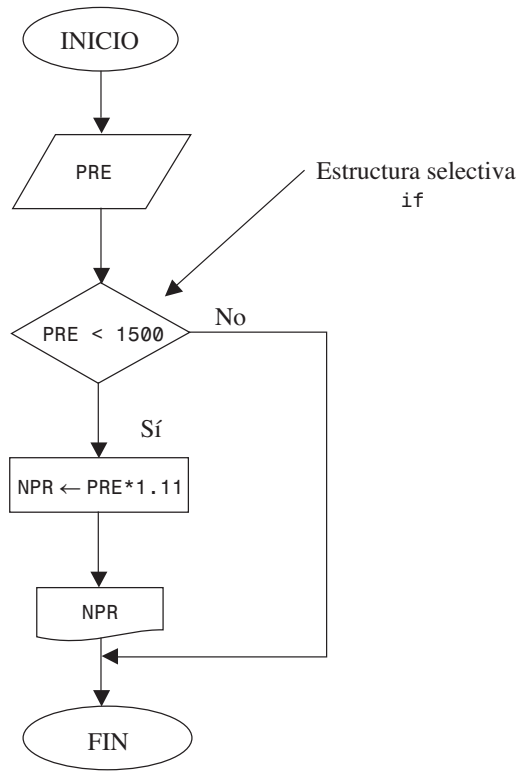
```
#include <stdio.h>

 /* Promedio curso.
 El programa, al recibir como dato el promedio de un alumno en un curso
➥universitario, escribe aprobado si su promedio es mayor o igual a 6.

 PRO: variable de tipo real. */

 void main(void)
 {
 float PRO;
 printf("ingrese el promedio del alumno: ");
 scanf("%f", &PRO);
 if (PRO >= 6)
    printf("\nAprobado");
 }
```

EJEMPLO 2.2

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato el precio de un producto importado, incremente 11% el mismo si es inferior a $1,500 y que además escriba el nuevo precio del producto.

Dato: PRE (variable de tipo real que representa el precio del producto).

Diagrama de flujo 2.2



Donde: NPR es una variable de tipo real que almacena el nuevo precio del producto.

Programa 2.2

```
#include <stdio.h>
 /* Incremento de precio.
 El programa, al recibir como dato el precio de un producto importado,
➥incrementa 11% el mismo si éste es inferior a $1,500.
 PRE y NPR: variable de tipo real. */

 void main(void)
 {
 float PRE, NPR;
 printf("ingrese el precio del producto: ");
 scanf("%f", &PRE);
```

```
if (PRE > 1500)
{
     NPR = PRE * 1.11;
     printf("\nNuevo precio: %7.2f",NPR);
}
}
```

2.3. La estructura selectiva doble `if-else`

La estructura selectiva doble **`if-else`** permite que el flujo del diagrama se bifurque por dos ramas diferentes en el punto de la toma de decisión. Si al evaluar la condición el resultado es verdadero, entonces se sigue por un camino específico —el de la izquierda— y se ejecuta una acción determinada o un conjunto de ellas. Por otra parte, si el resultado de la evaluación es falso, entonces se sigue por otro camino —el de la derecha— y se realiza(n) otra(s) acción(es). En ambos casos, luego de ejecutar las acciones correspondientes, se continúa con la secuencia normal del diagrama de flujo. Observemos la representación gráfica de esta estructura.

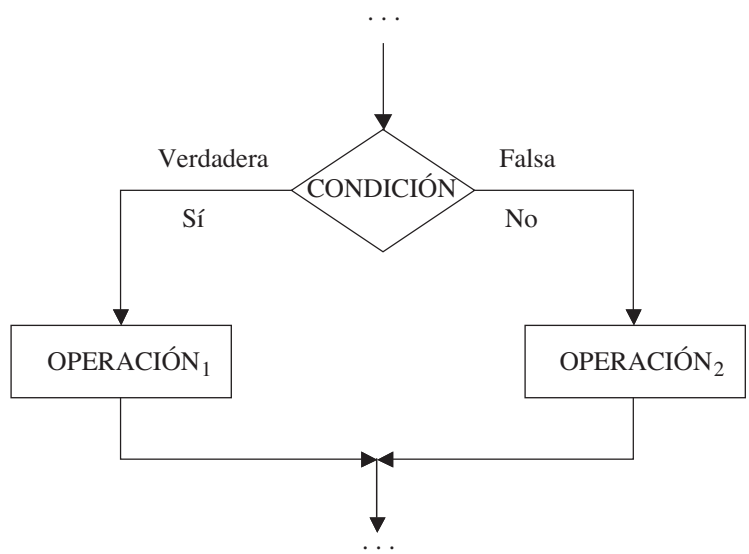


FIGURA 2.2
Estructura selectiva doble `if-else`

En el lenguaje **C** la estructura selectiva doble `if-else` se escribe de la siguiente forma:

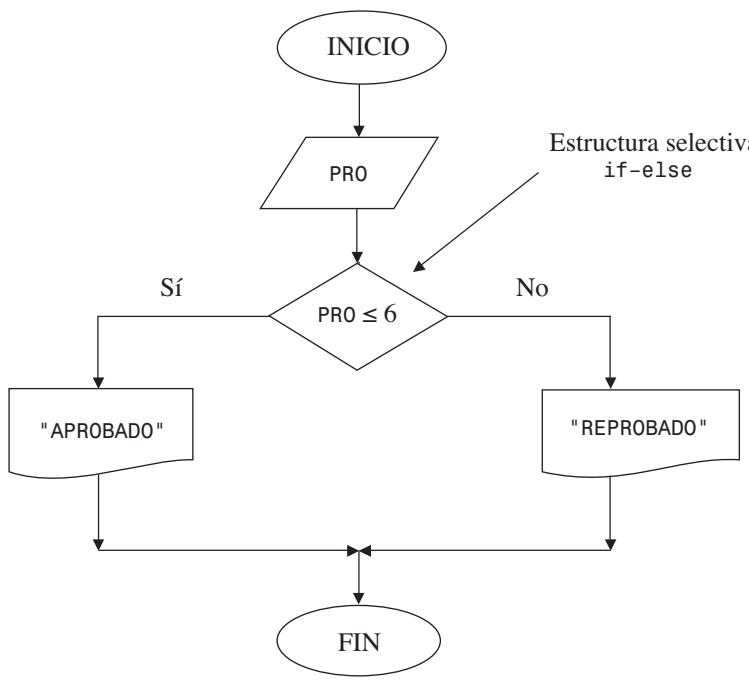
```
/* El conjunto de instrucciones muestra la sintaxis de la estructura
➥if-else en C. */
. . .
if  (<condición>)
        <operación1>;
else
        <operación2>;
. . .
```


EJEMPLO 2.3

Construye un diagrama de flujo y el programa correspondiente en **C** que, al recibir como dato el promedio de un alumno en un curso universitario, escriba `aprobado` si su promedio es mayor o igual a 6 y `reprobado` en caso contrario.

Dato: `PRO` (variable de tipo real que representa el promedio del alumno).

Diagrama de flujo 2.3



En la tabla 2.2 podemos observar el seguimiento del algoritmo para diferentes corridas.

TABLA 2.2.

Número de corrida	*Dato* PRO	*Resultado*
1	4.75	Reprobado
2	6.40	Aprobado
3	8.85	Aprobado
4	5.90	Reprobado

A continuación se presenta el programa correspondiente.

Programa 2.3

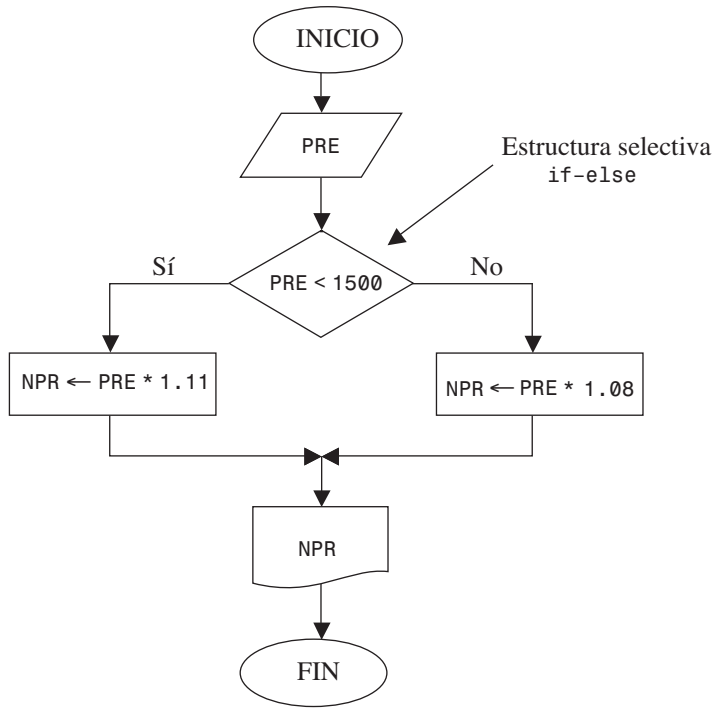
```
#include <stdio.h>

/* Promedio curso.
El programa, al recibir como dato el promedio de un alumno en un curso
➥universitario, escribe aprobado si su promedio es mayor o igual a 6, o
➥reprobado en caso contrario.

PRO: variable de tipo real.  */

 void main(void)
 {
 float PRO;
 printf("Ingrese el promedio del alumno: ");
 scanf("%f", &PRO);
 if (PRO >= 6.0)
    printf("\nAprobado");
 else
    printf("\nReprobado");
}
```

EJEMPLO 2.4

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato el precio de un producto importado, incremente 11% el mismo si es inferior a $1,500, y 8% si fuera mayor o igual a dicho precio; además, debe escribir el nuevo precio del producto..

Dato: PRE (variable de tipo real que representa el precio del producto).

Diagrama de flujo 2.4



Donde: NPR es una variable de tipo real que almacena el nuevo precio del producto.

Programa 2.4

```
#include <stdio.h>

/* incremento de precio.
El programa, al recibir como dato el precio de un producto, incrementa al
➥mismo 11% si  es menor a 1500$ y 8% en caso contrario (mayor o igual).

PRE y NPR: variables de tipo real. */

void main(void)
{
float PRE, NPR;
printf("Ingrese el precio del producto: ");
scanf("%f", &PRE);
if (PRE < 1500)
      NPR = PRE * 1.11;
else
```

```
        NPR = PRE * 1.08;
printf("\nNuevo precio del producto: %8.2f", NPR);
}
```

2.4. La estructura selectiva múltiple `switch`

La estructura selectiva `switch` permite que el flujo del diagrama se bifurque por varias ramas en el punto de la toma de decisión. La elección del camino a seguir depende del contenido de la variable conocida como `selector`, la cual puede tomar valores de un conjunto previamente establecido. El camino elegido, entonces, dependerá del valor que tome el selector. Así, si el selector toma el valor 1, se ejecutará la acción 1; si toma el valor 2, se ejecutará la acción 2, y si toma el valor N, se realizará la acción N. A continuación se presenta la figura que ilustra esta estructura selectiva.

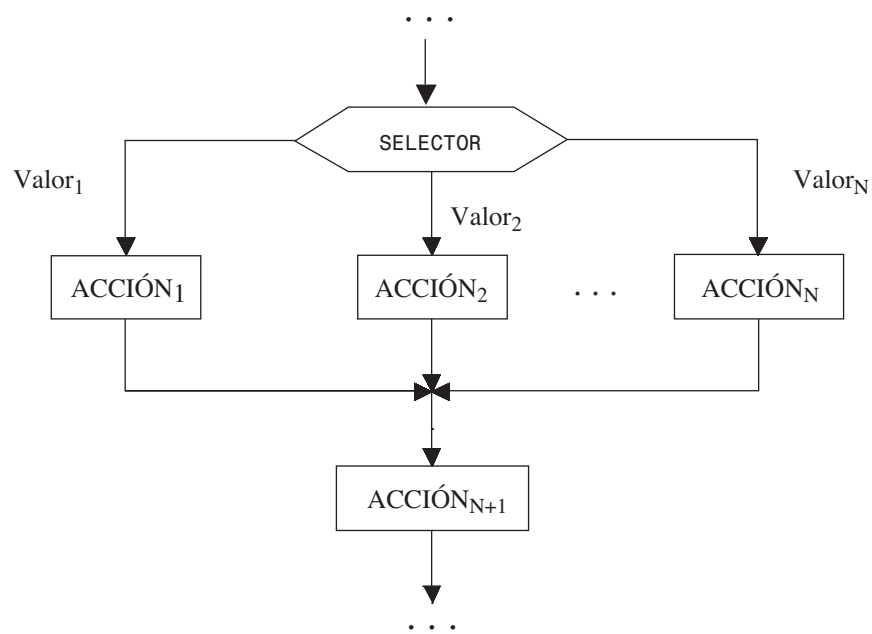


FIGURA 2.3
Estructura selectiva múltiple **`switch`**

En el lenguaje **C** la estructura selectiva múltiple **switch** se escribe de esta forma:

```
/* El conjunto de instrucciones muestra la sintaxis de la estructura switch
➥en C. */

. . .
switch(<selector>)
{
       case <valor1>    : <acción1>;
            break;      /* Es necesario escribir break para no evaluar los
                           ➥otros casos. */
       case <valor2>    : < acción2>;
            break;
       case <valorN>    : <acciónN>;
            break;
}
acciónN+1;
. . .
```


Es importante señalar que la estructura selectiva switch es muy flexible, lo que permite que se pueda aplicar de diferentes formas. Por ejemplo, si el selector tomara un valor distinto de los comprendidos entre 1 y N, entonces se debe seguir el camino etiquetado con *De otra forma*.

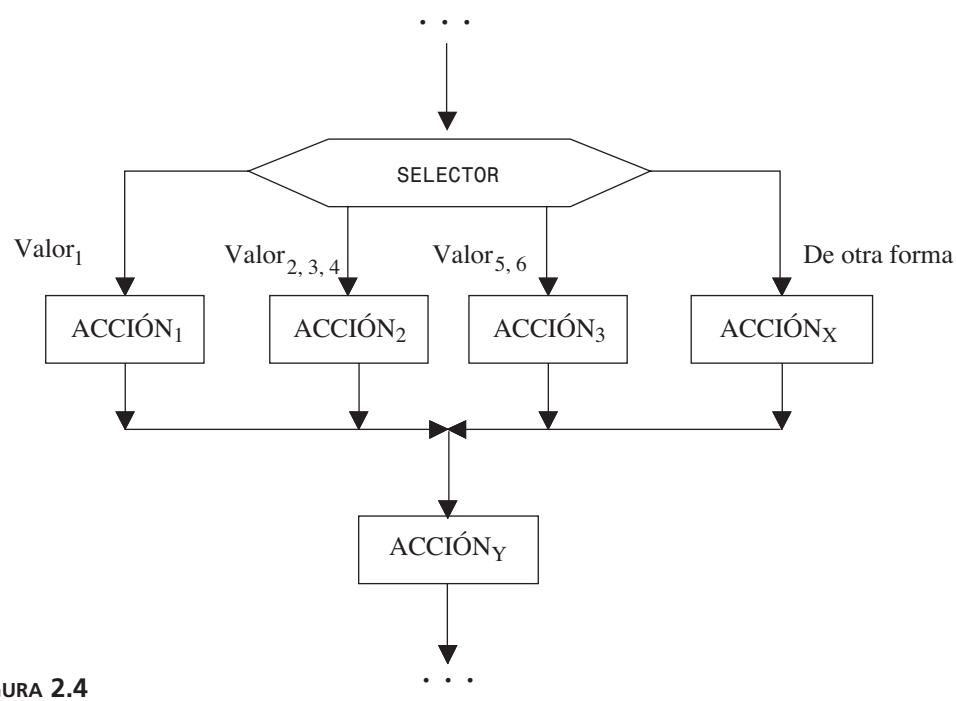


FIGURA 2.4
Estructura selectiva múltiple **switch**

En el lenguaje **C** el diagrama de flujo se escribe de la siguiente forma:

```
/* El conjunto de instrucciones muestra la sintaxis de la estructura switch
   ➥en C. */
. . .
switch(<selector>)
{
   case <valor1>   :  <acción1>;
        break;                                  /* Para salir del switch */
   case <valor2>   :
   case <valor3>   :
   case <valor4>   :  <acción2>;
        break;
   case <valor5>   :
   case <valor6>   :  <acción3>;
        break;
   default:           :  <acciónx>;
        break;
}
acciónY;
. . .
```

EJEMPLO 2.5

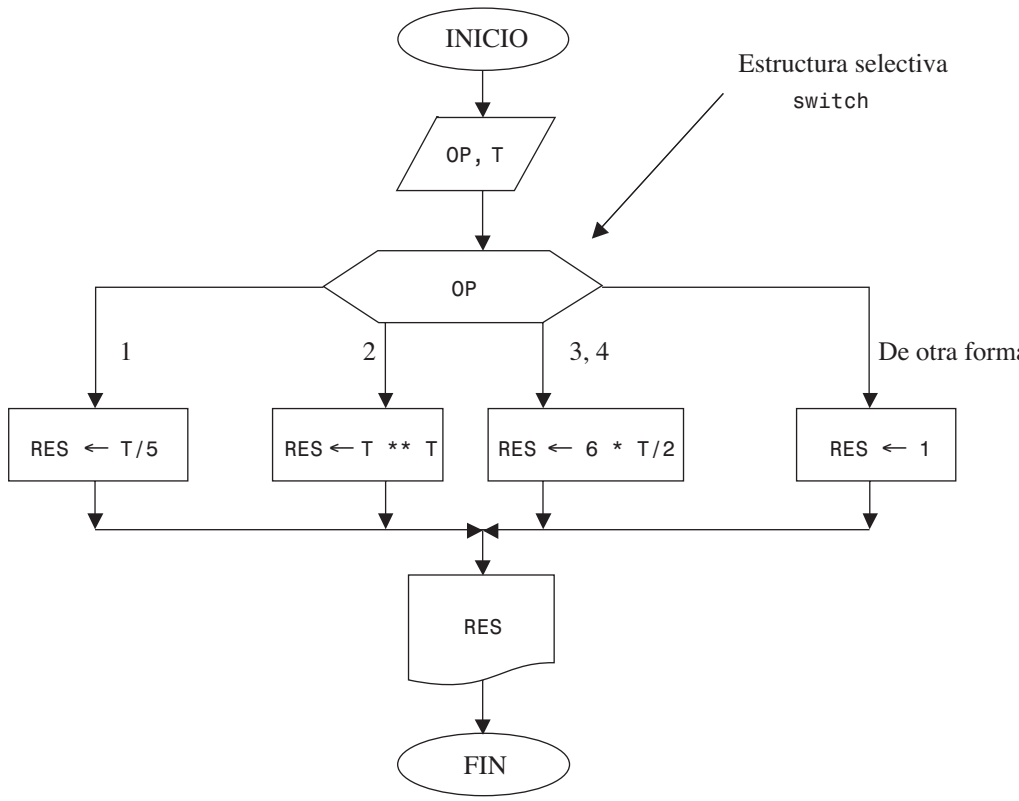
Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos dos variables de tipo entero, obtenga el resultado de la siguiente función:

$$f(T) = \begin{cases} T/5 & \text{Si OP = 1} \\ T ** T & \text{Si OP = 2} \\ 6*T/2 & \text{Si OP = 3, 4} \\ 1 & \text{Para cualquier otro caso.} \end{cases}$$

Datos: `OP` y `T`

Donde: `OP` es una variable de tipo entero que representa el cálculo a realizar.
`T` es una variable de tipo entero que se utiliza para el cálculo de la función.

Diagrama de flujo 2.5



Donde: RES es una variable de tipo real que almacena el resultado de la función.

Programa 2.5

```
#include <stdio.h>
#include <math.h>

/* Función matemática.
El programa obtiene el resultado de una función.

OP y T: variables de tipo entero.
RES: variable de tipo real. */

void main(void)
{
 int OP, T;
 float RES;
 printf("Ingrese la opción del cálculo y el valor entero: ");
```

```
   scanf("%d %d", &OP, &T);
   switch(OP)
   {
        case 1: RES = T / 5;
          break;
        case 2: RES = pow(T,T);
        /* La función pow está definida en la biblioteca math.h */
            break;
        case 3:
        case 4: RES = 6 * T/2;
            break;
        default: RES = 1;
            break;
  }
   printf("\nResultado:   %7.2f", RES);
  }
```

EJEMPLO 2.6

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos el nivel académico de un profesor de una universidad así como su salario, incremente este último siguiendo las especificaciones de la tabla 2.3 e imprima tanto el nivel del profesor como su nuevo salario.

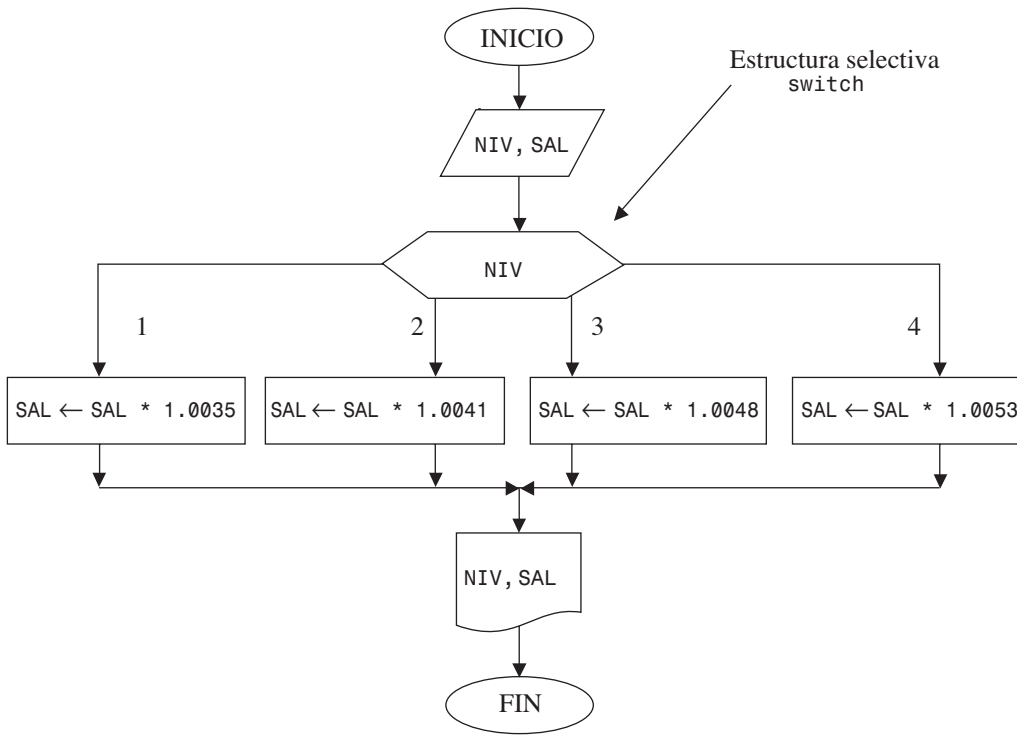
Datos: `NIV` y `SAL`

Donde: `NIV` es una variable de tipo entero que representa el nivel del profesor.
`SAL` es una variable de tipo real que representa el salario del profesor.

TABLA 2.3.

Nivel	*Incremento*
1	3.5%
2	4.1%
3	4.8%
4	5.3%

Diagrama de flujo 2.6



El programa en lenguaje **C** se escribe de esta forma:

Programa 2.6

```
  #include <stdio.h>

  /* Incremento de salario.
  El programa, al recibir como dato el nivel de un profesor, incrementa su
➥salario en función de la tabla 2.3.

  NIV: variable de tipo entero.
  SAL: variables de tipo real. */

  void main(void)
  {
   float SAL;
   int NIV;
   printf("Ingrese el nivel académico del profesor:  ");
   scanf("%d", &NIV);
```

```
 printf("Ingrese el salario:  ");
 scanf("%f", &SAL);
 switch(NIV)
 printf("ingrese el salario:  ");
 scanf("%f", &SAL);
 switch(NIV)
{
      case 1: SAL = SAL * 1.0035;  break;
      case 2: SAL = SAL * 1.0041;  break;
      case 3: SAL = SAL * 1.0048;  break;
      case 4: SAL = SAL * 1.0053;  break;
}
 printf("\n\nNivel: %d \tNuevo salario: %8.2f",NIV, SAL);
}
```

2.5. Estructuras selectivas en cascada

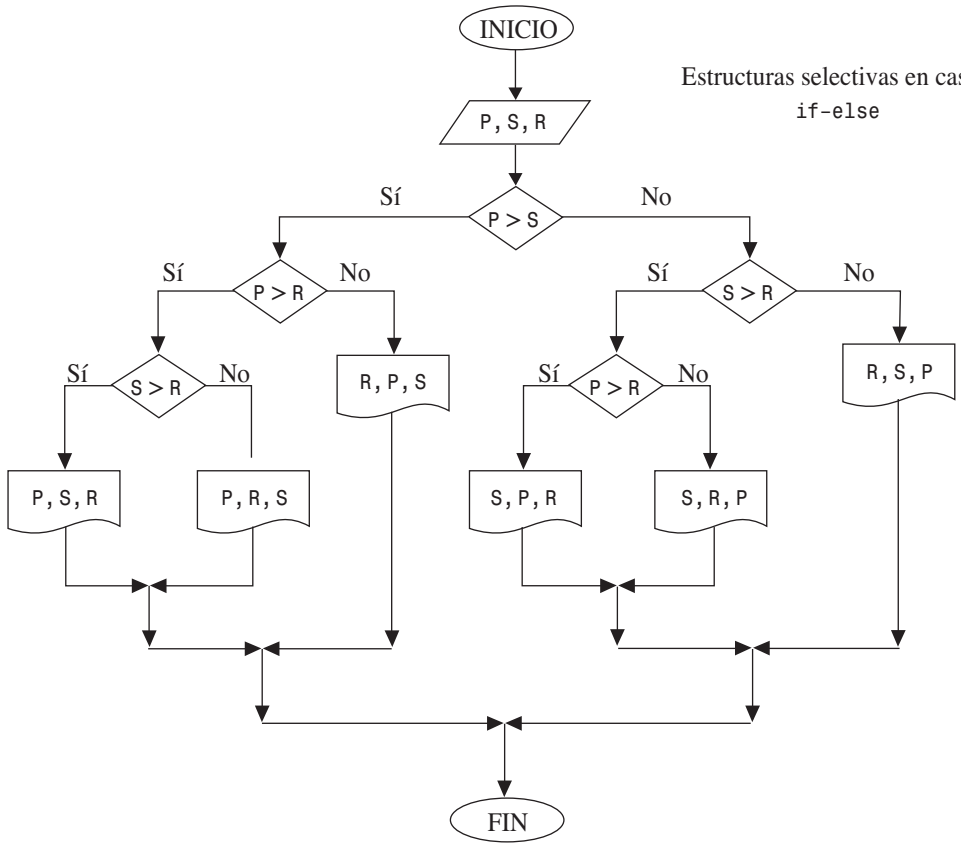
En el desarrollo de la solución de problemas se encuentran frecuentemente casos en los que, luego de tomar una decisión y señalar el correspondiente camino a seguir, es necesario tomar otra decisión. Este proceso se puede repetir numerosas veces. Una forma de solucionarlo es aplicando *estructuras selectivas en cascada*. Analicemos a continuación diferentes casos.

EJEMPLO 2.7

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos las ventas de tres vendedores de una tienda de discos, escriba las mismas en forma descendente. Considera que todas las ventas son diferentes y no utilices operadores lógicos para agrupar las condiciones.

Datos: `P`, `S` y `R` (variables de tipo real que representan las ventas de los tres vendedores).

Diagrama de flujo 2.7



A continuación se presenta el programa correspondiente.

Programa 2.7

```
#include <stdio.h>

/*  ventas descendentes.
El programa, al recibir como datos tres valores que representan las ventas
➥de los vendedores de una tienda de discos, escribe las ventas en
➥orden descendente.

P, S y R: variables de tipo real.    */
```

```
 void main(void)
{
 float P, S, R;
 printf("\nIngrese las ventas de los tres vendedores: ");
 scanf("%f %f %f", &P, &S, &R);
 if (P > S)
   if (P > R)
      if (S > R)
         printf("\n\n El orden es P, S y R: %8.2f %8.2f %8.2f", P, S, R);
      else
        printf("\n\n El orden es P, R y S: %8.2f %8.2f %8.2f", P, R, S);
   else
      printf("\n\n El orden es R, P y S: %8.2f %8.2f %8.2f", R, P, S);
 else
   if (S > R)
      if (P > R)
          printf("\n\n El orden es S, P y R: %8.2f %8.2f %8.2f", S, P, R);
      else
          printf("\n\n El orden es S, R y P: %8.2f %8.2f %8.2f", S, R, P);
   else
      printf("\n\n El orden es R, S y P: %8.2f %8.2f %8.2f", R, S, P);
 }
```

EJEMPLO 2.8

Construye un diagrama de flujo y el programa correspondiente que, al recibir como datos la matrícula, la carrera, el semestre que cursa y el promedio de un alumno de una universidad privada de Lima, Perú, determine si el alumno puede ser *asistente* de alguna de las carreras que se ofrecen en la universidad. si el alumno reúne los requisitos planteados en la tabla 2.4, se debe escribir su matrícula, la carrera y el promedio correspondiente.

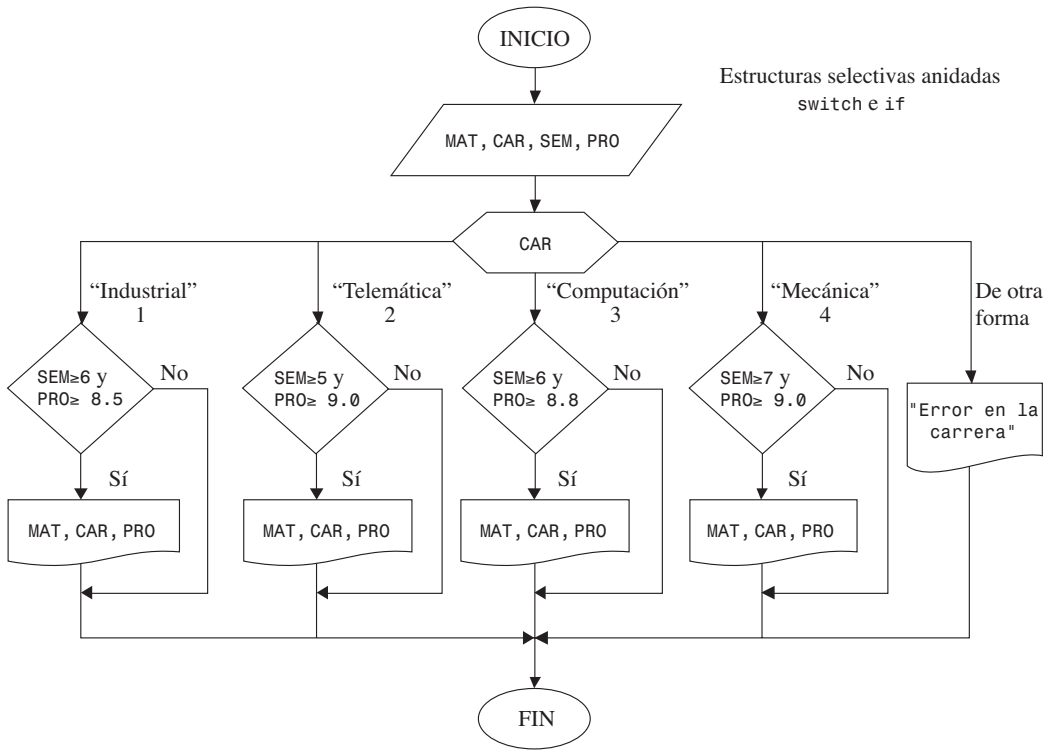
TABLA 2.4

Carrera	*Semestre*	*Promedio*
Industrial: 1	≥ 6	≥ 8.5
Telemática: 2	≥ 5	≥ 9.0
Computación: 3	≥ 6	≥ 8.8
Mecánica: 4	≥ 7	≥ 9.0

Datos: `MAT, CAR, SEM y PRO`

Donde: MAT es una variable de tipo entero que indica la matrícula del alumno.
CAR es una variable de tipo entero que representa la carrera.
SEM es una variable de tipo entero que señala el semestre que cursa.
PRO es una variable de tipo real que indica el promedio del alumno.

Diagrama de flujo 2.8



A continuación presentamos el programa correspondiente.

Programa 2.8

```
#include <stdio.h>

/* Asistentes.
```

```
  El programa, al recibir como datos la matrícula, la carrera, el semestre
  ➥y el promedio de un  alumno de una universidad privada, determina si
  ➥éste puede ser asistente de su carrera.

  MAT, CAR y SEM: variables de tipo entero.
  PRO: variable de tipo real. */

  void main(void)
  {
   int MAT, CAR, SEM;
   float PRO;
   printf("Ingrese matrícula: ");
   scanf("%d", &MAT);
   printf("Ingrese carrera (1-Industrial 2-Telemática 3-Computación
   4-Mecánica) : ");
   scanf("%d", &CAR);
   printf("Ingrese semestre: ");
   scanf("%d", &SEM);
   printf("Ingrese promedio: ");
   scanf("%f", &PRO);
   switch(CAR)
  {
      case 1: if (SEM >= 6 && PRO >= 8.5)
                  printf("\n%d %d %5.2f", MAT, CAR, PRO);
            break;
      case 2: if (SEM >= 5 && PRO >= 9.0)
                  printf("\n%d %d %5.2f", MAT, CAR, PRO);
            break;
      case 3: if (SEM >= 6 && PRO >= 8.8)
                  printf("\n%d %d %5.2f", MAT, CAR, PRO);
            break;
      case 4: if (SEM >= 7 && PRO >= 9.0)
                  printf("\n%d %d %5.2f", MAT, CAR, PRO);
            break;
         default: printf("\n Error en la carrera");
            break;
  }
}
```

A continuación se presenta una serie de problemas resueltos, diseñados como elementos de ayuda para el análisis y la retención de los conceptos. Más adelante se presenta una serie de problemas suplementarios cuidadosamente seleccionados.

Problemas resueltos

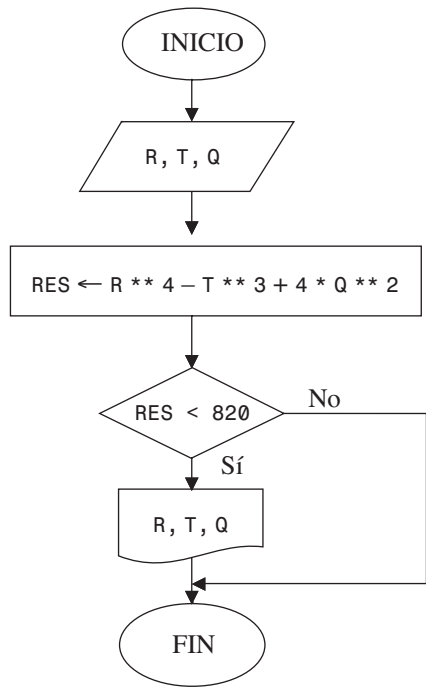
Problema PR2.1

Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos tres valores enteros `R`, `T` y `Q`, determine si los mismos satisfacen la siguiente expresión, y que, en caso afirmativo, escriba los valores correspondientes de `R`, `T` y `Q`.

$$R^4 - T^3 + 4 * Q^2 < 820$$

Datos: `R`, `T` y `Q` (variables de tipo entero).

Diagrama de flujo 2.9



Donde: `ew3` es una variable de tipo real que almacena el resultado de la expresión.

Programa 2.9

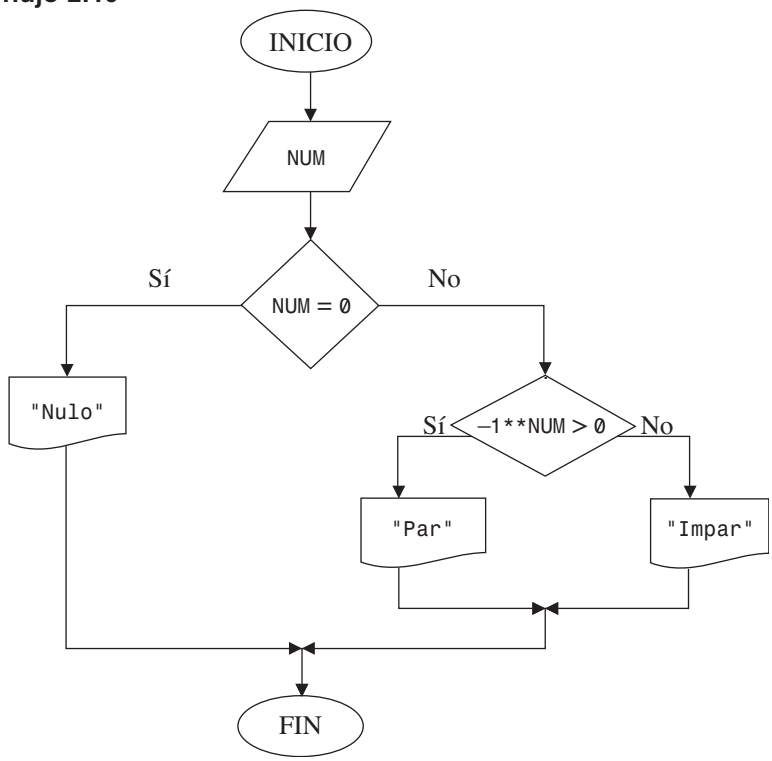
```
#include <stdio.h>
#include <math.h>

/* Expresión.
 El programa, al recibir como datos tres valores enteros, establece si los
 ➥mismos satisfacen una expresión determinada.

R, T y Q: variables de tipo entero.
RES: variable de tipo real. */

void main(void)
{
float RES;
int R, T, Q;
printf("Ingrese los valores de R, T y Q: ");
scanf("%d %d %d", &R, &T, &Q);
RES = pow(R, 4) - pow(T, 3) + 4 * pow(Q, 2);
if (RES < 820)
      printf("\nR = %d\tT = %d\t Q = %d", R, T, Q);
}
```

Problema PR2.2

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un número entero, determine e imprima si el mismo es par, impar o nulo.

Dato: `NUM` (variable entera que representa el número que se ingresa).

Diagrama de flujo 2.10



Programa 2.10

```
#include <stdio.h>
#include <math.h>

/* Par, impar o nulo.

NUM: variable de tipo entero. */

void main(void)
{
int NUM;
printf("Ingrese el número: ");
scanf("%d", &NUM);
if (NUM == 0)
       printf("\nNulo");
else
       if (pow (-1, NUM) > 0)
          printf("\nPar");
       else
          printf("\nImpar");
}
```

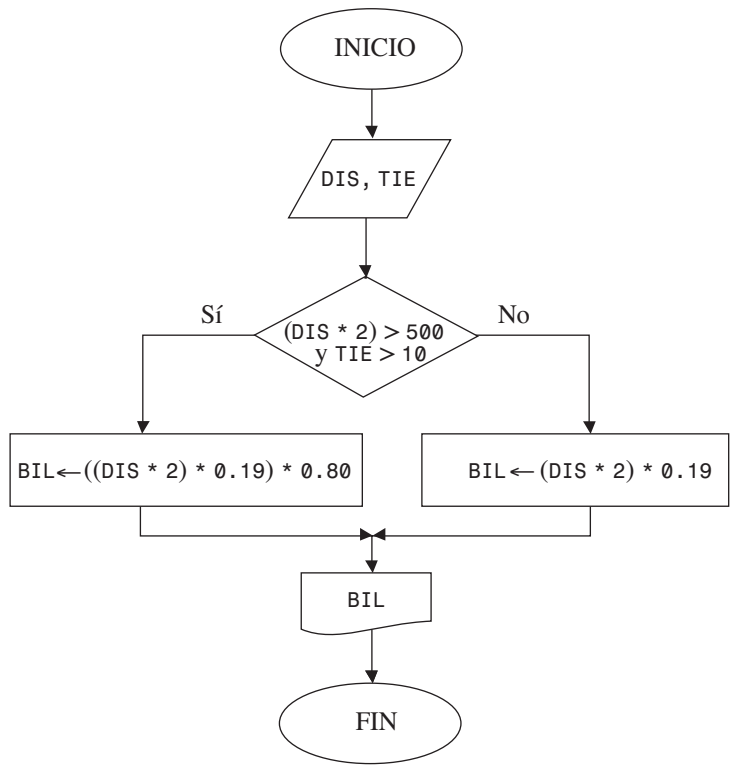
Problema PR2.3

Construye un diagrama de flujo y el correspondiente programa en **C** que permita calcular el precio del billete ida y vuelta en ferrocarril, conociendo tanto la distancia entre las dos ciudades como el tiempo de estancia en la ciudad destino. Si el número de días de estancia es superior a 10 y la distancia total (ida y vuelta) a recorrer es superior a 500 km, el precio del billete se reduce 20%. El precio por km es de $0.19.

Datos: `DIS` y `TIE`.

Donde: `DIS` es una variable de tipo entero que representa la distancia entre las dos ciudades.
`TIE` es una variable de tipo entero que expresa el tiempo de estancia.

Diagrama de flujo 2.11



Donde: `BIL` es una variable de tipo real que almacena el costo del billete.

Programa 2.11

```
#include <stdio.h>

/* Billete de ferrocarril.
El programa calcula el costo de un billete de ferrocarril teniendo en
➥cuenta la distancia entre las dos ciudades y el tiempo de permanencia
➥del pasajero.

DIS y TIE: variables de tipo entero.
BIL: variable de tipo real. */

void main(void)
{
 int DIS, TIE;
 float BIL;
 printf("Ingrese la distancia entre ciudades y el tiempo de estancia: ");
 scanf("%d  %d", &DIS, &TIE);
 if ((DIS*2 > 500) && (TIE > 10))
    BIL = DIS * 2 * 0.19 * 0.8;
 else
    BIL = DIS * 2  * 0.19;
 printf("\n\nCosto del billete: %7.2f", BIL);
}
```

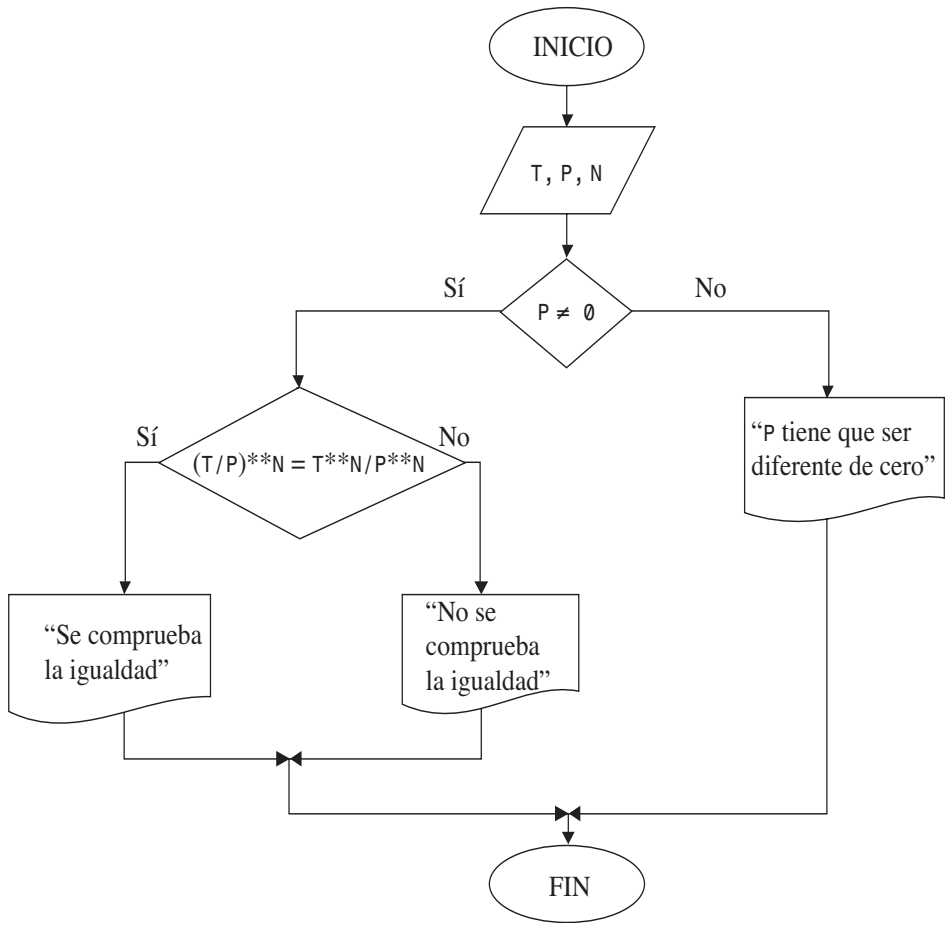
Problema PR2.4

Construye un diagrama de flujo y un programa en **C** que, al recibir como datos tres valores enteros `T`, `P` y `N`, permita comprobar la igualdad de la siguiente expresión:

$$\left[\frac{T}{P}\right]^N = \frac{T^N}{P^N}$$

Datos: `T`, `P` y `N` (variables de tipo entero que representan los datos que se ingresan).

Diagrama de flujo 2.12



Programa 2.12

```
  #include <stdio.h>
  #include <math.h>

  /* Igualdad de expresiones.
  El programa, al recibir como datos T, P y N, comprueba la igualdad de
una expresión determinada.

  T, P y N: variables de tipo entero. */

  void main(void)
  {
```

```
    int T, P, N;
    printf("Ingrese los valores de T, P y N: ");
    scanf("%d %d %d", &T, &P, &N);
    if (P != 0)
    {
       if (pow(T / P, N) ==  (pow(T, N) / pow(P, N))
          printf("\nSe comprueba la igualdad");
       else
          printf("\nNo se comprueba la igualdad");
    }
    else
         printf("\nP tiene que ser diferente de cero");
}
```

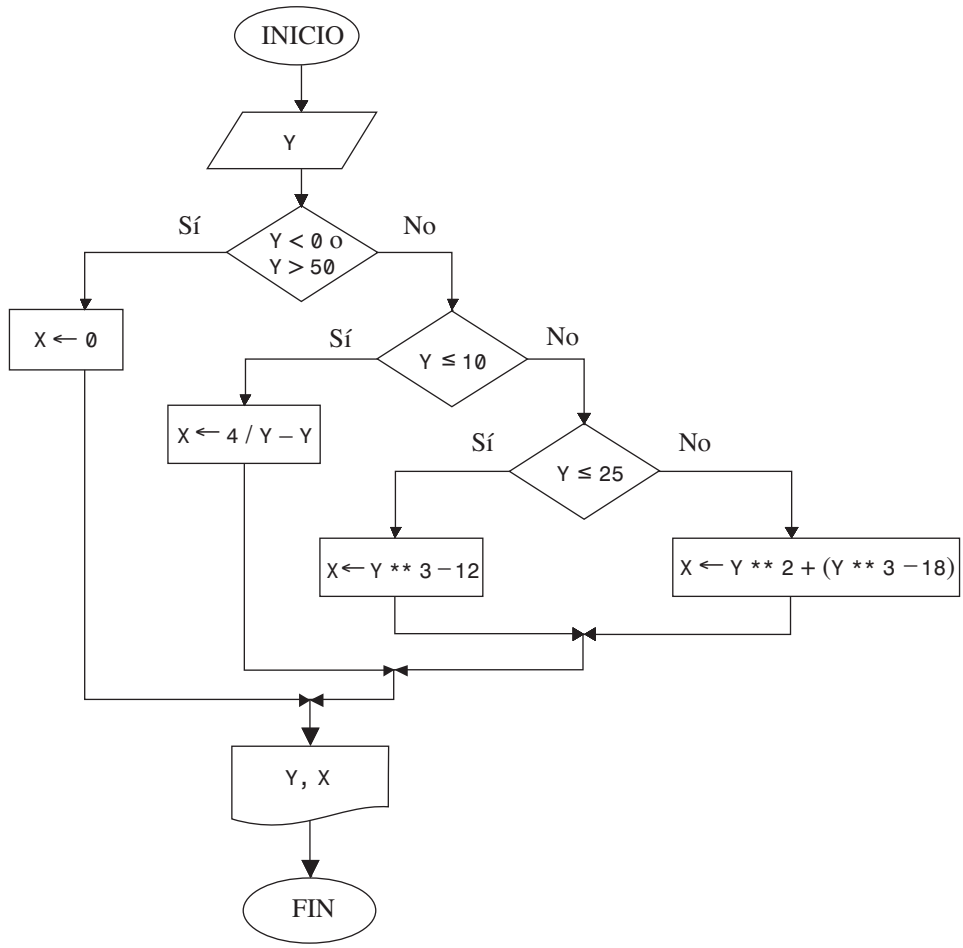

Problema PR2.5

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato Y, calcule el resultado de la siguiente función e imprima los valores de X y Y.

$$F(X) = \begin{cases} 4/Y - Y & \text{Si } 0 \leq Y \leq 10 \\ Y^3 - 12 & \text{Si } 11 < Y \leq 25 \\ Y^2 + (Y^3 - 18) & \text{Si } 25 < Y \leq 50 \\ 0 & \text{Para otro valor de Y} \end{cases}$$

Dato: Y (variable de tipo entero).

Diagrama de flujo 2.13



Donde: X es una variable de tipo real que almacena el resultado de la función.

Programa 2.13

```
#include <stdio.h>
#include <math.h>

/* Función.
El programa, al recibir como dato un valor entero, calcula el resultado de
➥una función.

Y: variable de tipo entero.
X: variable de tipo real. */
```

```
void main(void)
{
float X;
int Y;
printf("Ingrese el valor de Y: ");
scanf("%d", &Y);
if (Y < 0 || Y > 50)
    X = 0;
else
      if (Y <= 10)
          X =  4 / Y - Y;
      else
         if (Y <= 25)
             X = pow(Y, 3) - 12;
         else
             X = pow(Y, 2) + pow(Y, 3) - 18;
printf("\n\nY = %d\tX = %8.2f", Y, X);
}
```


Problema PR2.6

Una empresa de telecomunicaciones canadiense ofrece servicio de *callback* a un precio atractivo. El costo de las llamadas telefónicas depende tanto del lugar de origen de la llamada como de la zona geográfica en la que se encuentre el país destino. En la tabla 2.5 se presenta el costo por 60 segundos para las llamadas originadas en México. Construye un diagrama de flujo y el correspondiente programa en **C** que permita calcular e imprimir el costo total de una llamada telefónica, considerando tanto la zona como la duración de la llamada.

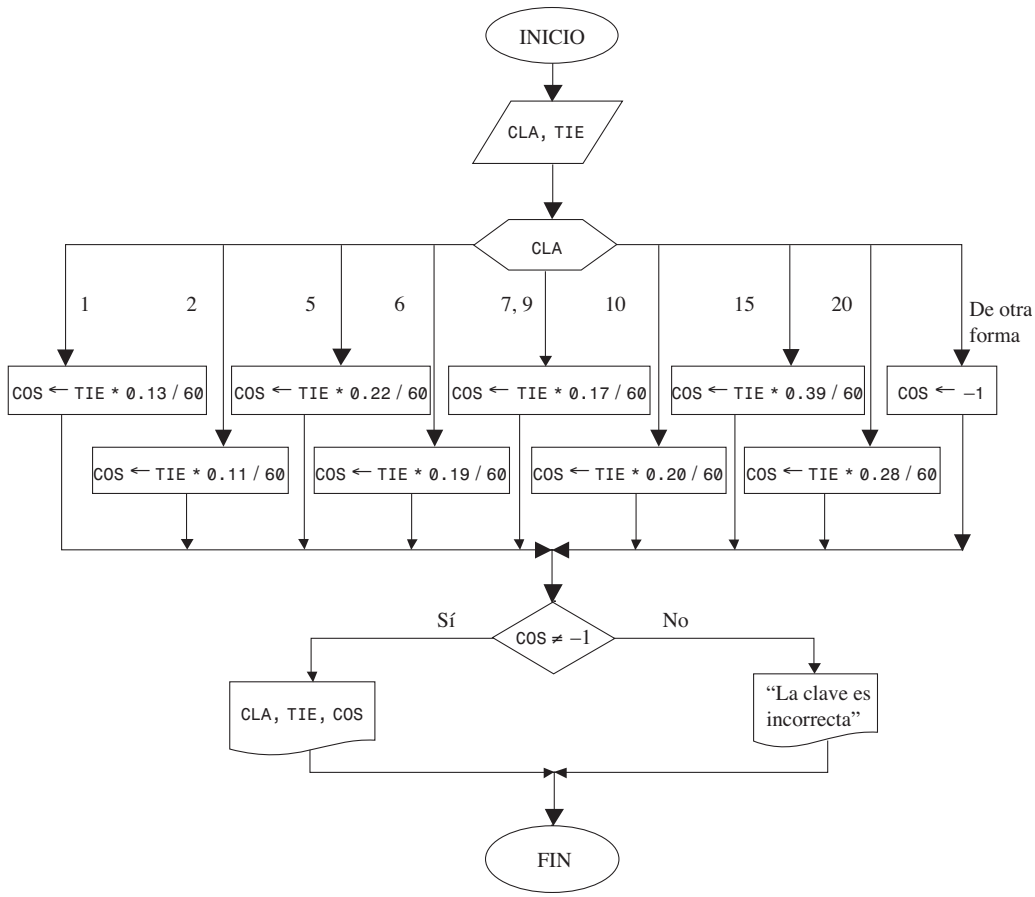
TABLA 2.5

Clave	*Zona*	*Precio*
1	Estados Unidos	0.13
2	Canadá	0.11
5	América del Sur	0.22
6	América Central	0.19
7	México	0.17
9	Europa	0.17
10	Asia	0.20
15	África	0.39
20	Oceanía	0.28

Datos: CLA y TIE

Donde: CLA es una variable de tipo entero que representa la zona geográfica.
TIE es una variable de tipo entero que representa la llamada en segundos.

Diagrama de flujo 2.14



Donde: COS es una variable de tipo real que almacena el costo de la llamada.

Programa 2.14

```
#include <stdio.h>

/* Teléfono.
El programa, al recibir como datos la clave de la zona geográfica y el
➥número de segundos de una llamada telefónica, calcula el costo de la misma.
```

```
CLA y TIE: variables de tipo entero.
COS: variable de tipo real. */

void main(void)
{
 int CLA, TIE;
 float COS;
 printf("Ingresa la clave y el tiempo: ");
 scanf("%d %d", &CLA, &TIE);
 switch(CLA)
 {
        case 1: COS = TIE * 0.13 / 60;  break;
        case 2: COS = TIE * 0.11 / 60;  break;
        case 5: COS = TIE * 0.22 / 60;  break;
        case 6: COS = TIE * 0.19 / 60;  break;
        case 7:
        case 9: COS = TIE * 0.17 / 60;  break;
        case 10: COS = TIE * 0.20 / 60;  break;
        case 15: COS = TIE * 0.39 / 60;  break;
        case 20: COS = TIE * 0.28 / 60;  break;
        default : COS = -1; break;
 }
 if (COS != -1)
        printf("\n\nClave: %d\tTiempo: %d\tCosto: %6.2f", CLA, TIE, COS);
 else
        printf("\nError en la clave");
}
```


Problema PR2.7

En un *spa* de *Ixtapan de la Sal* realizaron un análisis de los clientes registrados en los últimos cinco años con el objeto de conocer los gastos de internación de cada cliente. Construye un diagrama de flujo y el correspondiente programa en **C** que calcule el costo de internación de un cliente, según los datos de la tabla 2.6. Se sabe que los clientes mayores de 60 años tienen un descuento de 25% y los clientes menores de 25 años, de 15%.

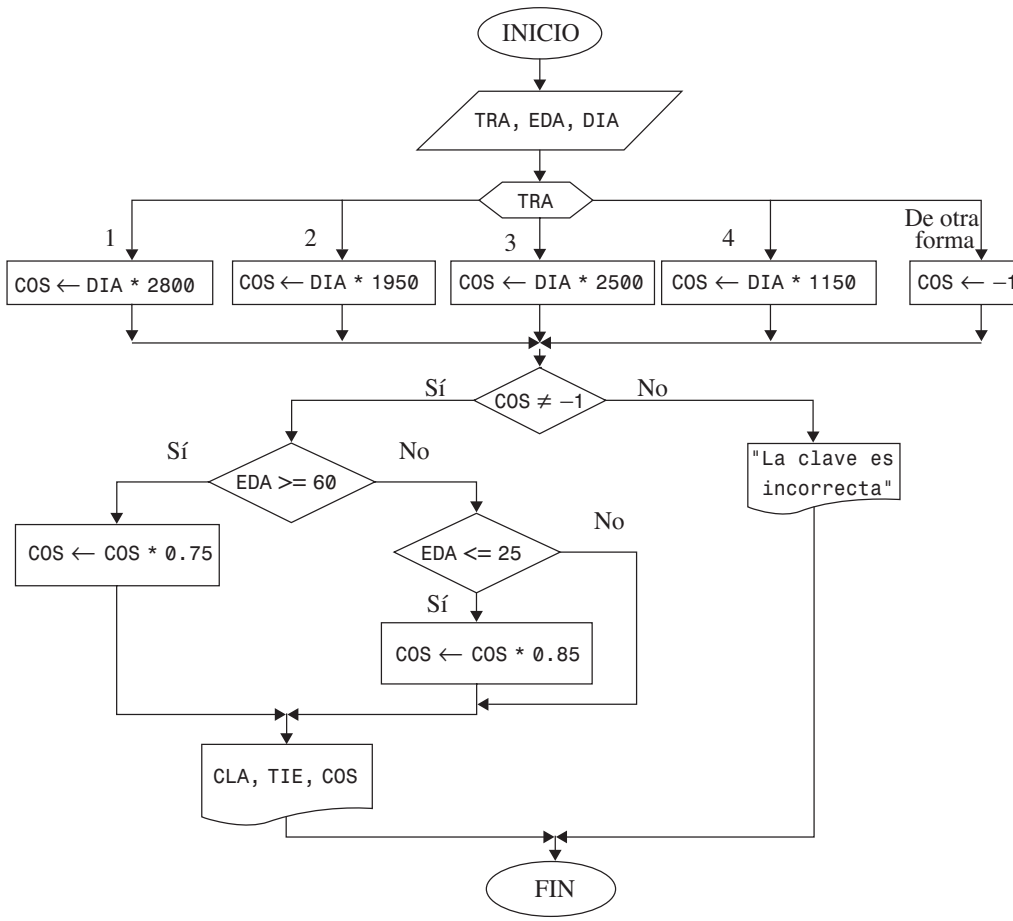
TABLA 2.6

Tipo de tratamiento	*Costo/cliente/día*
1	2800
2	1950
3	2500
4	1150

Datos: TRA, EDA y DIA

Donde: TRA es una variable de tipo entero que representa el tipo de tratamiento.
EDA es una variable de tipo entero que representa la edad del cliente.
DIA es una variable de tipo entero que expresa el número de días.

Diagrama de flujo 2.15



Donde: COS es una variable de tipo real que representa el costo del tratamiento.

Programa 2.15

```
#include <stdio.h>

/* Spa.
El programa, al recibir como datos el tipo de tratamiento, la edad y el
➥número de días de internación de un cliente en un spa, calcula el costo
➥total del tratamiento.

TRA, EDA, DIA: variables de tipo entero.
COS: variable de tipo real. */

void main(void)
{
 int TRA, EDA, DIA;
 float COS;
 printf("Ingrese tipo de tratamiento, edad y días: ");
 scanf("%d %d %d", &TRA, &EDA, &DIA);
 switch(TRA)
 {
        case 1: COS = DIA * 2800; break;
        case 2: COS = DIA * 1950;  break;
        case 3: COS = DIA * 2500;  break;
        case 4: COS = DIA * 1150; break;
        default: COS = -1; break;
 }
 if (COS != -1)
 {
        if (EDA >= 60)
           COS = COS * 0.75;
        else
           if (EDA <= 25)
             COS = COS * 0.85;
        printf("\nClave tratamiento: %d\t Días: %d\t Costo total: %8.2f",
TRA, DIA, COS);
    }
    else
       printf("\nLa clave del tratamiento es incorrecta");
}
```

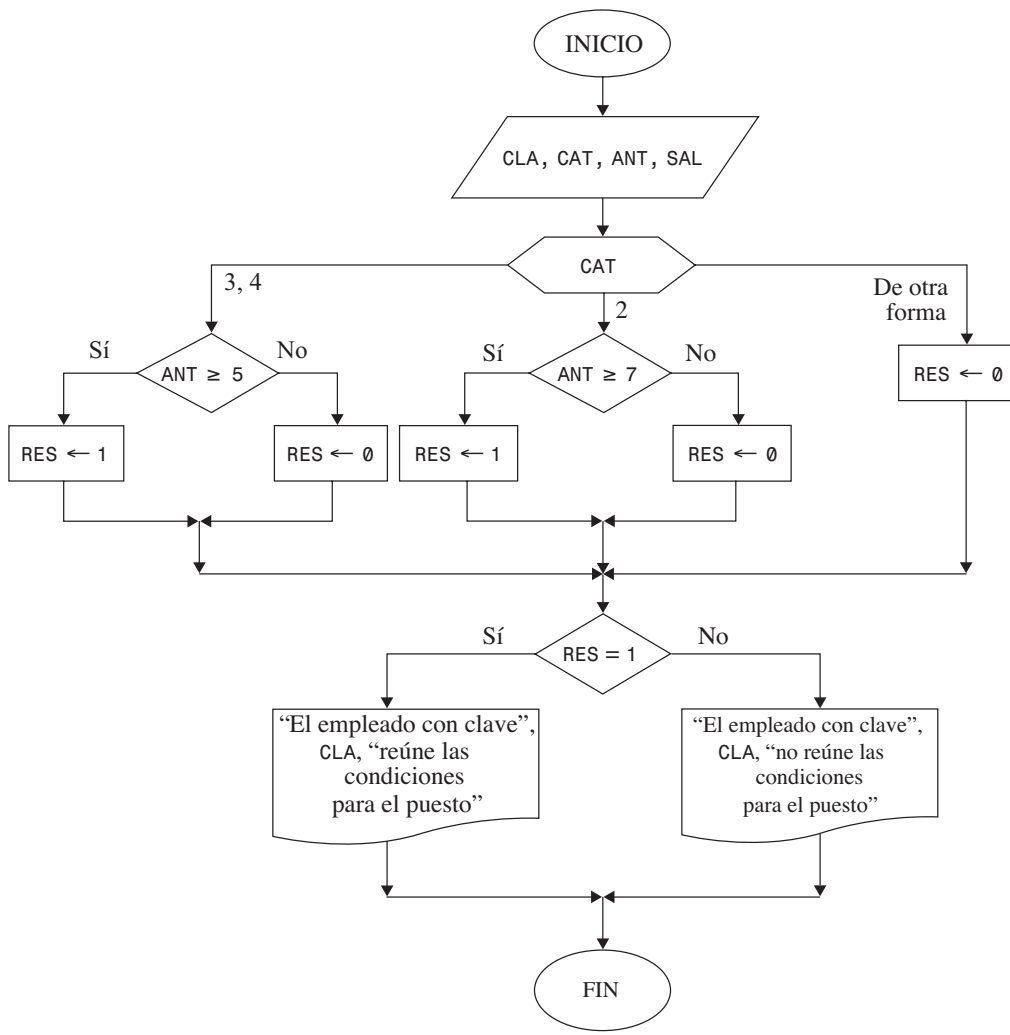
Problema PR2.8

En una empresa textil ubicada en La Paz, Bolivia, necesitan un empleado para una sucursal. Construye un diagrama de flujo y el correspondiente programa en **C** que compruebe e imprima si un empleado determinado reúne las condiciones necesarias para tal puesto. Las condiciones que estableció la empresa son las siguientes: categoría 3 o 4 y antigüedad mayor a 5 años, o bien categoría 2 y antigüedad mayor a 7 años.

Datos: CLA, CAT, ANT y SAL

Donde: CLA es una variable de tipo entero que representa la clave del trabajador.
CAT es una variable de tipo entero que representa la categoría del empleado.
ANT es una variable de tipo entero que expresa la antigüedad del trabajador en la empresa.
SAL es una variable de tipo real que representa el salario del trabajador.

Diagrama de flujo 2.16



Donde: RES es una variable de tipo entero que almacena la decisión sobre el candidato, 1 si reúne las condiciones y 0 en caso contrario.

Programa 2.16

```
# include <stdio.h>

/* Empresa textil.
El programa, al recibir como datos decisivos la categoría y antigüedad de
➥un empleado, determina si el mismo reúne las condiciones establecidas por
➥la empresa para ocupar un nuevo cargo en una sucursal.

CLA, CAT, ANT, RES: variables de tipo entero.
SAL: variable de tipo real. */

void main(void)
{
int CLA, CAT, ANT, RES;
printf("\nIngrese la clave, categoría y antigüedad del trabajador:");
scanf("%d %d %d", &CLA, &CAT, &ANT);
switch(CAT)
{
       case 3:
       case 4: if (ANT >= 5)
                   RES = 1;
               else
                     RES = 0;
               break;
       case 2: if (ANT >= 7)
                       RES = 1;
               else
                         RES = 0;
               break;
       default: RES = 0;
              break;
}
if (RES)
       printf("\nEl trabajador con clave %d reúne las condiciones para el
       ➥puesto", CLA);
else
       printf("\nEl trabajador con clave %d no reúne las condiciones para
       ➥el puesto", CLA);
}
```

Problemas suplementarios

Problema PS2.1

El número de sonidos emitidos por un grillo en un minuto es una función de la temperatura. Es posible entonces determinar el nivel de la temperatura (fórmula 2.1) utilizando un grillo como termómetro. Construye un diagrama de flujo y el correspondiente programa en **C** que calcule la temperatura con base en el número de sonidos emitidos por el grillo.

$$FA = S / 4 + 40$$

Fórmula 2.1

Donde: `FA` representa la temperatura en grados Fahrenheit y S el número de sonidos emitidos por minuto.

Dato: `S` (variable de tipo entero que representa el número de sonidos emitidos por el grillo).

Problema PS2.2

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato el salario de un profesor de una universidad, calcule el incremento del salario de acuerdo con el siguiente criterio y escriba el nuevo salario del profesor.

Salario < \$18,000 ⇒ Incremento 12%.
\$18,000 ≤ Salario ≤ \$30,000 ⇒ Incremento 8%.
\$30,000 < Salario ≤ \$50,000 ⇒ Incremento 7%.
\$50,000 < Salario ⇒ Incremento 6%.

Dato: `SAL` (variable de tipo real que representa el salario del profesor).

Problema PS2.3

Construye un diagrama de flujo y el correspondiente programa en **C** que determine, al recibir como datos dos números enteros, si un número es divisor de otro.

Datos: `N1` y `N2` (variables de tipo entero que representan los datos que se ingresan).

Problema PS2.4

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos de entrada tres valores enteros diferentes entre sí, determine si los mismos están en orden creciente.

Datos : `N1`, `N2` y `N3` (variables de tipo entero que representan los datos que se ingresan).

Problema PS2.5

En una tienda departamental ofrecen descuentos a los clientes en la Navidad, de acuerdo con el monto de su compra. El criterio para establecer el descuento se muestra abajo. Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato el monto de la compra del cliente, obtenga el precio real que debe pagar luego de aplicar el descuento correspondiente.

Compra < \$800 ⇒ Descuento 0%.
\$800 ≤ Compra ≤ \$1500 ⇒ Descuento 10%.
\$1500 < Compra ≤ \$5000 ⇒ Descuento 15%.
\$5000 < Compra ⇒ Descuento 20%.

Dato: `COM` (variable de tipo real que representa el monto de la compra).

Problema PS2.6

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos tres números reales, identifique cuál es el mayor. Considera que los números pueden ser iguales.

Datos: `N1`, `N2` y `N3` (variables de tipo real que representan los números que se ingresan).

Problema PS2.7

Construye un diagrama de flujo y el correspondiente programa en **C** que permita calcular el valor de *f(x)* según la siguiente expresión:

$$f(X) = \begin{cases} Y^3 & \text{Si } (Y \bmod 4) = 0 \\ (Y^2 - 14)/Y^3 & \text{Si } (Y \bmod 4) = 1 \\ Y^3 + 5 & \text{Si } (Y \bmod 4) = 2 \\ \sqrt{Y} & \text{Si } (Y \bmod 4) = 3 \end{cases}$$

Dato: `Y` (variable de tipo entero).

Problema PS2.8

Construye un diagrama de flujo y el correspondiente programa en C que permita convertir de pulgadas a milímetros, de yardas a metros y de millas a kilómetros.

Datos: `MED` y `VAL`

Donde: `MED` es una variable de tipo entero que se utiliza para el tipo de conversión que se quiere realizar.
`VAL` es una variable de tipo entero que representa el valor a convertir.

Consideraciones:

- 1 pulgada equivale a 25.40 milímetros.
- 1 yarda equivale a 0.9144 metros.
- 1 milla equivale a 1.6093 kilómetros.

Problema PS2.9

Construye un diagrama de flujo y el correspondiente programa en C que permita realizar la conversión de medidas de pesos, longitud y volumen, de acuerdo con la tabla 2.7. Se debe escribir el valor a convertir, la medida en que está expresado el valor, el nuevo valor y la nueva medida correspondiente.

TABLA 2.7

Medidas de longitud	*Medidas de volumen*	*Medidas de peso*
1 pulgada ≡ 25.40 milímetros	1 pie^3 ≡ 0.02832 metros^3	1 onza ≡ 28.35 gramos
1 yarda ≡ 0.9144 metros	1 yarda^3 ≡ 0.7646 metros^3	1 libra ≡ 0.45359 kilogramos
1 milla ≡ 1.6093 kilómetros	1 pinta ≡ 0.56826 litros	1 ton. inglesa ≡ 1.0160 toneladas

TABLA 2.7 Continuación

Medidas de longitud	*Medidas de volumen*	*Medidas de peso*
1 pulgada^2 ≡ 6.452 centímetros^2	1 galón ≡ 4.54609 litros	
1 pie^2 ≡ 0.09290 metros^2		
1 yarda^2 ≡ 0.8361 metros^2		
1 acre ≡ 0.4047 hectáreas		
1 milla^2 ≡ 2.59 kilómetros^2		

Datos: MED, SME y VAL

Donde: MED es una variable de tipo entero que representa el tipo de conversión que se va a realizar (longitud, volumen, peso).
SME es una variable de tipo entero que representa dentro de cada tipo de medida, el tipo de conversión que se va a realizar.
VAL es una variable de tipo entero que representa el valor que se va a convertir.

Problema PS2.10

En algunas oficinas del gobierno pagan horas extra a los burócratas, además del salario correspondiente. Escribe un diagrama de flujo y el correspondiente programa en **C** que permita calcular la cantidad a pagar a un trabajador tomando en cuenta su salario y las horas extra trabajadas. Las horas extra se calculan en función de la tabla 2.8. Cada trabajador puede tener como máximo 30 horas extra, si tienen más, sólo se les pagarán las primeras 30. Los trabajadores con categoría 4 o mayor a 4 no pueden recibir este beneficio.

TABLA 2.8

Categoría trabajador	*Hora extra*
1	$40
2	$50
3	$85

Datos: SAL, CAT y PHE

Donde: SAL es una variable de tipo real que representa el salario del burócrata.
CAT es una variable de tipo entero que representa la categoría del trabajador.
PHE es una variable de tipo entero que representa el número de horas extra.

Problema PS2.11

Construye un diagrama de flujo y el respectivo programa en **C** que, al recibir como datos tres variables reales que representan los lados de un probable triángulo, determine si esos lados corresponden a un triángulo. En caso de serlo, además de escribir el área correspondiente compruebe si el mismo es equilátero, isósceles o escaleno.

Datos: L1, L2 y L3 (variables de tipo real que representan los posibles lados de un triángulo).

Consideraciones:

- Si se cumple la propiedad de que la suma de los dos lados menores es menor a la del lado restante, es un triángulo.
- El área se obtiene aplicando la siguiente fórmula:

$$\text{ÁREA} = \sqrt{S * (SA) * (SB) * (SC)}$$

Fórmula 2.2

Problema PS2.12

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un número entero de cuatro dígitos, determine si todos los dígitos del número son pares. Por ejemplo, si el número fuera 5688, no cumpliría la condición ya que el dígito más significativo —5— sería impar; si, por el contrario, el número fuera 6244, sí cumpliría, ya que todos los dígitos son pares.

Dato: NUM (variable de tipo entero de cuatro dígitos).

CAPÍTULO 3

Estructuras algorítmicas repetitivas

3.1 Introducción

En la práctica, durante la solución de problemas, es muy común encontrar, operaciones que se deben ejecutar un número determinado de veces. Si bien las instrucciones son las mismas, los datos varían. El conjunto de instrucciones que se ejecuta repetidamente recibe el nombre de **ciclo**.

Todo ciclo debe terminar luego de repetirse un número finito de veces. Dentro del conjunto de instrucciones siempre debe existir una condición de parada o fin de ciclo. En cada iteración del mismo son evaluadas las condiciones necesarias para decidir si se debe seguir ejecutando o si debe detenerse.

En algunos algoritmos podemos establecer de antemano el número de veces que se debe repetir el ciclo. En este caso, el número de

repeticiones no depende de las proposiciones dentro del ciclo. La estructura algorítmica repetitiva `for` se utiliza para resolver problemas en los que conocemos el número de veces que se debe repetir el ciclo.

Por otra parte, en algunos algoritmos no podemos establecer de antemano el número de veces que se debe repetir el ciclo. Este número *depende* de las proposiciones que contenga el mismo. La estructura repetitiva `while` se utiliza para resolver problemas de este tipo.

Otra estructura algorítmica repetitiva es `do-while`. A diferencia de las estructuras anteriores en las que las condiciones se evalúan al principio del ciclo, en ésta se evalúan al final. Esto implica que el conjunto de instrucciones se ejecuta al menos una vez. El `do-while` es una estructura de menor interés que las anteriores y sólo se debe utilizar en ciertos casos, como en la validación de datos de entrada.

En este capítulo estudiaremos las tres estructuras algorítmicas repetitivas que ofrece el lenguaje **C**: `for`, `while` y `do-while`.

3.2 La estructura repetitiva `for`

Ésta es la estructura algorítmica utilizada para repetir un conjunto de instrucciones un número definido de veces. Este tipo de estructura se encuentra prácticamente en todos los lenguajes de programación. Es similar a la estructura `do` de Fortran y `for` de Pascal.

La estructura `for` del lenguaje **C** es muy similar a la estructura `while`. Sin embargo, por cuestiones didácticas, sólo aplicaremos la estructura `for` en aquellos problemas en los que se conozca previamente el número de veces que se debe repetir el ciclo. La estructura `while`, por otra parte, sólo se utilizará en la solución de aquellos problemas en los que el número de veces que se debe repetir el ciclo dependa de las proposiciones que contenga el mismo. El diagrama de flujo de la estructura algorítmica `for` es el siguiente:

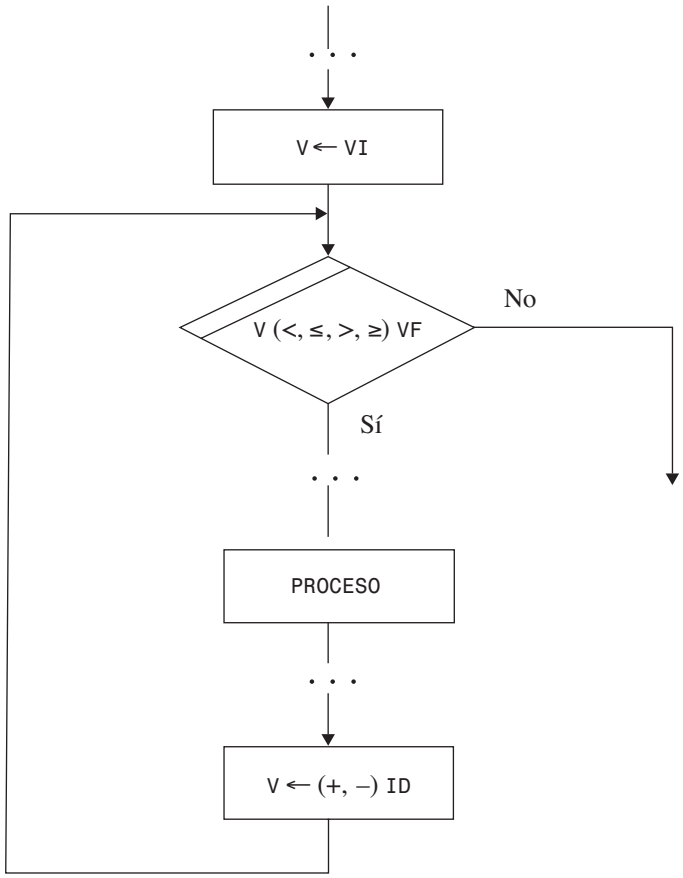


FIGURA 3.1
Estructura repetitiva `for`.

Donde: `V` representa la variable de control del ciclo, `VI` expresa el valor inicial, `VF` representa al valor final e `ID` representa el incremento o decremento de la variable de control, según si el ciclo es ascendente o descendente.

En el lenguaje **C** la estructura repetitiva `for` se escribe de la siguiente forma:

```
/* El conjunto de instrucciones muestra la sintaxis de la estructura for en C. */
. . .
for (V = VI; V (<, <=, >, >=) VF; V = V (+, -) ID)
{
    proceso;              /* cuerpo de la estructura for */
}
```

Observa que en la estructura `for` la variable de control del ciclo —`V`— va desde el valor inicial —`VI`— hasta el valor final —`VF`—. En cada iteración del ciclo el valor de `V` se incrementa o decrementa de acuerdo con —`ID`—, dependiendo si el ciclo es ascendente o descendente.

Nota 1: Es muy importante destacar que hay dos tipos de variables que se utilizan frecuentemente en los ciclos. Éstas se conocen como **contadores** y **acumuladores**. Los contadores, como su nombre lo indica, sirven para contar, y los acumuladores, para acumular. Ambas variables se inicializan generalmente en cero antes de iniciar el ciclo, aunque este valor puede ser diferente dependiendo del problema que se vaya a resolver.

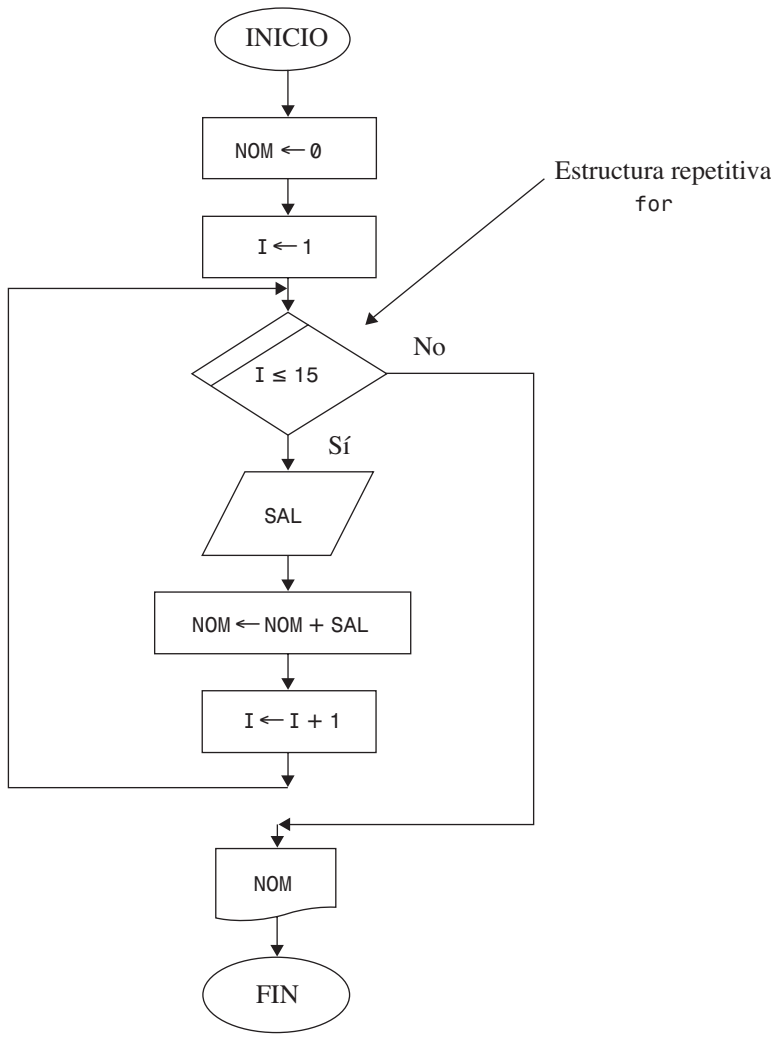
EJEMPLO 3.1

Construye un diagrama de flujo y el programa correspondiente en **C** que, al recibir como datos los salarios de 15 profesores de una universidad, obtenga el total de la nómina.

Datos: SAL_1, SAL_2,..., SAL_{15}

Donde: SAL_i $(1 \leq i \leq 15)$ es una variable de tipo real que representa el salario del profesor `i`.

Diagrama de flujo 3.1



Donde: `I` es una variable de tipo entero que representa la variable de control del ciclo.

`NOM` es una variable de tipo real que acumula los salarios. Generalmente, las variables que funcionan como *acumuladores* se inicializan en cero afuera del ciclo.

Nota 2: Observa que para resolver el problema se lee el salario y se acumula posteriormente para obtener el total. Estas dos operaciones se repiten 15 veces. Este dato lo conocemos de antemano y por esta razón utilizamos la estructura `for`.

En la tabla 3.1 se puede observar el seguimiento del algoritmo.

TABLA 3.1. Seguimiento del algoritmo

	I	*Datos*	*Resultados*
		SAL	*NOM*
	1		0
Inicio del ciclo →	2	12500.00	12500.00
	3	13600.50	26100.50
	4	12800.80	38901.30
	5	5600.50	44501.80
	6	7500.00	52002.60
	7	27500.00	79502.60
	8	19600.00	99102.60
	9	8500.00	107602.60
	10	32800.90	140403.50
	11	27640.35	168043.85
	12	16830.40	184874.25
	13	19650.70	204524.95
	14	15600.00	220124.95
	15	9750.30	229875.25
Fin del ciclo →	16	11800.90	**241676.15**

A continuación se presenta el programa correspondiente.

Programa 3.1

```
#include <stdio.h>

/* Nómina.
El programa, al recibir los salarios de 15 profesores, obtiene el total de la
➥nómina de la universidad.

I: variable de tipo entero.
SAL y NOM: variables de tipo real. */

void main(void)
{
int I;
float SAL, NOM;
NOM = 0;
for (I=1; I<=15; I++)
{
    printf("\Ingrese el salario del profesor%d:\t", I);
    scanf("%f", &SAL);
    NOM = NOM + SAL;
}
printf("\nEl total de la nómina es: %.2f", NOM);
}
```


EJEMPLO 3.2

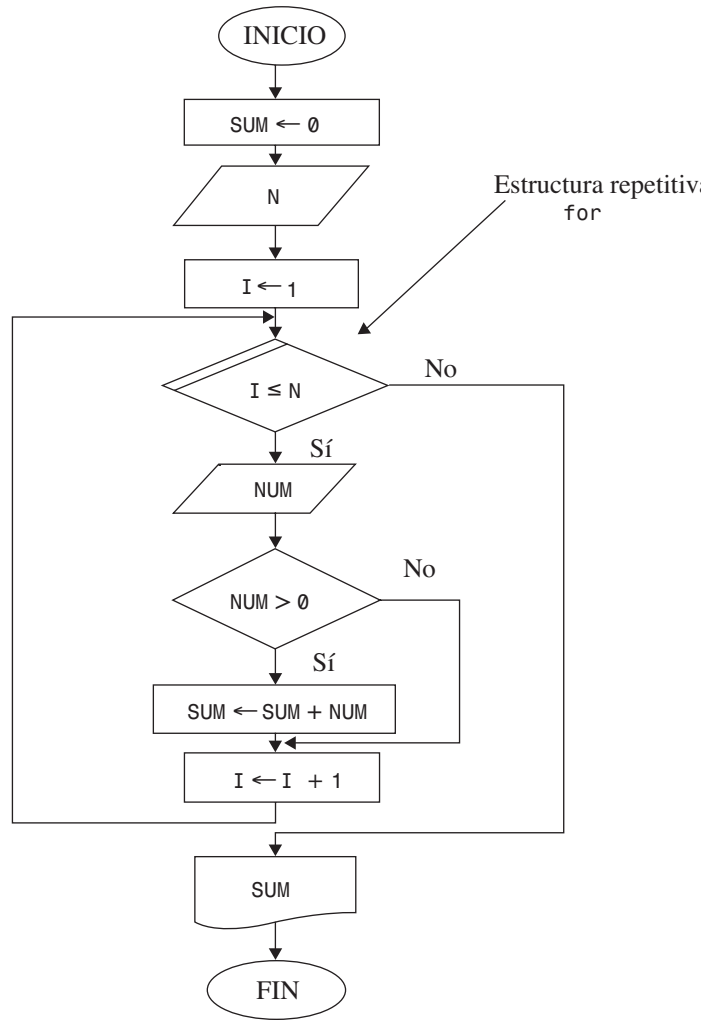
Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos N números enteros, obtenga solamente la suma de los números positivos.

Datos: N, NUM_1, NUM_2,..., NUM_N

Donde: N es una variable de tipo entero que representa el número de datos que se ingresan.

NUM_i ($1 \leq i \leq N$) es una variable de tipo entero que representa al número i.

Diagrama de flujo 3.2



Donde: `I` es una variable de tipo entero que representa al contador del ciclo.
`SUM` es una variable de tipo entero que se utiliza para sumar los números positivos. Observa que `SUM` se inicializa en cero antes de comenzar el ciclo.

Programa 3.2

```
#include <stdio.h>

/* Suma positivos.
El programa, al recibir como datos N números enteros, obtiene la suma de los
➥enteros positivos.

I, N, NUM, SUM: variables de tipo entero. */

void main(void)
{
int I, N, NUM, SUM;
SUM = 0;
printf("Ingrese el número de datos:\t");
scanf("%d", &N);
for (I=1; I<=N; I++)
{
   printf("Ingrese el dato número %d:\t", I);
   scanf("%d", &NUM);
   if (NUM >  0)
       SUM = SUM + NUM;
}
printf("\nLa suma de los números positivos es: %d", SUM);
}
```


3.3. La estructura repetitiva `while`

La estructura algorítmica repetitiva `while` permite repetir un conjunto de instrucciones. Sin embargo, el número de veces que se debe repetir depende de las proposiciones que contenga el ciclo. Cada vez que corresponde iniciar el ciclo se evalúa una condición, si ésta es verdadera (diferente de cero) se continúa con la ejecución, de otra forma se detiene.

Hay una gran cantidad de casos en los que podemos aplicar la estructura repetitiva `while`. Supongamos que debemos obtener el importe total de los pagos que realizamos en el último mes, pero no sabemos cuántos fueron. Debemos entonces sumar los mismos hasta que no encontremos más. Consideremos ahora que tenemos que obtener el promedio de calificaciones de un examen que realizaron un grupo de alumnos, pero no sabemos con exactitud cuántos lo aplicaron. Tenemos que sumar todas las calificaciones e ir contando el número de alumnos para posteriormente calcular el promedio. El ciclo se repite mientras tengamos calificaciones de alumnos.

El diagrama de flujo de la estructura repetitiva `while` es el siguiente:

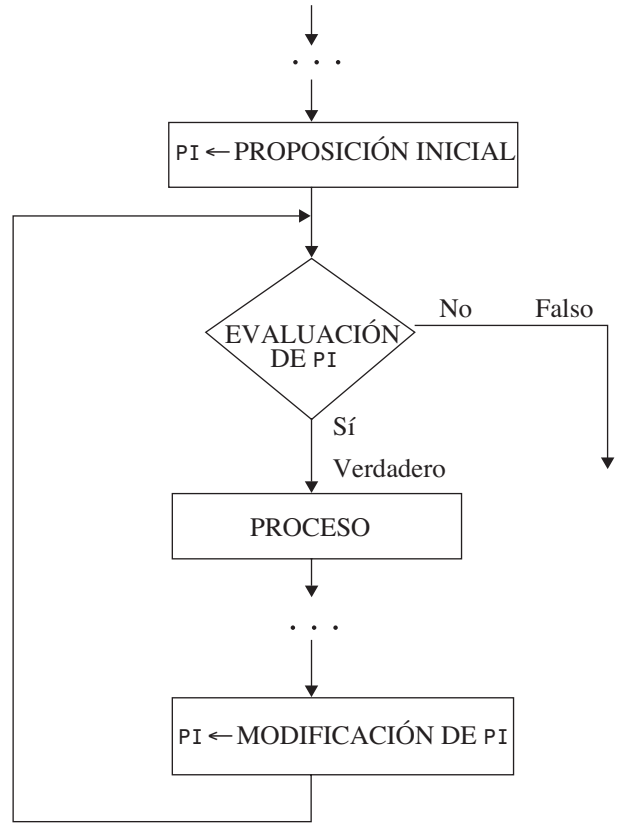


FIGURA 3.2
Estructura repetitiva `while`.

Donde: `PI` representa la proposición inicial. Debe tener un valor verdadero (diferente de cero) inicialmente para que el ciclo se ejecute. Además, dentro del ciclo siempre debe existir un enunciado que afecte la condición, de tal forma que aquél no se repita de manera infinita.

En el lenguaje **C** la estructura repetitiva `while` se escribe de la siguiente forma:

```
/* Estructura repetitiva while.
El conjunto de instrucciones muestra la sintaxis de la estructura while en el
➥lenguaje  C. */
. . .
PI = proposición inicial;
while (PI)             /* PI debe tener un valor verdadero para que el ciclo se
                       ➥ejecute */
{
```

```
        proceso;            /* cuerpo de la estructura while */
        . . .
        PI = modificación de PI;
        . . .
    }
```

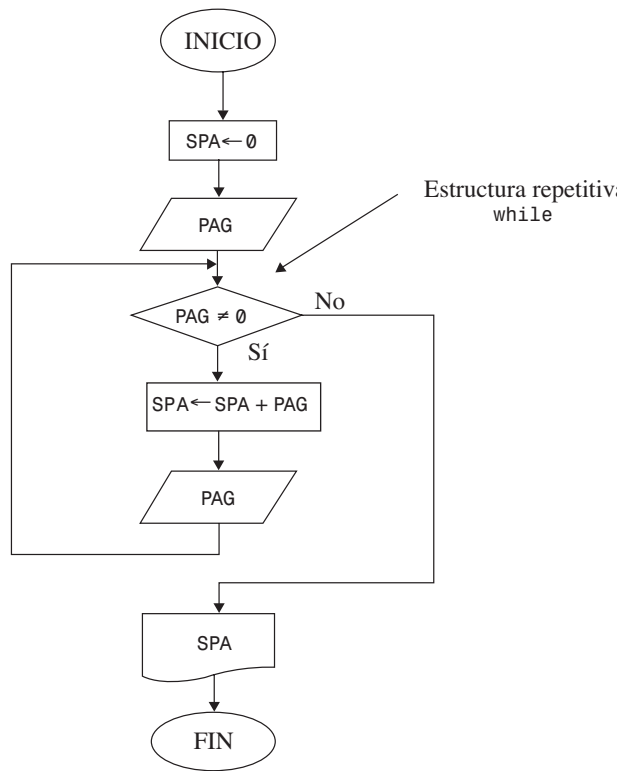
EJEMPLO 3.3

Construye un diagrama de flujo y el programa correspondiente en ***C*** que, al recibir como datos los pagos efectuados en el último mes, permita obtener la suma de los mismos.

Datos: PAG_1, PAG_2,..., 0

Donde: PAG_i es una variable de tipo real que representa al pago número i. Se ingresa 0 como último dato para indicar que ya no hay más pagos que contemplar.

Diagrama de flujo 3.3.



Donde: SPA es una variable de tipo real que se utiliza para acumular los pagos y se inicializa en cero.

Programa 3.3

```
#include <stdio.h>

/* Suma pagos.
El programa, al recibir como datos un conjunto de pagos realizados en el último
➥mes, obtiene la suma de los mismos.

PAG y SPA: variables de tipo real.   */

void main(void)
{
float PAG, SPA;
SPA = 0;
printf("Ingrese el primer pago:\t");
scanf("%f", &PAG);
while (PAG)
/* Observa que la condición es verdadera mientras el pago es diferente de cero. */
{
    SPA = SPA + PAG;
    printf("Ingrese el siguiente pago:\t ");
    scanf("%f", &PAG);
     /* Observa que la proposición que modifica la condición es una lectura. */
}
printf("\nEl total de pagos del mes es: %.2f", SPA);
}
```

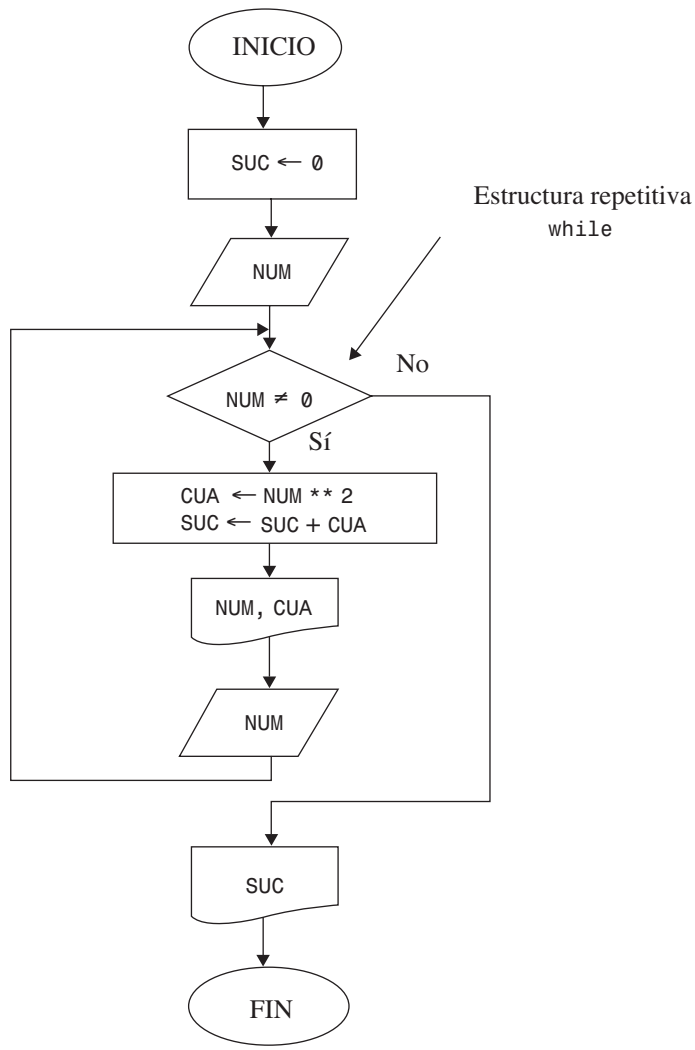
EJEMPLO 3.4

Construye un diagrama de flujo y el programa correspondiente en **C** que, al recibir como datos un grupo de números naturales positivos, calcule el cuadrado de estos números. Imprima el cuadrado del número y al final la suma de los cuadrados.

Datos: NUM_1, NUM_2, NUM_3,..., 0

Donde: NUM_i es una variable de tipo entero que representa al número positivo i. El fin de los datos está dado por 0.

Diagrama de flujo 3.4.



Donde: CUA es una variable de tipo entero extendido (*long*) que almacena el cuadrado del número que se ingresa.
SUC es una variable de tipo long que se utiliza para almacenar la suma de los cuadrados.

Programa 3.4

```
#include <stdio.h>
#include <math.h>

/* Suma cuadrados.
El programa, al recibir como datos un grupo de enteros positivos, obtiene el
➥cuadrado de los mismos y la suma correspondiente a dichos cuadrados. */

void main(void)
{
int NUM;
long CUA, SUC = 0;
printf("\nIngrese un número entero -0 para terminar-:\t");
scanf("%d", &NUM);
while (NUM)
/* Observa que la condición es verdadera mientras el entero es diferente de cero. */
{
    CUA = pow (NUM, 2);
    printf("%d al cubo es %ld", NUM, CUA);
    SUC = SUC + CUA;
    printf("\nIngrese un número entero -0 para terminar-:\t");
    scanf("%d", &NUM);
}
printf("\nLa suma de los cuadrados es %ld", SUC);
}
```

3.4. La estructura repetitiva `do-while`

Otra de las estructuras algorítmicas repetitivas en el lenguaje **C** es `do-while`. Es una estructura que se encuentra prácticamente en cualquier lenguaje de programación de alto nivel, similar al `repeat` de los lenguajes Pascal y Fortran. A diferencia de las estructuras `for` y `while`, en las cuales las condiciones se evalúan al principio del ciclo, en ésta se evalúan al final. Esto implica que el ciclo se debe ejecutar por lo menos una vez.

La estructura es adecuada cuando no sabemos el número de veces que se debe repetir un ciclo, pero conocemos que se debe ejecutar por lo menos una vez. Es decir, se ejecuta el conjunto de instrucciones una vez, y luego cada vez que corresponde iniciar nuevamente el ciclo se evalúan las condiciones, siempre al final del conjunto de instrucciones. Si el resultado es verdadero (diferente de cero) se continúa con la ejecución, de otra forma se detiene.

La estructura repetitiva `do-while` tiene una menor importancia que las estructuras repetitivas estudiadas en primer término. Sin embargo, puede utilizarse de manera

eficiente para verificar los datos de entrada de un programa. En la siguiente figura se presenta la estructura repetitiva `do-while`:

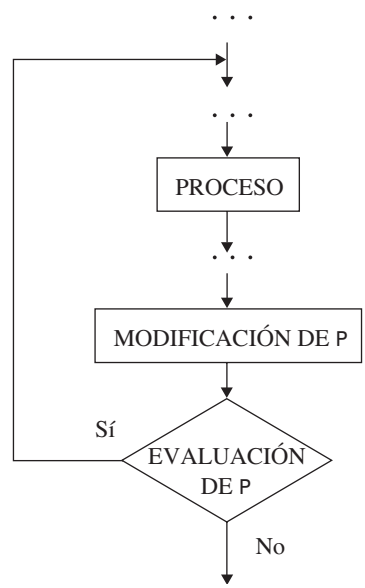


FIGURA 3.3
Estructura repetitiva `do-while`.

Donde: `P` representa la condición inicial. Debe tener un valor verdadero (diferente de cero) para que el conjunto de instrucciones se pueda volver a ejecutar. Siempre debe existir un enunciado dentro del ciclo que afecte la condición, para que éste no se repita de manera infinita.

En lenguaje **C**, la estructura repetitiva `do-while` se escribe de la siguiente forma:

```
/* Estructura repetitiva do-while.
El conjunto de instrucciones muestra la sintaxis de la estructura do-while en el
➥lenguaje C. */
 . . .
 do
 {
     proceso;           /* cuerpo de la estructura do-while. */
     . . .
     modificación de P;
 }
 while (P)
 /* P debe tener un valor verdadero para que el ciclo se vuelva a ejecutar */
 . . .
```

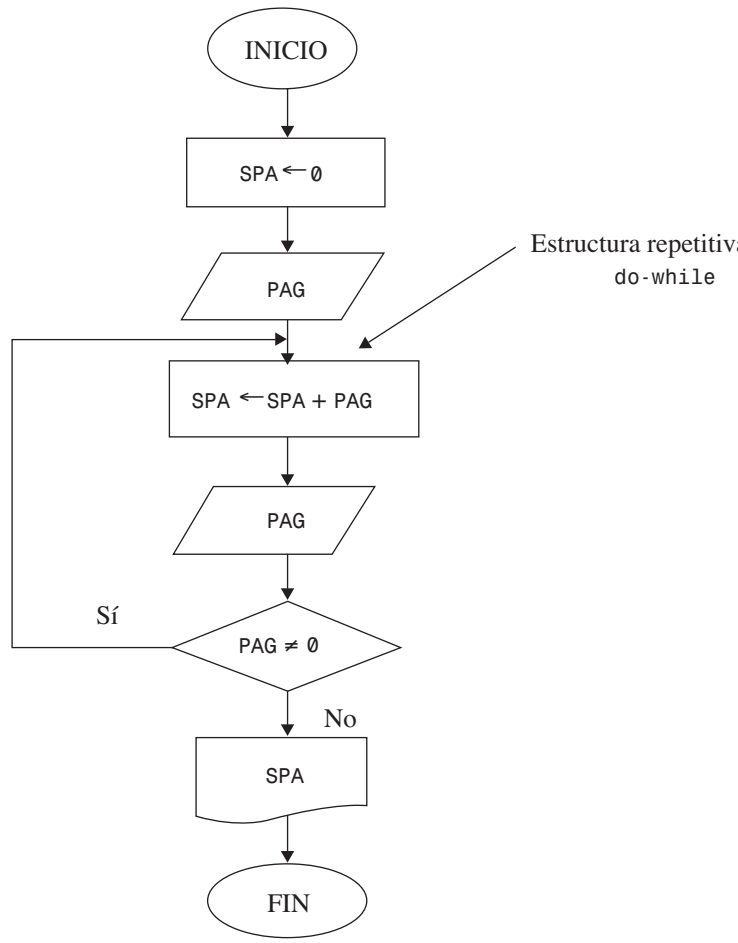
EJEMPLO 3.5

A continuación se resuelve el problema del ejemplo 3.3 aplicando la estructura repetitiva `do-while`. Supongamos que debemos obtener la suma de los pagos realizados en el último mes, pero no sabemos exactamente cuántos fueron. Los datos se expresan de la siguiente forma:

Datos: PAG_1, PAG_2,..., 0

Donde: PAG_i es una variable de tipo real que representa al pago número `i`. Se ingresa 0 como último dato para indicar que ya no hay más pagos que contemplar.

Diagrama de flujo 3.5



Donde: SPA es una variable de tipo real que se utiliza para acumular los pagos.

Nota 3: Observa que al menos debemos ingresar un pago para que no ocurra un error de ejecución en el programa.

A continuación se presenta el programa en el lenguaje **C**.

Programa 3.5

```
#include <stdio.h>

/* Suma pagos.
El programa obtiene la suma de los pagos realizados el último mes.

PAG y SPA: variables de tipo real.*/

void main(void)
{
float PAG, SPA = 0;
printf("Ingrese el primer pago:\t");
scanf("%f", &PAG);
/* Observa que al utilizar la estructura do-while al menos se necesita un pago.*/
do
{
   SPA = SPA + PAG;
   printf("Ingrese el siguiente pago  -0 para terminar-:\t ");
   scanf("%f", &PAG);
}
while (PAG);
printf("\nEl total de pagos del mes es: %.2f", SPA);
}
```

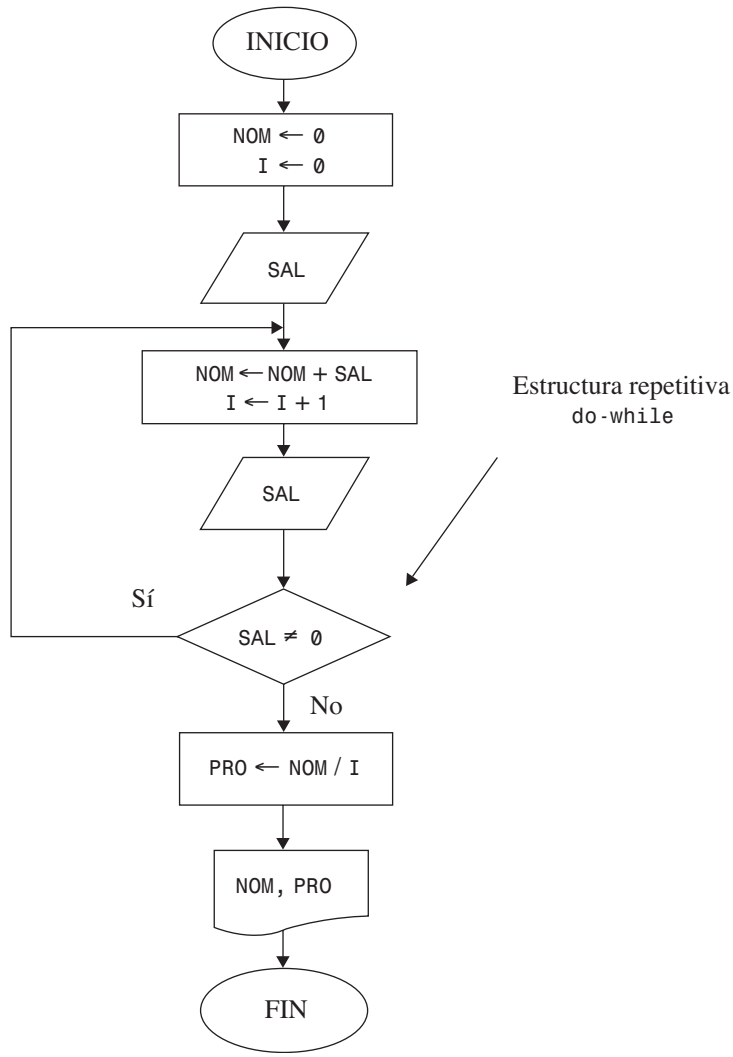

EJEMPLO 3.6

Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos los salarios de los profesores de una universidad, obtenga tanto la nómina como el promedio de los salarios.

Datos: SAL_1, SAL_2, SAL_3,..., 0

Donde: SAL_i es una variable de tipo real que representa el salario del profesor i.
El fin de datos está dado por 0.

Diagrama de flujo 3.6



Donde: `I` es una variable de tipo entero que se utiliza para contar el número de salarios.
`NOM` es una variable de tipo real que almacena la suma de los salarios.
`PRO` es una variable de tipo real que obtiene el promedio.

Programa 3.6

```
#include <stdio.h>

/* Nómina de profesores.
El programa, al recibir como datos los salarios de los profesores de una
➥universidad, obtiene la nómina y el promedio de los salarios.

I: variable de tipo entero.
SAL, NOM y PRO: variables de tipo real. */

void main(void)
{
int I = 0;
float SAL, PRO, NOM = 0;
printf("Ingrese el salario del profesor:\t");
/* Observa que al menos se necesita ingresar el salario de un profesor para que
➥no ocurra un error de ejecución del programa. */
scanf("%f", &SAL);
do
{
   NOM = NOM + SAL;
   I = I + 1;
   printf("Ingrese el salario del profesor -0 para terminar- :\t");
   scanf("%f", &SAL);
}
while (SAL);
PRO = NOM / I;
printf("\nNómina: %.2f \t Promedio de salarios: %.2f", NOM, PRO);
}
```


La estructura repetitiva `do-while`, como vimos anteriormente, tiene una menor importancia que las estructuras repetitivas `for` y `while`. Sin embargo, se puede utilizar de manera eficiente para verificar los datos de entrada del programa. Observemos a continuación el siguiente ejemplo.

EJEMPLO 3.7

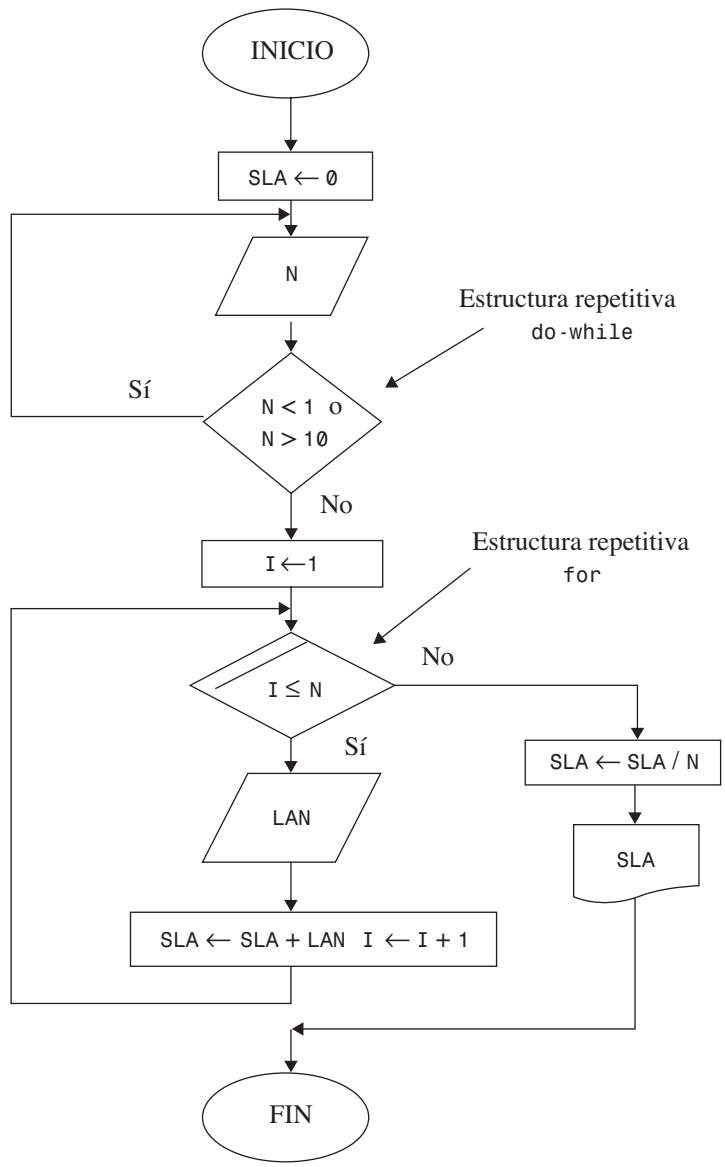
Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos los `N` lanzamientos del martillo de la atleta cubana ganadora de la medalla de oro en las últimas olimpiadas celebradas en Atenas, calcule el promedio de dichos lanzamientos.

Datos: $N, LAN_1, LAN_2, \ldots, LAN_N$

Donde: `N` es una variable de tipo entero que representa los lanzamientos de la atleta, `0 < N < 11`.

SAL_i es una variable de tipo real que representa el lanzamiento `i` de la atleta.

Diagrama de flujo 3.7



Donde: I es una variable de tipo entero que se utiliza para el control del ciclo.
SLA es una variable de tipo real que almacena los N lanzamientos. Al final se utiliza esa misma variable para almacenar el promedio de los lanzamientos.

Programa 3.7

```
#include <stdio.h>

/* Lanzamiento de martillo.
El programa, al recibir como dato N lanzamientos de martillo, calcula el promedio
➥de los lanzamientos de la atleta cubana.

I, N: variables de tipo entero.
LAN, SLA: variables de tipo real. */

void main(void)
{
int I, N;
float LAN, SLA = 0;
do
{
    printf("Ingrese el número de lanzamientos:\t");
    scanf("%d", &N);
}
while (N < 1 || N > 11);
/* Se utiliza la estructura do-while para verificar que el valor de N sea
➥correcto. */
for (I=1; I<=N; I++)
{
     printf("\nIngrese el lanzamiento %d: ", I);
     scanf("%f", &LAN);
     SLA = SLA + LAN;
}
SLA = SLA / N;
printf("\nEl promedio de lanzamientos es: %.2f", SLA);
}
```

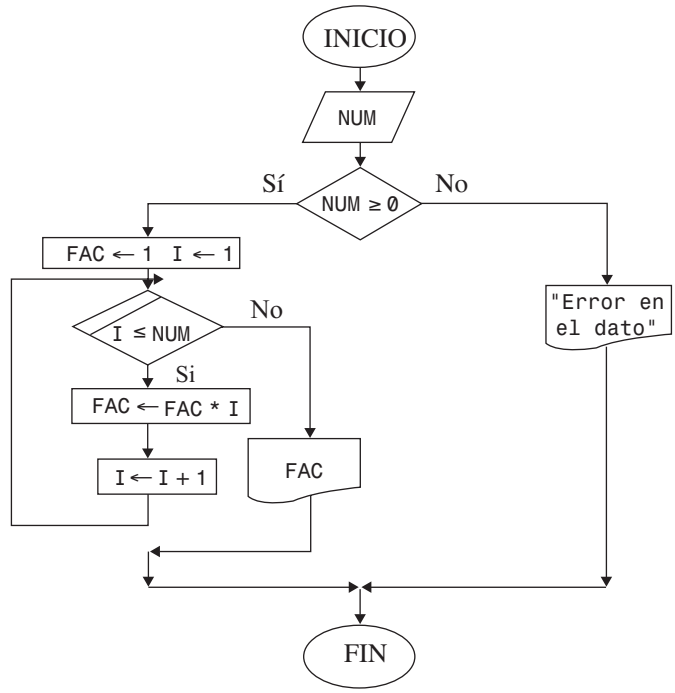

Problemas resueltos

Problema PR3.1

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un número entero N, calcule el factorial de dicho número. Recuerda que el *Factorial (0) es 1*, el *Factorial(1) es 1*, y el *Factorial (n)* se calcula como *n * Factorial(n-1).*

Dato: NUM (variable de tipo entero que representa el número que se ingresa).

Diagrama de flujo 3.8



Donde: `FAC` es una variable de tipo entero que se utiliza para almacenar el factorial.
`I` es una variable de tipo entero que se utiliza para el control del ciclo.

Programa 3.8

```
# include <stdio.h>

/* Factorial.
El programa calcula el factorial de un número entero.

FAC, I, NUM: variables de tipo entero. */

void main(void)
{
int I, NUM;
long FAC;
printf("\nIngrese el número: ");
scanf("%d", &NUM);
if (NUM >= 0)
{
   FAC = 1;
   for (I=1; I <= NUM; I++)
```

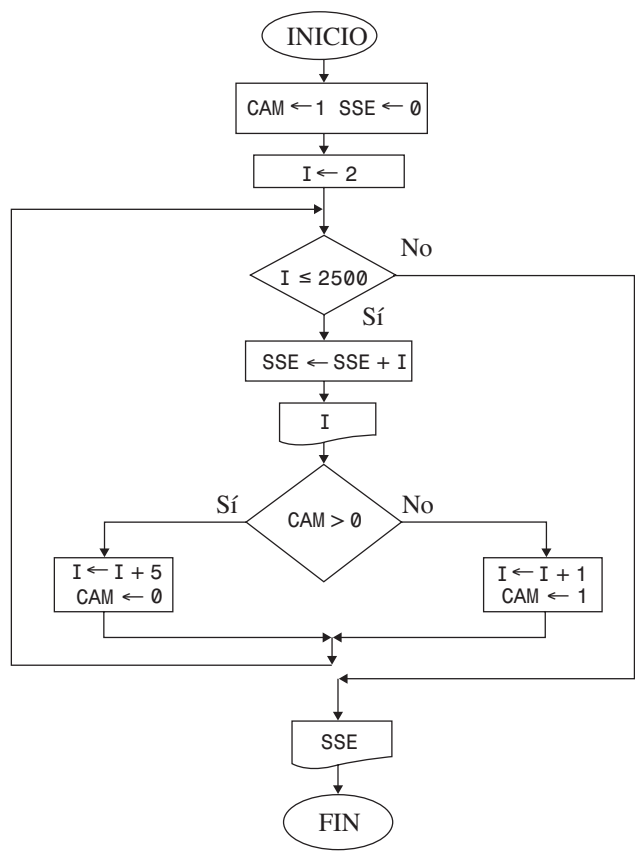
```
            FAC *= I;
      printf("\El factorial de %d es: %ld", NUM, FAC);
   }
   else
      printf("\nError en el dato ingresado");
   }
```

Problema PR3.2

Escribe un diagrama de flujo y el correspondiente programa en **C** que obtenga y escriba tanto los términos como la suma de los términos de la siguiente serie:

```
2, 7, 10, 15, 18, 23, . . ., 2500
```


Diagrama de flujo 3.9



Donde: I es una variable de tipo entero que se utiliza para incrementar el valor de los términos de la serie.
SSE es una variable de tipo entero que se utiliza para sumar los términos.
CAM es una variable de tipo entero que se utiliza para distinguir el valor a sumar.

Programa 3.9

```
#include <stdio.h>

/* Serie.
El programa imprime los términos y obtiene la suma de una determinada serie.

I, SSE y CAM: variable de tipo entero. */

void main(void)
{
int I = 2, CAM = 1;
long SSE = 0;
while (I <= 2500)
{
    SSE = SSE + I;
    printf("\t %d", I);
    if (CAM)
    {
         I += 5;
         CAM--;
    }
    else
    {
         I += 3;
         CAM++;
    }
}
printf("\nLa suma de la serie es: %ld", SSE);
}
```

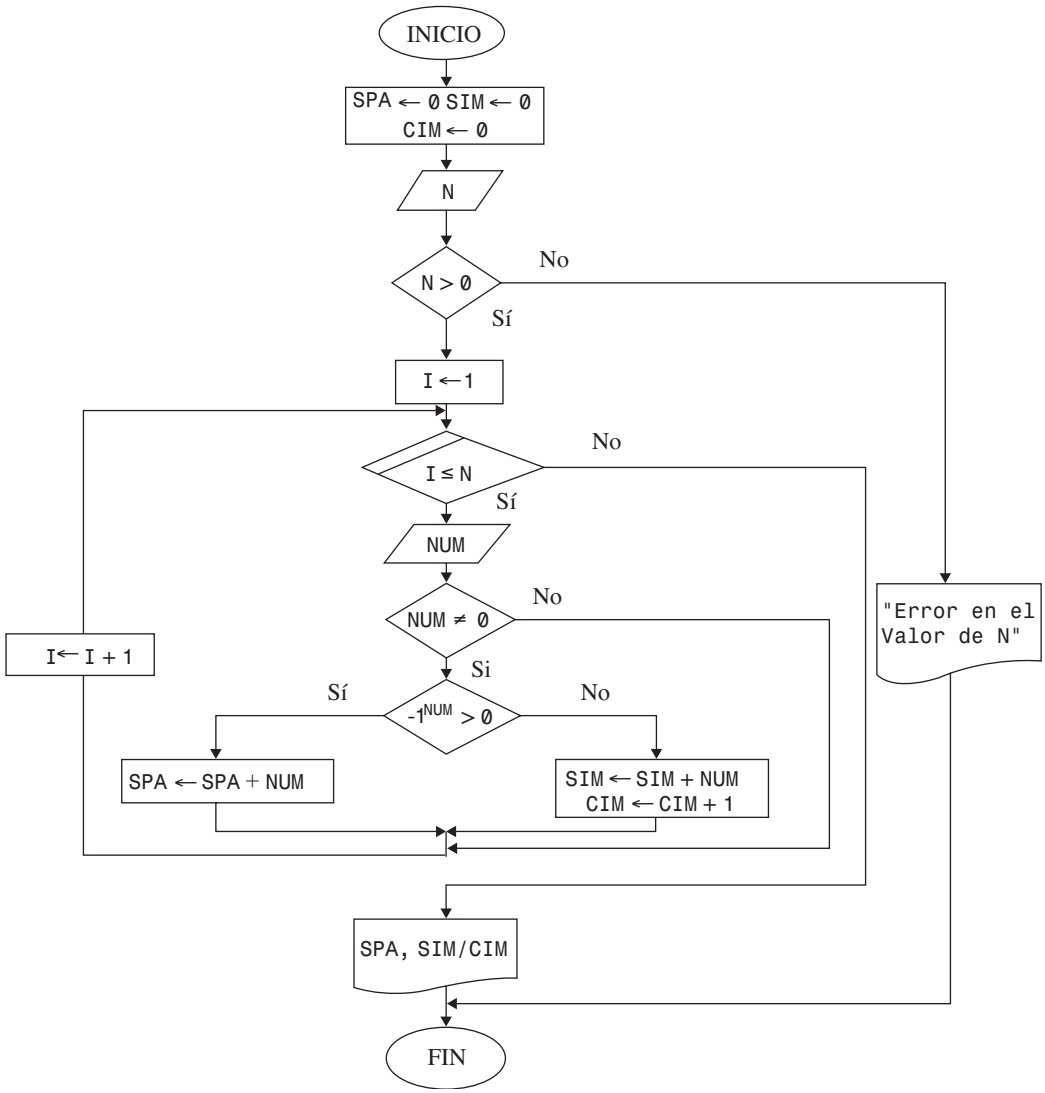
Problema PR3.3

Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos N números enteros, obtenga la suma de los números pares y el promedio de los impares.

Datos: $N, NUM_1, NUM_2, \ldots, NUM_N$

Donde: N es una variable de tipo entero que representa el número de datos que se ingresa.
NUM_i $(1 \leq i \leq N)$ es una variable de tipo entero que representa al número entero i.

Diagrama de flujo 3.10



Donde: `I` es una variable de tipo entero que se utiliza para el control del ciclo.
`SPA` y `SIM` son variables de tipo entero que se utilizan para sumar los pares e impares, respectivamente.
`CIM` es una variable de tipo entero que se utiliza para contar los impares.

Programa 3.10

```
#include <stdio.h>
#include <math.h>

/* Pares e impares.
El programa, al recibir como datos N números enteros, obtiene la suma de los
➥números pares y calcula el promedio de los impares.

I, N, NUM, SPA, SIM, CIM: variables de tipo entero. */

void main(void)
{
int I, N, NUM, SPA = 0, SIM = 0, CIM = 0;
printf("Ingrese el número de datos que se van a procesar:\t");
scanf("%d", &N);
if (N > 0)
{
    for (I=1; I <= N; I++)
    {
        printf("\nIngrese el número %d: ", I);
        scanf("%d", &NUM);
        if (NUM)
             if (pow(-1, NUM) > 0)
                  SPA = SPA + NUM;
             else
                 {
                     SIM = SIM + NUM;
                     CIM++;
                 }
    }
    printf("\n La suma de los números pares es: %d", SPA);
    printf("\n El promedio de números impares es: %5.2f", (float)(SIM / CIM);
}
else
    printf("\n El valor de N es incorrecto");
}
```

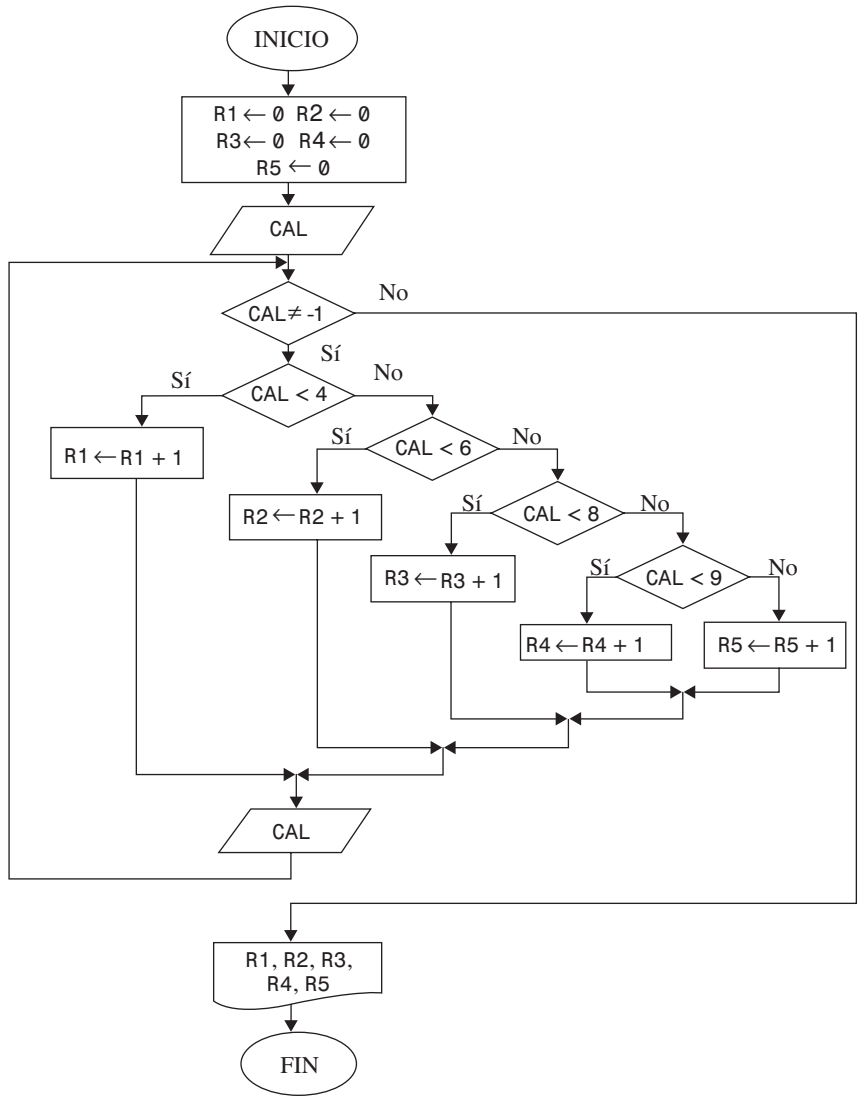
Problema PR3.4

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos las calificaciones de un grupo de alumnos que presentaron su examen de admisión para ingresar a una universidad privada en México, calcule y escriba el número de calificaciones que hay en cada uno de los siguientes rangos:

0 . . . 3.99
4 . . . 5.99
6 . . . 7.99
8 . . . 8.99
9 . . . 10

Datos: $CAL_1, CAL_2, \ldots, -1$ (CAL_i es una variable de tipo real que representa la calificación del alumno `i`).

Diagrama de flujo 3.11



Donde: `R1`, `R2`, `R3`, `R4` y `R5` son variables de tipo entero que funcionan como acumuladores.

Programa 3.11

```
#include <stdio.h>

/* Examen de admisión.
El programa, al recibir como datos una serie de calificaciones de un examen,
➥obtiene el rango en que se encuentran éstas.

R1, R2, R3, R4 y R5: variable de tipo entero.
CAL: variable de tipo real. */

void main(void)
{
int R1 = 0, R2 = 0, R3 = 0, R4 = 0, R5 = 0;
float CAL;
printf("Ingresa la calificación del alumno: ");
scanf("%f", &CAL);
while (CAL != -1)
{
  if (CAL < 4)
       R1++;
  else
      if (CAL < 6)
           R2++;
       else
          if (CAL < 8)
               R3++;
          else
              if (CAL < 9)
                   R4++;
              else
                 R5++;
    printf("Ingresa la calificación del alumno: ");
    scanf("%f", &CAL);
}
printf("\n0..3.99 = %d", R1);
printf("\n4..5.99 = %d", R2);
printf("\n6..7.99 = %d", R3);
printf("\n8..8.99 = %d", R4);
printf("\n9..10   = %d", R5);
}
```

Problema PR3.5

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un entero positivo, obtenga e imprima la sucesión de ULAM, la cual se llama así en honor del matemático S. Ulam.

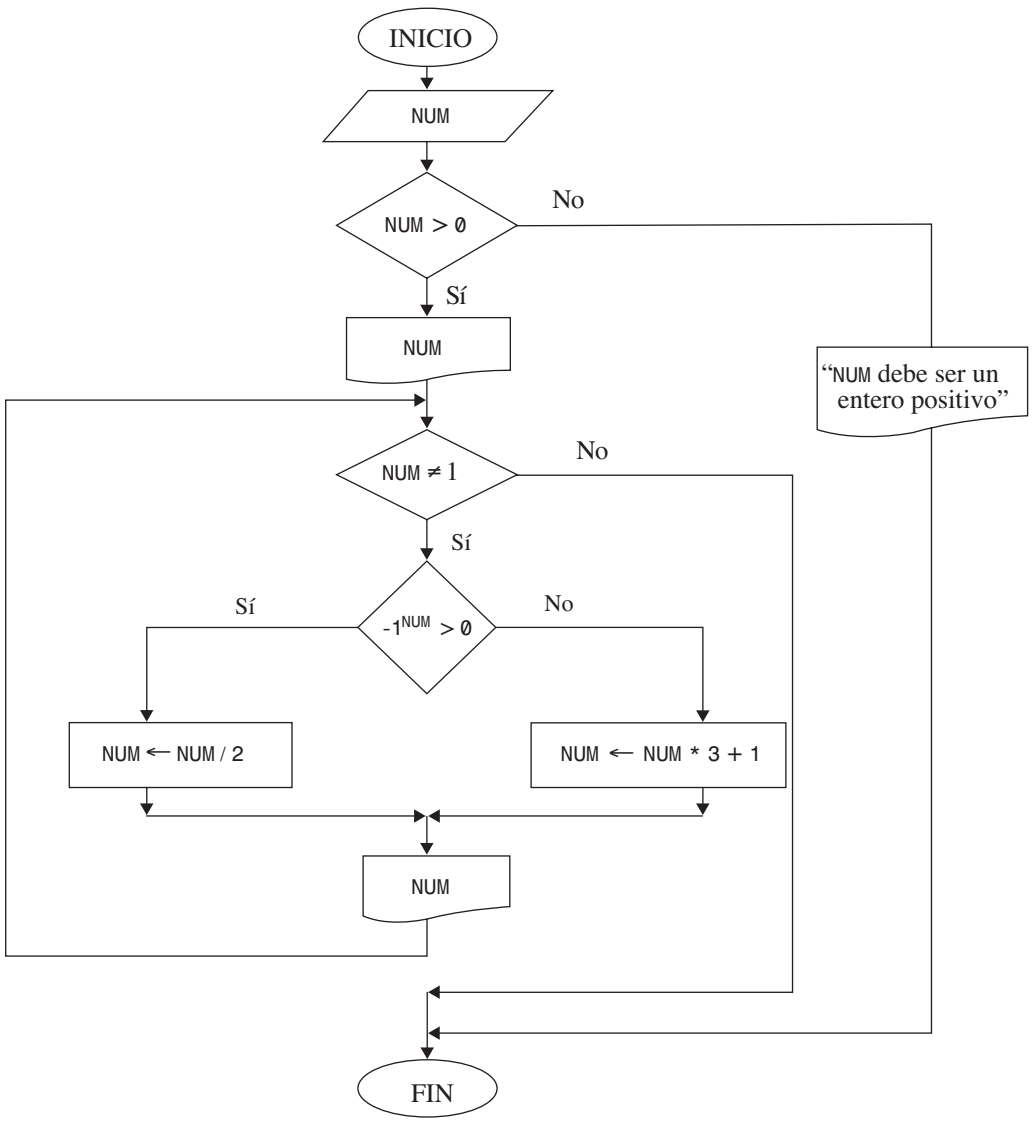
Sucesión de ULAM

1. Inicia con cualquier entero positivo.
2. Si el número es par, divídelo entre 2. Si es impar, multiplícalo por 3 y agrégale 1.
3. Obtén sucesivamente números enteros repitiendo el proceso.

Al final obtendrás el número 1. Por ejemplo, si el entero inicial es 45, la secuencia es la siguiente: 45, 136, 68, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Dato: N (variable de tipo entero que representa el entero positivo que se ingresa).

Diagrama de flujo 3.12



Programa 3.12

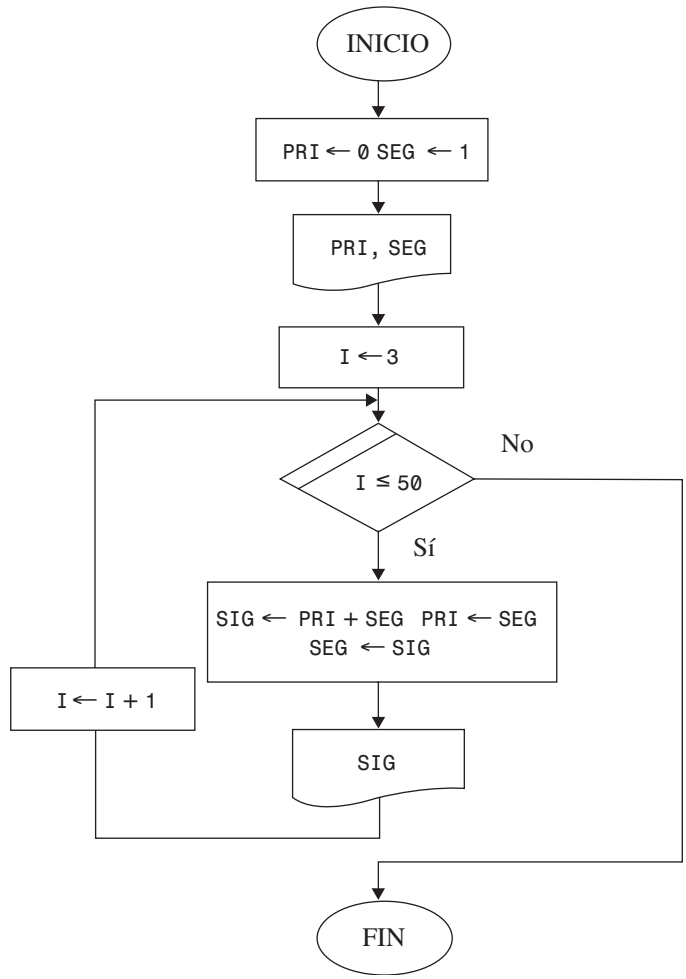
```
#include <stdio.h>
#include <math.h>

/* Serie de ULAM.
El programa, al recibir como dato un entero positivo, obtiene y escribe
➥la serie de ULAM.

NUM: variable de tipo entero. */

void main(void)
{
int NUM;
printf("Ingresa el número para calcular la serie: ");
scanf("%d", &NUM);
if (NUM > 0)
{
    printf("\nSerie de ULAM\n");
    printf("%d \t", NUM);
    while (NUM != 1)
    {
        if (pow(-1, NUM) > 0)
             NUM = NUM / 2;
        else
             NUM = NUM * 3 + 1;
        printf("%d \t", NUM);
    }
}
else
    printf("\n NUM debe ser un entero positivo");
}
```

Problema PR3.6

Escribe un diagrama de flujo y el correspondiente programa en **C** que calcule e imprima los primeros 50 números de Fibonacci. Recuerda que *Fibonacci(0) es 0*, *Fibonacci(1) es 1*, y *Fibonacci(n)* se calcula como *Fibonacci(n-1) + Fibonacci(n-2)*.

Ejemplo de la serie: **0, 1, 1, 2, 3, 5, 8, 13, . . .**

Diagrama de flujo 3.13



Donde: `I` es una variable de tipo entero que representa la variable de control del ciclo.

Se inicializa en 3, dado que hay dos asignaciones previas al inicio del ciclo.

`PRI`, `SEG` y `SIG` son variables de tipo entero que representan los valores que se suman para obtener el siguiente valor de la serie (`SIG`).

Programa 3.13

```
#include <stdio.h>

/* Fibonacci.
El programa calcula y escribe los primeros 50 números de Fibonacci.

I, PRI, SEG, SIG: variables de tipo entero. */

void main(void)
{
int I, PRI = 0, SEG = 1, SIG;
printf("\t %d \t %d", PRI, SEG);
for (I = 3; I< = 50; I++)
{
     SIG = PRI + SEG;
     PRI = SEG;
     SEG = SIG;
     printf("\t %d", SIG);
}
}
```

Problema PR3.7

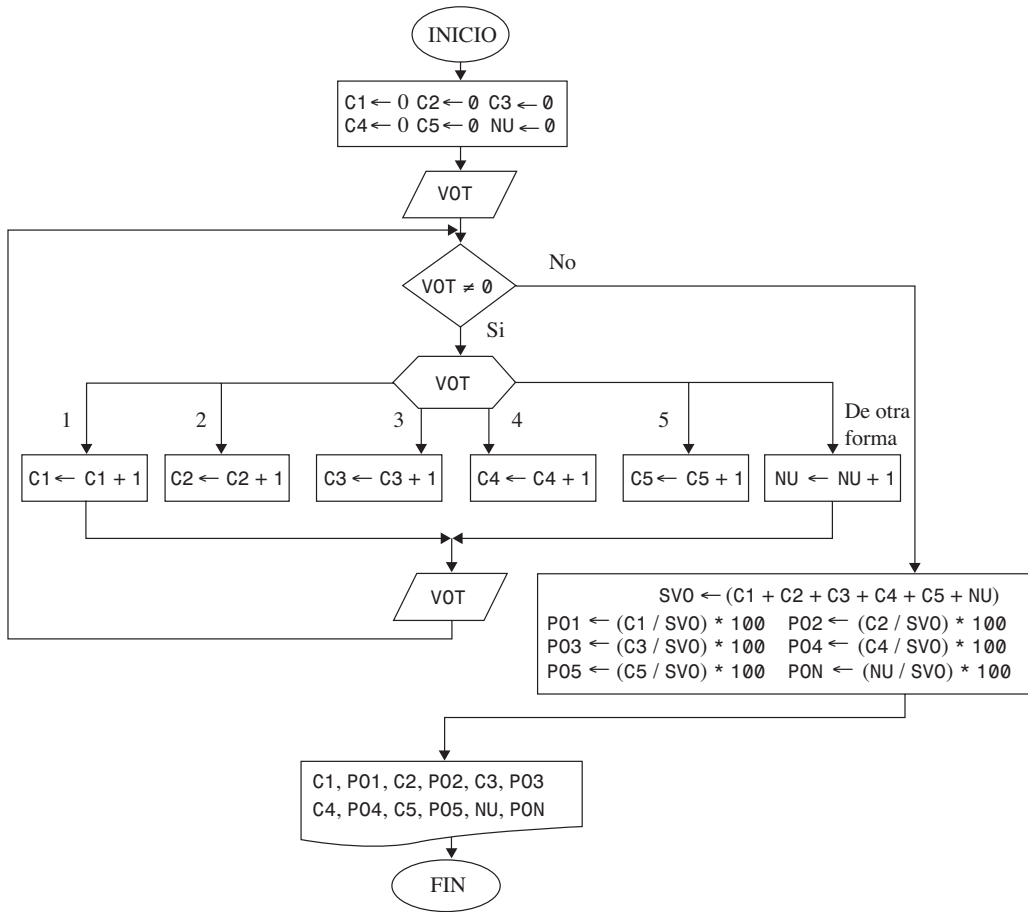
Los organizadores de un acto electoral solicitaron realizar un programa de cómputo para manejar el conteo de los votos. En la elección hay cinco candidatos, los cuales se representan con los valores comprendidos de 1 a 5. Construye un diagrama de flujo y el correspondiente programa en **C** que permita obtener el número de votos de cada candidato y el porcentaje que obtuvo respecto al total de los votantes. El usuario ingresa los votos de manera desorganizada, tal y como se obtienen en una elección, el final de datos se representa por un cero. Observa como ejemplo la siguiente lista:

```
2 5 5 4 3 4 4 5 1 2 4 3 1 2 4 5 0
```

Donde: `1` representa un voto para el candidato `1`, `3` un voto para el candidato `3`, y así sucesivamente.

Datos: `VOT1, VOT2,...,` 0 (variable de tipo entero que representa el voto a un candidato).

Diagrama de flujo 3.14



Donde: `C1, C2, C3, C4, C5` y `NU` son variables de tipo entero que funcionan como acumuladores (de los votos de los candidatos).

`SVO` es una variable de tipo entero que cuenta todos los votos registrados.

`PO1, PO2, PO3, PO4, PO5` y `PON` son variables de tipo real utilizadas para almacenar el porcentaje de votos.

Nota 4: Observa que para obtener el porcentaje dividimos entre `SVO`. Si `SVO` fuera cero, es decir si no hubiéramos ingresado ningún dato, entonces tendríamos un error ya que dividiríamos entre cero.

Programa 3.14

```
#include <stdio.h>

/* Elección.
 El programa obtiene el total de votos de cada candidato y el porcentaje
➥correspondiente. También considera votos nulos.

 VOT, C1, C2, C3, C4, C5, NU, SVO: variables de tipo entero.
 PO1, PO2, PO3, PO4, PO5, PON: variables de tipo real.*/

void main(void)
{
 int VOT, C1 = 0, C2 = 0, C3 = 0, C4 = 0,  C5 = 0, NU = 0, SVO;
 float PO1, PO2, PO3, PO4, PO5, PON;
 printf("Ingrese el primer voto: ");
 scanf("%d", &VOT);
 while (VOT)
 {
    switch(VOT)
    {
       case 1: C1++; break;
       case 2: C2++; break;
       case 3: C3++; break;
       case 4: C4++; break;
       case 5: C5++; break;
       default: NU++; break;
    }
    printf("Ingrese el siguiente voto -0 para terminar-: ");
    scanf("%d", &VOT);
 }
 SVO = C1 + C2 + C3 + C4 + C5 + NU;
 PO1 = ((float) C1 / SVO) * 100;
 PO2 = ((float) C2 / SVO) * 100;
 PO3 = ((float) C3 / SVO) * 100;
 PO4 = ((float) C4 / SVO) * 100;
 PO5 = ((float) C5 / SVO) * 100;
 PON = ((float) NU / SVO) * 100;
 printf("\nTotal de votos: %d", SVO);
 printf("\n\nCandidato 1: %d votos -- Porcentaje: %5.2f", C1, PO1);
 printf("\nCandidato 2: %d votos -- Porcentaje: %5.2f", C2, PO2);
 printf("\nCandidato 3: %d votos -- Porcentaje: %5.2f", C3, PO3);
 printf("\nCandidato 4: %d votos -- Porcentaje: %5.2f", C4, PO4);
 printf("\nCandidato 5: %d votos -- Porcentaje: %5.2f", C5, PO5);
 printf("\nNulos:       %d votos -- Porcentaje: %5.2f", NU, PON);
}
```

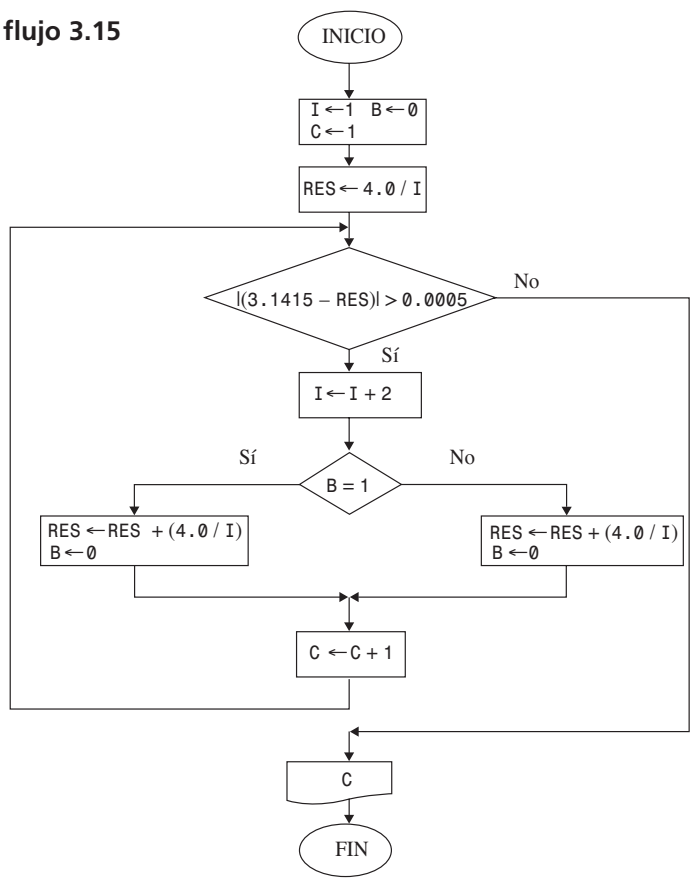
Problema PR3.8

Construye un diagrama de flujo y el correspondiente programa en **C** que calcule el valor de Π utilizando la siguiente serie:

$$\Pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \cdots$$

La diferencia entre la serie y Π debe ser menor a 0.0005. Imprime el número de términos requerido para obtener esta precisión.

Diagrama de flujo 3.15



Donde: `I` es una variable de tipo entero que incrementa de dos en dos, en cada vuelta del ciclo, el divisor. Comienza en 1.
`C` es una variable de tipo entero utilizado para contar los términos de la serie.
`B` es una variable de tipo entero utilizada para decidir si hay que sumar o restar en la serie.

RES es una variable real de doble precisión que almacena el resultado de la serie.

Programa 3.15

```
# include <stdio.h>
# include <math>

/* Cálculo de P.
El programa obtiene el valor de P aplicando una serie determinada.

I, B, C: variables de tipo entero.
RES: variable de tipo real de doble precisión. */

void main(void)
{
int I = 1, B = 0, C;
double RES;
RES = 4.0 / I;
C = 1;
while ((fabs (3.1415 - RES)) > 0.0005)
{
   I += 2 ;
   if (B)
   {
        RES += (double) (4.0 / I);
        B = 0;
   }
   else
   {
        RES -= (double) (4.0 / I);
        B = 1;
   }
   C++;
}
printf("\nNúmero de términos:%d", C);
}
```

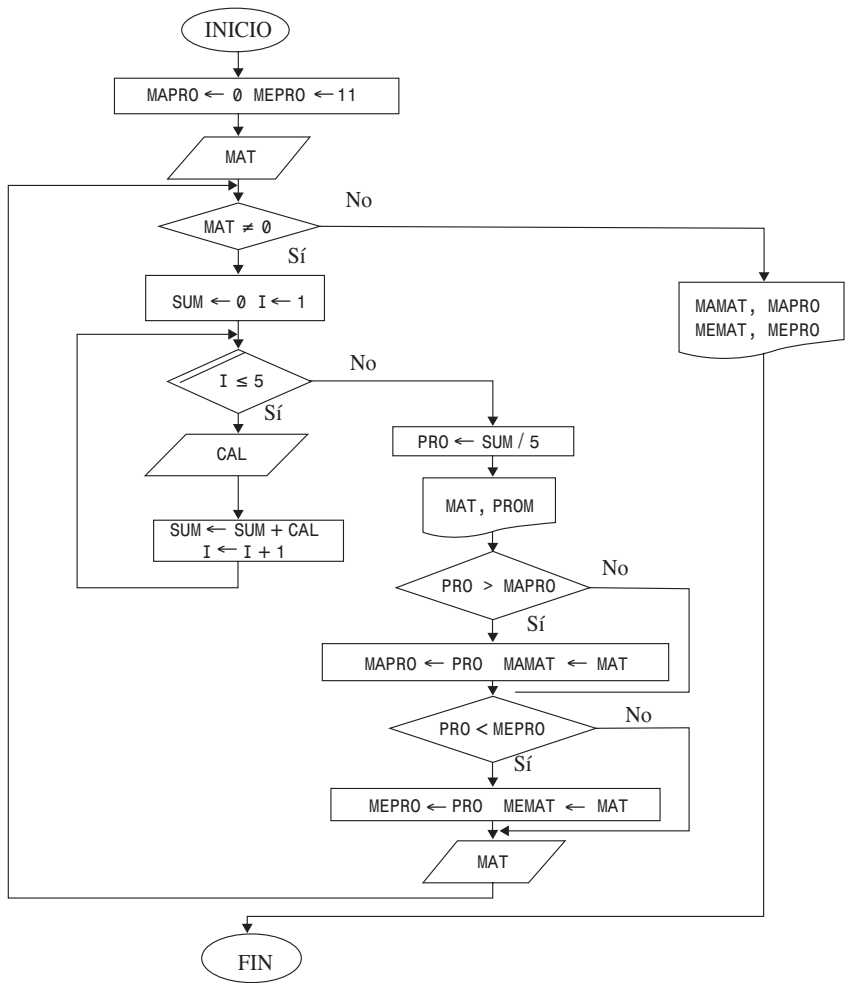
Problema PR3.9

A la clase de Estructuras de Datos del profesor López asiste un grupo numeroso de alumnos. Construye un diagrama de flujo y el correspondiente programa en **C** que imprima la matrícula y el promedio de las cinco calificaciones de cada alumno. Además, debe obtener la matrícula y el promedio tanto del mejor como del peor alumno.

Datos: MAT_1, $CAL_{1,1}$, $CAL_{1,2}$, $CAL_{1,3}$, $CAL_{1,4}$, $CAL_{1,5}$
MAT_2, $CAL_{2,1}$, $CAL_{2,2}$, $CAL_{2,3}$, $CAL_{2,4}$, $CAL_{2,5}$
. . .
0

Donde: MAT_i es una variable de tipo entero que representa la matrícula del alumno `i`.
El fin de datos está dado por 0.
$CAL_{i,j}$ es una variable de tipo real que representa la calificación `j` del alumno `i`.

Diagrama de flujo 3.16



Donde: `I` es una variable de tipo entero que representa la variable de control del ciclo interno.
`SUM` es una variable de tipo real utilizada como acumulador.

MAPRO y MEPRO son variables de tipo real utilizadas para almacenar el mejor y el peor promedio.
MAMAT y MEMAT son variables de tipo entero utilizadas para almacenar las matrículas de los alumnos con mejor y peor promedio.
PRO es una variable de tipo real utilizada para almacenar el promedio de un alumno.

Programa 3.16

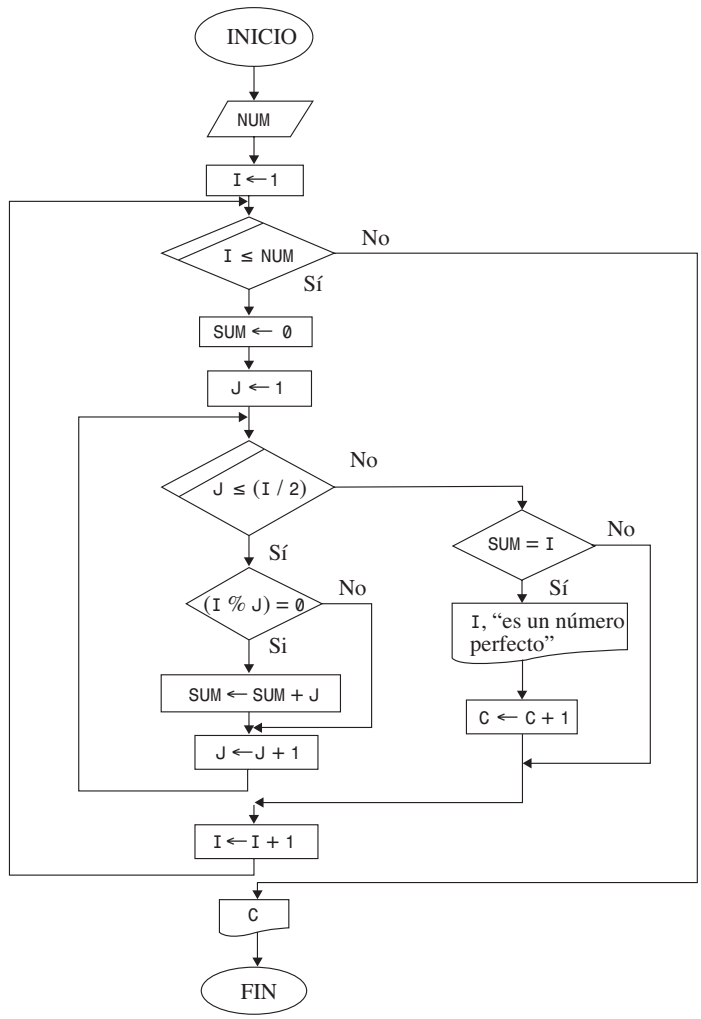
```
#include <stdio.h>

/* Calificaciones.
El programa, al recibir un grupo de calificaciones de un alumno, obtiene el promedio de calificaciones de cada uno de ellos y, además, los alumnos con el mejor y peor promedio.

 I, MAT, MAMAT y MEMAT: variables de tipo entero.
 CAL, SUM, MAPRO, MEPRO y PRO: variables de tipo real.*/

void main(void)
{
 int I, MAT, MAMAT, MEMAT;
 float SUM, PRO, CAL, MAPRO = 0.0, MEPRO = 11.0;
 printf("Ingrese la matrícula del primer alumno:\t");
 scanf("%d", &MAT);
 while (MAT)
 {
     SUM = 0;
     for (I = 1; I< = 5; I++)
     {
          printf("\tIngrese la calificación del alumno: ", I);
          scanf("%f", &CAL);
          SUM += CAL;
     }
     PRO = SUM / 5;
     printf("\nMatrícula:%d\tPromedio:%5.2f", MAT, PRO);
     if (PRO > MAPRO)
     {
          MAPRO = PRO;
          MAMAT = MAT;
     }
     if (PRO < MEPRO)
    {
          MEPRO = PRO;
          MEMAT = MAT;
     }
      printf("\n\nIngrese la matrícula del siguiente alumno: ");
      scanf("%d", &MAT);
 }
 printf("\n\nAlumno con mejor Promedio:\t%d\t\%5.2f", MAMAT, MAPRO);
 printf("\n\nAlumno con peor  Promedio:\t%d\t\%5.2f", MEMAT, MEPRO);
}
```

Problema PR3.10

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un entero positivo, escriba todos los *números perfectos* que hay entre 1 y el número dado, y que además imprima la cantidad de números perfectos que hay en el intervalo. Un número se considera perfecto si la suma de todos sus divisores es igual al propio número.

Dato: `NUM` (variable de tipo entero que representa al número límite que se ingresa).

Diagrama de flujo 3.17



Donde: I y J son variables de tipo entero que se utilizan para controlar los ciclos.
SUM es una variable de tipo entero utilizada para sumar los divisores.
C es una variable de tipo entero que representa el límite del intervalo.

Programa 3.17

```
#include <stdio.h>

/* Números perfectos.
El programa, al recibir como dato un número entero positivo como límite, obtiene
los números perfectos que hay entre 1 y ese número, y además imprime cuántos nú-
meros perfectos hay en el intervalo.

I, J, NUM, SUM, C: variables de tipo entero. */

void main(void)
{
int I, J, NUM, SUM, C = 0;
printf("\nIngrese el número límite: ");
scanf("%d", &NUM);
for (I = 1; I <= NUM; I++)
{
      SUM = 0;
      for (J = 1; J <= (I / 2); J++)
         if ((I % J) == 0)
              SUM += J;
      if (SUM == I)
      {
          printf("\n%d es un número perfecto", I);
          C++;
      }
}
printf("\nEntre 1 y %d hay %d números perfectos", NUM, C);
}
```

Problemas suplementarios

Problema PS3.1

Escribe un diagrama de flujo y el correspondiente programa en **C** que permita generar la tabla de multiplicar de un número entero positivo N, comenzando desde 1.

Dato: N (variable de tipo entero que representa el número del cual queremos obtener la tabla de multiplicar).

Problema PS3.2

Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un número entero N, calcule el resultado de la siguiente serie:

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{N}$$

Dato: N (variable de tipo entero que representa el número de términos de la serie).

Problema PS3.3

Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un número entero N, calcule el resultado de la siguiente serie:

$$1/\frac{1}{2} * \frac{1}{3}/\frac{1}{4} * \ldots (*/) \frac{1}{N}$$

Dato: N (variable de tipo entero que representa el número de términos de la serie).

Problema PS3.4

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos N números naturales, determine cuántos de ellos son positivos, negativos o nulos.

Datos: $N, NUM_1, NUM_2, \ldots, NUM_N$

Donde: N es una variable de tipo entero.

NUM_i ($1 \leq i \leq N$) es una variable de tipo entero que representa al número i.

Problema PS3.5

Construye un diagrama de flujo y el correspondiente programa en **C** que calcule e imprima la productoria de los N primeros números naturales.

$$\prod_{i=1}^{N} i$$

Dato: N (variable de tipo entero que representa el número de naturales que se ingresan).

Problema PS3.6

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos el peso, la altura y el sexo de N personas que pertenecen a un

estado de la República Mexicana, obtenga el promedio del peso (edad ≥ 18) y el promedio de la altura (edad ≥ 18), tanto de la población masculina como de la femenina.

Datos: $N, PES_1, ALT_1, SEX_1, PES_2, ALT_2, SEX_2, \ldots, PES_N, ALT_N, SEX_N$

Donde: N es una variable de tipo entero que representa el número de personas.
PES_i es una variable de tipo real que indica el peso de la persona i $(1 \leq i \leq N)$.
ALT_i es una variable de tipo real que expresa la altura de la persona i $(1 \leq i \leq N)$.
SEX_i es una variable de tipo entero que representa el sexo de la persona i $(1 \leq i \leq N)$. Se ingresa 1 si es hombre y 0 si es mujer.

Problema PS3.7

Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un número entero N, obtenga el resultado de la siguiente serie:

$$1^1 - 2^2 + 3^3 - \ldots \pm N^N$$

Dato: N (variable de tipo entero que representa el número de términos de la serie).

Problema PS3.8

Escribe un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos N valores de Y, obtenga el resultado de la siguiente función:

$$F(X) = \begin{cases} Y^2 + 15 & \text{Si } 0 < Y \leq 15 \\ Y^3 - Y^2 + 12 & \text{Si } 15 < Y \leq 30 \\ 4 * Y^3 / Y^2 + 8 & \text{Si } 30 < Y \leq 60 \\ 0 & \text{Si } 60 < Y \leq 0 \end{cases}$$

Datos: $N, Y_1, Y_2, \ldots Y_N$

Donde: N es una variable de tipo entero.
Y_i es una variable de tipo entero que indica el valor de la i-ésima Y $(1 \leq i \leq N)$.

Problema PS3.9

En el centro meteorológico ubicado en Baja California Sur, en México, llevan los registros de los promedios mensuales de temperaturas de las principales regiones del país. Existen seis regiones denominadas NORTE, CENTRO, SUR, GOLFO, PACÍFICO y CARIBE. Construye un diagrama de flujo y el correspondiente programa en **C** que obtenga lo siguiente:

a. El promedio anual de cada región.
b. El mes y registro con la mayor y menor temperaturas, y que además indique a qué zona pertenecen estos registros.
c. Determine cuál de las regiones SUR, PACÍFICO y CARIBE tienen el mayor promedio de temperatura anual.

Datos: NOR_1, CEN_1, SUR_1, GOL_1, PAC_1, CAR_1,..., NOR_{12}, CEN_{12}, SUR_{12}, GOL_{12}, PAC_{12}, CAR_{12}.

Donde: NOR_i, CEN_i, SUR_i, GOL_i, PAC_i, CAR_i son variables de tipo real que representan los promedios de temperaturas en cada una de las regiones.

Problema PS3.10

Una empresa dedicada a la venta de localidades por teléfono e Internet maneja seis tipos de localidades para un circo ubicado en la zona sur de la Ciudad de México. Algunas de las zonas del circo tienen el mismo precio, pero se manejan diferente para administrar eficientemente la asignación de los asientos. Los precios de cada localidad y los datos referentes a la venta de boletos para la próxima función se manejan de la siguiente forma:

Datos: L1, L2, L3, L4, L5 y L6
CLA_1, CAN_1
CLA_2, CAN_2
. . . . . .
0 , 0

Donde: L1, L2, L3, L4, L5 y L6 son variables de tipo real que representan los precios de las diferentes localidades.
CLA_i y CAN_i son variables de tipo entero que representan el tipo de localidad y la cantidad de boletos, respectivamente, de la venta i.

Escribe un diagrama de flujo y el correspondiente programa en **C** que realice lo siguiente:

a. Calcule el monto correspondiente de cada venta.
b. Obtenga el número de boletos vendidos para cada una de las localidades.
c. Obtenga la recaudación total.

Problema PS3.11

En una bodega en Tarija, Bolivia, manejan información sobre las cantidades producidas de cada tipo de vino en los últimos años. Escribe un diagrama de flujo y el correspondiente programa en **C** que permita calcular lo siguiente:

a. El total producido de cada tipo de vino en los últimos años.
b. El total de la producción anual de los últimos años.

Datos: N, $VIN_{1,1}$, $VIN_{1,2}$, $VIN_{1,3}$, $VIN_{1,4}$
$VIN_{2,1}$, $VIN_{2,2}$, $VIN_{2,3}$, $VIN_{2,4}$
...
$VIN_{N,1}$, $VIN_{N,2}$, $VIN_{N,3}$, $VIN_{N,4}$

Donde: N es una variable de tipo entero que representa el número de años.
$VIN_{i,j}$ es una variable de tipo real que representa la cantidad de litros de vino en el año i del tipo j ($1 \leq i \leq N$, $1 \leq j \leq 4$).

Problema PS3.12

Se dice que un número N es **primo** si los únicos enteros positivos que lo dividen son exactamente 1 y N. Construye un diagrama de flujo y el correspondiente programa en **C** que lea un número entero positivo NUM y escriba todos los números primos menores a dicho número.

Dato: NUM (variable de tipo entero que representa al número entero positivo que se ingresa).

Problema PS3.13

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos dos números enteros positivos, obtenga e imprima todos los números **primos gemelos** comprendidos entre dichos números. Los primos gemelos son una pareja de números primos con una diferencia entre sí de exactamente dos. El 3 y el 5 son primos gemelos.

Datos: N1, N2 (variables de tipo entero que representan los números enteros positivos que se ingresan).

Problema PS3.14

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato una `X` cualquiera, calcule el `sen(x)` utilizando la siguiente serie:

$$\text{sen}(x) = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

La diferencia entre la serie y un nuevo término debe ser menor o igual a 0.001. Imprima el número de términos requerido para obtener esta precisión.

Dato: `X` (variable de tipo entero que representa el número que se ingresa).

Problema PS3.15

Construye un diagrama de flujo y el correspondiente programa en **C** que calcule el **máximo común divisor** (MCD) de dos números naturales `N1` y `N2`. El MCD entre dos números es el natural más grande que divide a ambos.

Datos: `N1, N2` (variables de tipo entero que representan los números que se ingresan).

Problema PS3.16

Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como dato un número entero positivo, escriba una figura como la que se muestra a continuación (ejemplo para `N = 6`):

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Dato: `N` (variable de tipo entero que representa el número que se ingresa).

Problema PS3.17

Construye un diagrama de flujo y un programa en **C** que, al recibir como dato un número entero positivo, escriba una figura como la que se muestra a continuación (ejemplo para `N = 7`):

```
1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
1 2 3 4 5 6 5 4 3 2 1
1 2 3 4 5 6 7 6 5 4 3 2 1
```

Dato: `N` (variable de tipo entero que representa el número que se ingresa).

Problema PS3.18

Construye un diagrama de flujo y un programa en **C** que, al recibir como dato un número entero positivo, escriba una figura como la que se muestra a continuación (ejemplo para `N = 7`):

```
1 2 3 4 5 6 7     7 6 5 4 3 2 1
1 2 3 4 5 6         6 5 4 3 2 1
1 2 3 4 5             5 4 3 2 1
1 2 3 4                 4 3 2 1
1 2 3                     3 2 1
1 2                         2 1
1                             1
```

Dato: `N` (variable de tipo entero que representa el número que se ingresa).

Problema PS3.19

Construye un diagrama de flujo y el correspondiente programa en **C** que genere una figura como la que se muestra a continuación:

```
                1
              2 3 2
            3 4 5 4 3
          4 5 6 7 6 5 4
        5 6 7 8 9 8 7 6 5
      6 7 8 9 0 1 0 8 7 6
    7 8 9 0 1 2 3 2 1 0 9 8 7
  8 9 0 1 2 3 4 5 4 3 2 1 0 9 8
9 0 1 2 3 4 5 6 7 6 5 4 3 2 1 0 9
0 1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1 0
```


Problema PS3.20

Construye un diagrama de flujo y el correspondiente programa en **C** que escriba todos los valores positivos de `T`, `P` y `R` que satisfagan la siguiente expresión.

$$7*T^4 - 6*P^3 + 12*R^5 < 5850$$

Nota: `T`, `P` y `R` sólo pueden tomar valores positivos.

CAPÍTULO 4

Funciones

4.1. Introducción

Para resolver problemas complejos y/o de gran tamaño es conveniente utilizar el concepto de **reducción de problemas**. De esta forma, el problema se descompone en subproblemas, los cuales a su vez pueden descomponerse en subsubproblemas, y así continuar hasta que el problema original queda reducido a un conjunto de actividades básicas, que no se pueden o no conviene volver a descomponer. La solución de cada una de estas actividades básicas permite, luego, aplicando razonamiento hacia atrás, la solución del problema final.

En el lenguaje de programación **C** la solución de un problema se expresa por medio de un programa; la solución de un subproblema, por medio de una **función**. El uso de funciones tiene múltiples ventajas: facilitan la lectura y escritura de los programas, permiten el trabajo

en paralelo —diferentes programadores se pueden encargar de diferentes funciones—, facilitan la asignación de responsabilidades, permiten que el código de la función se escriba solamente una vez y se utilice tantas veces como sea necesario, facilitan el mantenimiento de los programas, etc.

De esta manera, un programa en **C** está constituido por un programa principal y un conjunto de funciones. El programa principal consta generalmente de pocas líneas, las cuales pueden ser llamadas a funciones. La llamada a una función indica al procesador que debe continuar con el procesamiento de la función. Una vez que ésta concluye, el control regresa al punto de partida en el programa principal. Por otra parte, la función se escribe de forma similar al programa principal, pero con diferencias principalmente en el encabezado de la misma. Una función resuelve un subproblema de forma independiente y se ejecuta sólo cuando recibe una llamada desde el programa principal o desde otras funciones. El lenguaje de programación **C** permite que una función pueda incorporar llamadas a otras funciones.

La comunicación entre las funciones y el programa principal, al igual que entre las mismas funciones, se lleva a cabo por medio de **parámetros por valor, parámetros por referencia** y **variables globales**. Estas últimas son menos utilizadas por razones de eficiencia y seguridad en la escritura de programas. En los dos últimos capítulos utilizamos funciones que pertenecen a bibliotecas del lenguaje **C**. Por ejemplo, la función `pow` que se encuentra en la biblioteca `math.h` se ha utilizado en numerosas ocasiones. Pero cabe aclarar que las funciones que utilizaremos en este capítulo son diferentes, porque tienen la particularidad de que nosotros mismos las desarrollaremos. Es decir, no se encuentran en ninguna biblioteca del lenguaje **C**.

4.2. Variables locales, globales y estáticas

Las **variables** son objetos que pueden cambiar su valor durante la ejecución de un programa. En el lenguaje de programación **C** podemos distinguir entre tres tipos de variables: **locales**, **globales** y **estáticas**. Las **variables locales** son objetos definidos tanto en el programa principal como en las funciones y su alcance está limitado solamente al programa principal o a la función en la cual están definidas. Por ejemplo, cada variable local definida en una función comienza a *existir* sólo cuando se llama a esa función, y *desaparece* cuando el control regresa al

programa principal. Puesto que no retienen su valor, deben ser inicializadas cada vez que se ejecuta la función, de otra forma contendrán basura.

C permite además definir variables locales a un bloque —conjunto de instrucciones—, las cuales desaparecen luego de ejecutar el bloque. Cabe destacar que las variables locales tienen prioridad sobre las variables globales. Es decir, si tenemos una variable global entera `I` y una variable local entera también denominada `I` en una función, cada vez que utilicemos la variable en la función estaremos haciendo referencia a la variable local. Incluso una variable local definida en un bloque tiene prioridad sobre una variable local definida obviamente con el mismo nombre en la misma función.

Por otra parte, las **variables globales** son objetos definidos antes del inicio del programa principal y su alcance es muy amplio, ya que tiene influencia tanto en el programa principal como en todas las funciones.

Finalmente, las **variables estáticas** son similares a las locales, pero conservan su valor durante la ejecución del programa. Es decir, comienzan a *existir* cuando se llama a la función y conservan su valor aun cuando el control regresa al programa principal.

En el lenguaje **C** una **función** se escribe de la siguiente forma:

```
/* El conjunto de instrucciones muestra la sintaxis de una
   ➥función en el lenguaje C. */
tipo-de-resultado Nombre-de-función (Parámetros)
{
    instrucciones;      /* cuerpo de la función. */
}
```

Donde: **`tipo-de-resultado`** representa el tipo de resultado que devuelve la función (entero, real, caracter, cadena de caracteres, etc.); si no regresa ningún resultado, entonces se escribe la palabra reservada **`void`**.

`Nombre-de-función` representa el nombre de la función. Es conveniente utilizar un nombre representativo de lo que realiza la función.

`Parámetros` se utiliza para indicar la lista de parámetros que recibe la función (los analizaremos en la sección 4.3).

`instrucciones` representa, como su nombre lo indica, al conjunto de instrucciones que pertenecen a la función

A continuación presentamos diferentes ejemplos para clarificar los conceptos anteriores. Cabe destacar que a partir de este capítulo, por cuestiones básicamente didácticas y dando por sentado que ya conoces y puedes desarrollar diagramas de flujo, omitiremos el uso de éstos en la solución gráfica de los diferentes problemas.

EJEMPLO 4.1

Escribe un programa que calcule, utilizando una función, el cubo de los 10 primeros números naturales.

Programa 4.1

```
#include <stdio.h>

/* Cubo-1.
El programa calcula el cubo de los 10 primeros números naturales con la
ayuda de una función. En la solución del problema se utiliza una variable
global, aunque esto, como veremos más adelante, no es muy recomendable. */

int cubo(void);          /* Prototipo de función. */
int I;                   /* Variable global. */

void main(void)
{
int CUB;
for (I = 1; I <= 10; I++)
{
    CUB = cubo();        /* Llamada a la función cubo. */
    printf("\nEl cubo de %d es:  %d", I, CUB);
}
}

int cubo(void)           /* Declaración de la función. */
/* La función calcula el cubo de la variable global I. */
{
return (I*I*I);
}
```

Observa que la segunda instrucción del programa:

```
int cubo(void);
```

es un prototipo de la función. Puesto que esta función se declara posteriormente, pero se utiliza en el programa principal, el compilador del lenguaje **C** requiere estar enterado de la existencia de la misma. Toma nota de que el tipo de la función es entero (`int`) y no tiene parámetros. Las funciones pueden regresar resultados de diferentes tipos de datos o bien no regresar ninguno, en cuyo caso al inicio se debe incluir la palabra reservada **`void`**.

La línea

```
int I;
```

indica que `I` es una variable global de tipo entero. Observa que la variable global se definió antes del programa principal.

La instrucción

```
CUB = cubo();
```

expresa que el resultado de la función `cubo()` se asigna a la variable local `CUB`. Observa la forma de invocar o llamar a la función. Cabe destacar que es posible evitar el uso de la variable local `CUB` escribiendo el resultado del llamado a la función cubo como se muestra a continuación:

```
printf("\nEl cubo de %d es: %d", cubo());
```

Finalmente, ya en la función cubo, la instrucción:

```
return (I*I*I);
```

regresa el resultado obtenido al programa principal. Siempre que la función es de un tipo de datos determinado es necesario utilizar un **`return`** para regresar tanto el resultado como el control al programa principal.

A continuación se muestra el resultado que arroja el programa:

```
El cubo de 1 es:      3
El cubo de 2 es:      8
El cubo de 3 es:     27
El cubo de 4 es:     64
El cubo de 5 es:    125
El cubo de 6 es:    216
El cubo de 7 es:    343
El cubo de 8 es:    512
El cubo de 9 es:    729
El cubo de 10 es:  1000
```

EJEMPLO 4.2

Observa lo que sucede en la solución del problema anterior al declarar una variable local en la función cubo.

Programa 4.2

```
#include <stdio.h>

/* Cubo-2.
El programa calcula el cubo de los 10 primeros números naturales con la
ayuda de una función. */

int cubo(void);            /* Prototipo de función. */
int I;                     /* Variable global. */

void main(void)
{
  int CUB;
  for (I = 1; I <= 10; I++)
  {
       CUB = cubo();       /* Llamada a la función cubo. */
       printf("\nEl cubo de %d es:  %d", I, CUB);
  }
}

int cubo(void)             /* Declaración de la función. */
/* La función calcula el cubo de la variable local I. */
{
  int I = 2;               /* Variable local entera I con el mismo nombre
➥que la variable global. */
  return (I*I*I);
}
```

El resultado que arroja el programa es el siguiente:

```
El cubo de 1 es:   8
El cubo de 2 es:   8
El cubo de 3 es:   8
El cubo de 4 es:   8
El cubo de 5 es:   8
El cubo de 6 es:   8
El cubo de 7 es:   8
```

```
El cubo de 8 es:   8
El cubo de 9 es:   8
El cubo de 10 es:  8
```

La variable local `I` tiene prioridad sobre la variable global que tiene el mismo nombre y por esta razón siempre calcula el cubo del número entero 2.

4.2.1. Conflicto entre los nombres de las variables

Las variables locales, tal y como lo analizamos anteriormente, tienen prioridad sobre las globales que cuentan con el mismo nombre. Es decir, si existe una variable global entera `I` y una variable local entera con el mismo nombre, cada vez que utilicemos la variable en la función estaremos haciendo referencia a la variable local. Sin embargo, puede ocurrir que en algunos casos necesitemos hacer referencia a la variable global. En esos casos debemos incorporarle previamente a la variable global el símbolo **`::`**, de tal forma que si queremos hacer referencia a la variable global `I`, debemos escribir: **`::I`**.

EJEMPLO 4.3

Observemos a continuación el siguiente programa, el cual utiliza en una misma función dos variables con el mismo nombre, una local y otra global.

Programa 4.3

```
#include <stdio.h>

/* Conflicto de variables con el mismo nombre. */

void f1(void);              /* Prototipo de función. */
int K = 5;                 /* Variable global. */

void main (void)
{
  int I;
  for (I = 1; I <= 3; I++)
      f1();
}

void f1(void)
```

```
/* La función utiliza tanto la variable local I como la variable
➥global I. */
{
  int K = 2;                /* Variable local. */
  K += K;
  printf("\n\nEl valor de la variable local es: %d", K);
  ::K = ::K + K;            /* Uso de ambas variables. */
  printf("\nEl valor de la variable global es: %d", ::K);
}
```

Los resultados que arroja el programa son los siguientes:

```
El valor de la variable local es:    4
El valor de la variable global es:   9

El valor de la variable local es:    4
El valor de la variable global es:  13

El valor de la variable local es:    4
El valor de la variable global es:  17
```

A continuación se presenta un ejemplo en el cual se utilizan variables locales, globales y estáticas.

EJEMPLO 4.4

En el siguiente programa podemos observar el uso de variables locales, globales y estáticas. Incluso en la función `f4` se hace referencia tanto a la variable local como a la variable global que tienen el mismo nombre.

Programa 4.4

```
#include <stdio.h>
/* Prueba de variables globales, locales y estáticas.
El programa utiliza funciones en las que se usan diferentes tipos de
➥ variables.  */
int f1(void);
int f2(void);
```

```
int f3(void);                        /* Prototipos de funciones. */
int f4(void);

int K = 3;                           /* Variable global. */

void main(void)
{
int I;
for (I = 1; I <= 3; I++)
{
       printf("\nEl resultado de la función f1 es: %d", f1());
       printf("\nEl resultado de la función f2 es: %d", f2());
       printf("\nEl resultado de la función f3 es: %d", f3());
       printf("\nEl resultado de la función f4 es: %d", f4());
}
}

int f1(void)
/* La función f1 utiliza la variable global. */
{
K += K;
return (K);
}

int f2(void)
/* La función f2 utiliza la variable local. */
{
int K = 1;
K++;
return (K);
}

int f3(void)
/* La función f3 utiliza la variable estática. */
{
static int K = 8;
K += 2;
return (K);
}

int f4(void)
/* La función f4 utiliza dos variables con el mismo nombre: local
➥ y global. */
{
int K = 5;
K = K + ::K;        /* Uso de la variable local (K) y global (::K) */
return (K);
}
```

Los resultados que arroja el programa son los siguientes:

```
El resultado de la función f1 es:  6
El resultado de la función f2 es:  2
El resultado de la función f3 es:  10
El resultado de la función f4 es:  11

El resultado de la función f1 es:  12
El resultado de la función f2 es:  2
El resultado de la función f3 es:  12
El resultado de la función f4 es:  17

El resultado de la función f1 es:  24
El resultado de la función f2 es:  2
El resultado de la función f3 es:  14
El resultado de la función f4 es:  29
```

4.3. Parámetros por valor y por referencia

La comunicación entre las funciones y el programa principal, o bien entre las mismas funciones, se lleva a cabo mediante **variables globales** y **parámetros por valor** y **por referencia**. El uso de variables se estudió ampliamente en la sección anterior, y en ésta nos concentraremos en los parámetros por valor y por referencia.

Los **parámetros por valor** permiten pasar datos entre el programa principal y las funciones, e incluso entre las mismas funciones. En el parámetro se escribe una copia de la variable original. Si el parámetro sufre una alteración en la función que lo recibe, la variable original no se ve afectada.

Los **parámetros por referencia** también permiten la comunicación entre el programa principal y las funciones, o entre las mismas funciones. Sin embargo, en este caso, en lugar de escribir una copia de la variable en el parámetro se escribe la dirección de la misma. Si el parámetro sufre una alteración en la función que lo recibe, la variable original también se ve afectada. En **C,** las llamadas *por referencia* se realizan mediante **apuntadores**. Un apuntador es una variable que contiene la dirección de otra variable y se representa por medio de operadores de dirección (`&`) e indirección (`*`).

Observemos a continuación diferentes ejemplos para clarificar estos conceptos.

EJEMPLO 4.5

Escribe un programa que calcule el cubo de los 10 primeros números naturales, utilizando una función y realizando la comunicación mediante parámetros por valor.

Programa 4.5

```
#include <stdio.h>

/* Cubo-3.
El programa calcula el cubo de los 10 primeros números naturales con la
➥ ayuda de una función y utilizando parámetros por valor.

int cubo(int);          /* Prototipo de función. El parámetro es de
                        ➥ tipo entero. */

void main(void)
{
 int I;
 for (I = 1; I <= 10; I++)
   printf("\nEl cubo de I es:%d", cubo(I));
   /* Llamada a la función cubo. El paso del parámetro es por valor. */
}

int cubo(int K)         /* K es un parámetro por valor de tipo entero. */
/* La función calcula el cubo del parámetro K. */
{
 return (K*K*K);
}
```


Observa que la instrucción:

```
int cubo(int);
```

es un prototipo de función que informa al compilador que el parámetro que se utilizará es por valor y de tipo entero.

La siguiente instrucción permite escribir el resultado de la función. Observa que el parámetro que se utiliza es una copia de la variable `I`.

```
printf("\nEl cubo de %d es: %d", cubo(I));
```

Finalmente, ya en la función cubo, la instrucción:

```
return (K*K*K);
```

regresa el resultado obtenido al programa principal.

EJEMPLO 4.6

Observemos a continuación el siguiente programa, el cual utiliza ahora parámetros por referencia.

Programa 4.6

```
#include <stdio.h>

/* Prueba de parámetros por referencia.  */

void f1(int *);
/* Prototipo de función. El parámetro es de tipo entero y por referencia
—observa el uso del operador de indirección. */

void main(void)
{
int I, K = 4;
for (I = 1; I <= 3; I++)
{
    printf("\n\nValor de K antes de llamar a la función: %d", ++K);
    printf("\nValor de K después de llamar a la función: %d", f1(&K));
    /* Llamada a la función f1. Se pasa la dirección de la variable K,
➥ por medio del operador de dirección: &.  */
}
}

void f1(int *R)
/* La función f1 recibe un parámetro por referencia. Cada vez que el
➥ parámetro se utiliza en la función debe ir precedido por el operador de
➥ indirección. */
{
 *R += *R;
}
```

La instrucción:

```
void f1(int *);
```

es un prototipo de función que informa al compilador que el parámetro que se va a utilizar es por referencia y de tipo entero. Se utiliza el operador de indirección: *.

La siguiente instrucción permite escribir el resultado de la función. Observa que en la llamada a la función se utiliza un parámetro por referencia. La dirección de la variable se pasa mediante el operador de dirección: **&**.

```
printf("\nValor de K después de llamar a la función: %d", f1(&K));
```

Finalmente, ya en la función `f1`, cada vez que se utiliza el parámetro por referencia, se debe anteponer al mismo el operador de indirección. El resultado que arroja el programa es el siguiente:

```
Valor de K antes de llamar a la función:     5
Valor de K después de llamar a la función:  10

Valor de K antes de llamar a la función:    11
Valor de K después de llamar a la función:  22

Valor de K antes de llamar a la función:    23
Valor de K después de llamar a la función:  46
```


EJEMPLO 4.7

Analicemos a continuación el mismo programa del ejemplo anterior, pero utilizando ahora parámetros por valor en lugar de parámetros por referencia.

Programa 4.7

```
#include <stdio.h>

/* Prueba de parámetros por valor.  */

int f1 (int);       /* Prototipo de función. El parámetro es por valor
                    ➥ y de tipo entero. */

void main(void)
{
int I, K = 4;
for (I = 1; I <= 3; I++)
{
```

```
      printf("\n\nValor de K antes de llamar a la función:   %d", ++K);
      printf("\nValor de K después de llamar a la función: %d", f1(K));
      /* Llamada a la función f1. Se pasa una copia de la variable K. */
}
}

int f1(int R)
{
 R += R;
 return (R);
}
```

El resultado que arroja el programa es el siguiente:

```
Valor de K antes de llamar a la función:    5
Valor de K después de llamar a la función:  10

Valor de K antes de llamar a la función:    6
Valor de K después de llamar a la función:  12

Valor de K antes de llamar a la función:    7
Valor de K después de llamar a la función:  14
```

A continuación presentamos un programa en el que se combinan variables locales y globales, y parámetros tanto por valor como por referencia.

EJEMPLO 4.8

Analiza cuidadosamente el siguiente programa e indica qué imprime.

Programa 4.8

```
#include <stdio.h>

/* Combinación de variables globales y locales, y parámetros por valor
➥ y por referencia. */

int a, b, c, d;                /* Variables globales. */

void funcion1(int *, int *);
/* Prototipo de función. Observa que los dos parámetros son por
➥ referencia. */
```

```
int funcion2(int, int *);
/* En este prototipo el primer parámetro es por valor y el segundo por
➥ referencia. */

void main(void)
{
int a;       /*  Nota que a es una variable local. */
a = 1;       /*  Se asigna un valor a la variable local a. */
b = 2;       /*  Se asignan valores a las variables globales b, c y d. */
c = 3;
d = 4;
printf("\n%d %d %d %d", a, b, c, d);
funcion1 (&b, &c);
printf("\n%d %d %d %d", a, b, c, d);
a = funcion2(c, &d);
printf("\n%d %d %d %d", a, b, c, d);
}

void funcion1(int *b, int *c)
{
int d;
a = 5;       /*  Observa que se hace referencia a la variable global a. */
d = 3;       /*  Nota que se hace referencia a la variable local d.  */
(*b)++;
(*c) += 2;
printf("\n%d %d %d %d", a, *b, *c, d);
}

int funcion2(int c, int *d)
{
int b;
a++;
b = 7;
c += 3;
(*d) += 2;
printf("\n%d %d %d %d", a, b, c, *d);
return (c);
}
```

Compara tu resultado con el que presentamos a continuación.

```
1        2        3        4
5        3        5        3
1        3        5        4
6        7        8        6
8        3        5        6
```

4.4. Paso de funciones como parámetros

En la práctica encontramos problemas cuyas soluciones se podrían formular fácilmente si pudiéramos pasar funciones como parámetros. Por fortuna, en el lenguaje de programación **C** es posible realizar esta actividad, es decir, pasar una función a otra función como parámetro por referencia —en el parámetro se escribe la dirección de la función—. Debemos recordar que en **C** las llamadas *por referencia* se llevan a cabo por medio de *apuntadores*, y un apuntador no es más que una variable que contiene la dirección de otra variable o de una función y se representa por medio de los operadores de dirección (`&`) e indirección (`*`). Este paso de una función como parámetro permite que una función se transfiera a otra como si fuera simplemente una variable.

EJEMPLO 4.9

En el siguiente programa pasaremos una función a otra función como parámetro por referencia. Nota que la función `Control` recibe como parámetro una función. Dependiendo de cuál sea ésta, realiza la llamada a la función correspondiente `Suma` o `Resta`.

Programa 4.9

```
#include <stdio.h>

/* Paso de una función como parámetro por referencia. */

int Suma(int X, int Y)
/* La función Suma regresa la suma de los parámetros de tipo entero
➥ X y Y. */
{
return (X+Y);
}

int Resta(int X, int Y)
/* Esta función regresa la resta de los parámetros de tipo entero
➥ X y Y. */
{
return (X-Y);
}
```

```
int Control(int (*apf) (int, int), int X, int Y)
/* Esta función recibe como parámetro otra función —la dirección— y
➥ dependiendo de cuál sea ésta, llama a la función Suma o Resta. */
{
int RES;
RES = (*apf) (X, Y);         /* Se llama a la función Suma o Resta. */
return (RES);
}

void main(void)
{
int R1, R2;
R1 = Control(Suma, 15, 5);  /* Se pasa como parámetro la función Suma. */
R2 = Control(Resta, 10, 4); /* Se pasa como parámetro la función Resta.*/
printf("\nResultado 1: %d", R1);
printf("\nResultado 2: %d", R2);
}
```

Problemas resueltos

Problema PR4.1

Escribe un programa en **C** que, al recibir como datos dos números enteros, determine si el segundo número es múltiplo del primero.

Datos: `NU1`, `NU2` (variables de tipo entero que representan los números que se ingresan).

Programa 4.10

```
# include <stdio.h>

/* Múltiplo.
El programa, al recibir como datos dos números enteros, determina si
➥ el segundo es múltiplo del primero. */

int multiplo(int, int);       /* Prototipo de función. */

void main(void)
{
int NU1, NU2, RES;
printf("\nIngresa los dos números: ");
scanf("%d %d", &NU1, &NU2);
RES = multiplo(NU1, NU2);
```

```
if (RES)
   printf("\nEl segundo número es múltiplo del primero");
else
   printf("\nEl segundo número no es múltiplo del primero");
}

int multiplo(int N1, int N2)
/* Esta función determina si N2 es múltiplo de N1. */
{
int RES;
if ((N2 % N1) == 0)
    RES = 1;
else
    RES = 0;
return (RES);
}
```

Problema PR4.2

Escribe un programa en **C** que, al recibir como dato un número entero positivo, determine el mayor divisor de dicho número.

Dato: `NUM` (variable de tipo entero que representa el número que se ingresa).

Programa 4.11

```
# include <stdio.h>

/* Mayor divisor.
El programa, al recibir como dato un número entero positivo, calcula
➥ su mayor divisor. */

int mad(int);                /* Prototipo de función. */

void main(void)
{
int NUM, RES;
printf("\nIngresa el número: ");
scanf("%d", &NUM);
RES = mad(NUM);
printf("\nEl mayor divisor de %d es: %d", NUM, RES);
}

int mad(int N1)
/* Esta función calcula el mayor divisor del número N1. */
{
```

```
int I = (N1 / 2);
/* I se inicializa con el máximo valor posible que puede ser divisor
➥ de N1. */
while (N1 % I)
/* El ciclo se mantiene activo mientras (N1 % I) sea distinto de cero.
➥ Cuando el resultado sea 0, se detiene, ya que se habrá encontrado
➥ el mayor divisor de N1. */
    I--;
return I;
}
```

Problema PR4.3

Escribe un programa en **C** que, al recibir como datos dos números enteros, determine el máximo común divisor de dichos números.

Datos: `NU1`, `NU2` (variables de tipo entero que representan los números que se ingresan).

Programa 4.12

```
# include <stdio.h>

/* Máximo común divisor.
El programa, al recibir como datos dos números enteros, calcula el máximo
➥ común divisor de dichos números. */

int mcd(int, int);

void main(void)
{
int NU1, NU2, RES;
printf("\nIngresa los dos números enteros: ");
scanf("%d %d", &NU1, &NU2);
RES = mcd (NU1, NU2);
printf("\nEl máximo común divisor de %d y %d es: %d", NU1, NU2, RES);
}

int mcd(int N1, int N2)
/* Esta función calcula el máximo común divisor de los números N1
➥ y N2. */
{
int I;
if (N1 < N2)
   I = N1 / 2;
```

```
else
   I = N2 / 2;
/* I se inicializa con el máximo valor posible que puede ser divisor
➥ de N1 y N2. */
while ((N1 % I) || (N2 % I))
/* El ciclo se mantiene activo mientras (N1 % I) o (N2 % I) sean
➥ distintos de cero. Cuando el resultado de la evaluación sea 0, el
➥ ciclo se detiene ya que se habrá encontrado el máximo común divisor. */
   I--;
return I;
}
```

Problema PR4.4

Escribe un programa en **C** que, al recibir como datos `N` números enteros, determine cuántos de estos números son pares y cuántos impares.

Datos: `N`, NUM_1, NUM_2,..., NUM_N

Donde: `N` es una variable de tipo entero que representa el número de datos que se ingresan.
NUM_i es una variable de tipo entero que representa al número `i (1 ≤ i ≥ N)`.

Programa 4.13

```
#include <stdio.h>
#include <math.h>

/* Pares e impares.
El programa, al recibir como datos N números enteros, calcula cuántos
➥ de ellos son pares y cuántos impares, con la ayuda de una función. */

void parimp(int, int *, int *);           /* Prototipo de función. */

void main(void)
{
int I, N, NUM, PAR = 0, IMP = 0;
printf("Ingresa el número de datos: ");
scanf("%d", &N);
for (I =1; I <= N; I++)
{
   printf("Ingresa el número %d:", I);
   scanf("%d", &NUM);
   parimp(NUM, &PAR, &IMP);
```

```
   /* Llamada a la función. Paso de parámetros por valor y por
➥ referencia. */
}
printf("\nNúmero de pares: %d", PAR);
printf("\nNúmero de impares: %d", IMP);
}

void parimp(int NUM, int *P, int *I)
/* La función incrementa el parámetro *P o *I, según sea el número par
➥ o impar. */
{
int RES;
RES =  pow(-1, NUM);
if (RES > 0)
   *P += 1;
else
   if (RES < 0)
      *I += 1;
}
```

Problema PR4.5

Escribe un programa en **C** que, al recibir las calificaciones de un grupo de alumnos que presentaron su examen de admisión para ingresar a una universidad privada en la Ciudad de México, calcule y escriba el número de calificaciones que hay en cada uno de los siguientes rangos:

```
0 . . . 3.99

4 . . . 5.99

6 . . . 7.99

8 . . . 8.99

 9 . . .  10
```

Datos: CAL_1, CAL_2, ..., `-1` (CAL_i es una variable de tipo real que representa la calificación del alumno `i`).

Programa 4.14

```
#include <stdio.h>

/* Rango de calificaciones.
```

```
El programa, al recibir como dato una serie de calificaciones, obtiene
➥ el rango en el que se encuentran.*/

void Rango(int);           /* Prototipo de función. */

int RA1 = 0;
int RA2 = 0;
int RA3 = 0;              /* Variables globales de tipo entero. */
int RA4 = 0;
int RA5 = 0;
/* El uso de variables globales no es muy recomendable. En estos
➥ problemas se utilizan únicamente con el objetivo de que el lector
➥ pueda observar la forma en que se aplican. */

void main(void)
{
  float CAL;
 printf("Ingresa la primera calificación del alumno: ");
 scanf("%f", &CAL);
 while (CAL != -1)
 {
     Rango(CAL);    /* Llamada a la función Rango. Se pasa un parámetro
                       por valor. */
     printf("Ingresa la siguiente calificación del alumno: ");
     scanf("%f", &CAL);
}
printf("\n0..3.99 = %d", RA1);
printf("\n4..5.99 = %d", RA2);
printf("\n6..7.99 = %d", RA3);
printf("\n8..8.99 = %d", RA4);
printf("\n9..10   = %d", RA5);
}

void Rango(int VAL)
/* La función incrementa una variable dependiendo del valor del
➥ parámetro VAL. */
{
if (VAL < 4)
   RA1++;
else
   if (VAL < 6)
      RA2++;
   else
      if (VAL < 8)
         RA3++;
      else
         if (VAL < 9)
            RA4++;
          else
            RA5++;
}
```

Problema PR4.6

Escribe un programa en **C** que calcule e imprima la productoria de los `N` primeros números naturales.

Dato: `NUM` (variable de tipo entero que representa el número de naturales que se ingresan, `1 ≤ NUM ≤ 100`).

Programa 4.15

```
include <stdio.h>

/* Productoria.
El programa calcula la productoria de los N primeros números naturales. */

int Productoria(int);           /* Prototipo de función. */

void main(void)
{
int NUM;

/* Se escribe un do-while para verificar que el número del cual se
➥ quiere calcular la productoria sea correcto. */
do
{
     printf("Ingresa el número del cual quieres calcular la
     ➥ productoria: ");
     scanf("%d", &NUM);
}
while (NUM >100 || NUM < 1);
printf("\nLa productoria de %d es: %d", NUM, Productoria(NUM));
}

int Productoria(int N)
/* La función calcula la productoria de N. */
{
int I, PRO = 1;
for (I = 1; I <= N; I++)
    PRO *= I;
return (PRO);
}
```


Problema PR4.7

Escribe un programa que, al recibir como datos 24 números reales que representan las temperaturas registradas en el exterior en un periodo de 24 horas, encuentre,

con la ayuda de funciones, la temperatura promedio del día, así como la temperatura máxima y la mínima con el horario en el cual se registraron.

Datos: TEM_1, TEM_2, ..., TEM_{24} (variables de tipo real que representan las temperaturas).

Programa 4.16

```
#include <stdio.h>
#include <math.h>

/* Temperaturas.
El programa recibe como datos 24 números reales que representan las
➥ temperaturas en el exterior en un período de 24 horas. Calcula el
➥ promedio del día y las temperaturas máxima y mínima con la hora en la
➥ que se registraron. */

 void Acutem(float);
 void Maxima(float, int);                 /* Prototipos de funciones. */
 void Minima(float, int);

 float ACT = 0.0;
 float MAX = -50.0;                       /* Variables globales. */
 float MIN = 60.0;
 int HMAX;
 int HMIN;
/* Variables globales. ACT se utiliza para acumular las temperaturas,
➥ por esa razón se inicializa en cero. MAX se utiliza para calcular la
➥ máxima; se inicializa en -50 para que el primer valor que se ingrese
➥ modifique su contenido. MIN se usa para calcular la mínima; se
➥ inicializa con un valor muy alto para que el primer valor ingresado
➥ modifique su contenido. HMAX y HMIN se utilizan para almacenar el
➥ horario en que se produjeron las temperaturas máxima y mínima,
➥ respectivamente. */
 void main(void)
 {
 float TEM;
 int I;
 for (I = 1; I <= 24; I++)
 {
   printf("Ingresa la temperatura de la hora %d: ", I);
   scanf("%f", &TEM);
   Acutem(TEM);
   Maxima(TEM, I);      /* Llamada a las funciones. Paso de parámetros
                        ➥ por valor. */
   Minima(TEM, I);
 }
```

```
 printf("\nPromedio del día: %5.2f", (ACT / 24));
 printf("\nMáxima del día: %5.2f  \tHora: %d", MAX, HMAX);
 printf("\nMínima del día: %5.2f   \tHora: %d", MIN, HMIN);
 }

 void Acutem(float T)
 /* Esta función acumula las temperaturas en la variable global ACT
➥ para posteriormente calcular el promedio. */
 {
  ACT += T;
 }

 void Maxima(float T, int H)
 /* Esta función almacena la temperatura máxima y la hora en que se
➥ produjo en las variables globales MAX y HMAX, respectivamente. */
 {
 if (MAX < T)
 {
    MAX = T;
    HMAX = H;
 }
 }

 void Minima(float T, int H)
 /* Esta función almacena la temperatura mínima y la hora en que se
➥ produjo en las variables globales MIN y HMIN. */
 {
 if (MIN > T)
 {
    MIN = T;
    HMIN = H;
 }
 }
```


Problema PR4.8

En el centro meteorológico ubicado en Baja California Sur, en México, se registran los promedios mensuales pluviales de las principales regiones del país. Existen seis regiones denominadas NORTE, CENTRO, SUR, GOLFO, PACÍFICO y CARIBE. Escribe un programa en **C** que obtenga lo siguiente, sólo para las regiones GOLFO, PACÍFICO y CARIBE:

a) El promedio anual de las tres regiones.

b) La región con mayor promedio de lluvia anual (considera que los promedios de lluvias son diferentes).

Datos: GOL_1, PAC_1, CAR_1, GOL_2, PAC_2, CAR_2,..., GOL_{12}, PAC_{12}, CAR_{12}

Donde: GOL_i, PAC_i y CAR_i son variables reales que representan las lluvias mensuales de las diferentes regiones $(1 \leq i \leq 12)$.

Programa 4.17

```
#include <stdio.h>

/* Lluvias.
El programa permite calcular el promedio mensual de las lluvias caídas en
➥ tres regiones importantes del país. Determina también cuál es la región
➥ con mayor promedio de lluvia anual. */

 void Mayor(float, float, float);             /* Prototipo de función. */

 void main(void)
 {
 int I;
 float GOL, PAC, CAR, AGOL = 0, APAC = 0, ACAR = 0;
 for (I =1; I <= 12; I++)
 {
     printf("\n\nIngresa las lluvias del mes %d", I);
     printf("\nRegiones Golfo, Pacífico y Caribe: ");
     scanf("%f %f %f", &GOL, &PAC, &CAR);
     AGOL += GOL;
     APAC += PAC;
     ACAR += CAR;
 }
 printf("\n\nPromedio de lluvias Región Golfo: %6.2f", (AGOL / 12));
 printf("\nPromedio de lluvias Región Pacífico: %6.2f ", (APAC / 12));
 printf("\nPromedio de lluvias Región Caribe: %6.2f \n", (ACAR / 12));
 Mayor(AGOL, APAC, ACAR);
 /* Se llama a la función Mayor. Paso de parámetros por valor. */
 }

  void Mayor(float R1, float R2, float R3)
 /* Esta función obtiene la región con mayor promedio de lluvia anual. */
 {
 if (R1 > R2)
     if (R1 > R3)
        printf("\nRegión con mayor promedio: Región Golfo. Promedio:
        ➥ %6.2f",
        R1 / 12);
    else
        printf("\nRegión con mayor promedio: Región Caribe. Promedio:
        ➥ %6.2f",
        R3 / 12);
 else
```

```
    if (R2 > R3)
        printf("\nRegión con mayor promedio: Región Pacífico. Promedio:
        ➥ %6.2f",
            R2 / 12);
    else
        printf("\nRegión con mayor promedio: Región Caribe. Promedio:
        ➥ %6.2f",
            R3 / 12);
}
```

Problema PR4.9

Escribe un programa en **C** que imprima todos los valores de `T`, `P` y `Q` que satisfagan la siguiente expresión:

$$15*T^4 + 12*P^5 + 9*Q^6 < 5500$$

Nota: Observa que las tres variables sólo pueden tomar valores enteros positivos.

Programa 4.18

```
#include <stdio.h>
#include <math.h>

/* Expresión.
El programa escribe los valores de T, P y Q que satisfacen una determinada
➥ expresión.*/

int Expresion(int, int, int);              /* Prototipo de función. */

void main(void)
{
int EXP, T = 0, P = 0, Q = 0;
EXP = Expresion(T, P, Q);
while (EXP < 5500)
{
   while (EXP < 5500)
   {
      while (EXP < 5500)
      {
         printf("\nT: %d,  P: %d,  Q: %d, Resultado: %d", T, P, Q, EXP);
         Q++;
         EXP = Expresion(T, P, Q);
      }
      P++;
      Q = 0;
```

```
       EXP = Expresion(T, P, Q);
      }
      T++;
      P = 0;
      Q = 0;
      EXP = Expresion(T, P, Q);
}
}

int Expresion(int T, int P, int Q)
/* Esta función obtiene el resultado de la expresión para los valores
➥ de T, P y Q. */
{
int RES;
RES = 15 * pow(T,4) + 12 * pow(P,5) + 9 * pow(Q,6);
return (RES);
}
```

Problema PR4.10

Sigue y analiza cuidadosamente el siguiente programa e indica qué imprime. Si tus resultados coinciden con los presentados, felicitaciones. Si son diferentes, revisa principalmente la aplicación de los parámetros por referencia, porque tal vez hay algún concepto que aún no dominas.

Programa 4.19

```
#include <stdio.h>

/* Funciones y parámetros. */

int a, b, c, d;                        /* Variables globales. */

void funcion1(int, int *, int *);      /* Prototipos de funciones. */
int funcion2(int *, int);

void main(void)
{
int a;
a = 1;
b = 2;
c = 3;
d = 4;
printf("\n%d %d %d %d", a, b, c, d);
a = funcion2 (&a, c);
```

```
printf("\n%d %d %d %d", a, b, c, d);
}

void funcion1(int r, int *b, int *c)
{
int d;
a = *c;
d = a + 3 + *b;
if (r)
{
     *b = *b + 2;
     *c = *c + 3;
     printf("\n%d %d %d %d", a, *b, *c, d);
}
else
{
     *b = *b + 5;
     *c = *c + 4;
     printf("\n%d %d %d %d", a, *b, *c, d);
}
}

int funcion2(int *d, int c)
{
int b;
a = 1;
b = 7;
funcion1(-1, d, &b);
/* Observa que el parámetro d que enviamos a funcion1 es por referencia.
➥ Es equivalente escribir &*d a escribir solamente d. */

printf("\n%d %d %d %d", a, b, c, *d);
c += 3;
(*d) += 2;
printf("\n%d %d %d %d", a, b, c, *d);
return (c);
}
```


El programa genera los siguientes resultados:

```
 1        2        3        4
 7        3       10       11
 7       10        3        3
 7       10        6        5
 6        2        3        4
```

Problema PR4.11

Analiza cuidadosamente el siguiente programa e indica qué imprime. Si tus resultados coinciden con los presentados, felicitaciones. Si son diferentes, revisa principalmente la aplicación de los parámetros por referencia, porque tal vez hay algún concepto que aún no dominas.

Programa 4.20

```
# include <stdio.h>

/* Funciones y parámetros. */

int F1(int , int *);               /* Prototipo de función. */

int A = 3;
int B = 7;
int C = 4;                         /* Variables globales. */
int D = 2;

void main(void)
{
A = F1 (C, &D);
printf("\n%d %d %d %d", A, B, C, D);
C = 3;
C = F1 (A, &C);
printf("\n%d %d %d %d", A, B, C, D);
}

int F1(int X, int *Y)
{
int A;
A = X * *Y;
C++;
B += *Y;
printf("\n%d %d %d %d", A, B, C, D);
*Y--;
return (C);
}
```

El programa genera los siguientes resultados:

```
  8       9         5         2
  5       9         5         2
 15      13         4         2
  5      13         4         2
```

Problema PR4.12

Sigue y analiza cuidadosamente el siguiente programa e indica qué imprime. Si tus resultados coinciden con los presentados, felicitaciones. Si son diferentes, revisa principalmente la aplicación de los parámetros por referencia, porque seguramente hay algún concepto que aún no dominas.

Programa 4.21

```
# include <stdio.h>

/* Funciones y parámetros. */

int z, y;                          /* Variables globales. */

int F1(float);
void F2(float, int *);             /* Prototipos de funciones. */

void main(void)
{
int w;
float x;
z = 5;
y = 7;
w = 2;
x = (float)y / z;
printf("\nPrograma Principal: %d %d %.2f %d", z, y, x, w);
F2 (x, &w);
printf("\nPrograma Principal: %d %d %.2f %d", z, y, x, w);
}

int F1(float x)
{
int k;
if (x!= 0)
{
     k = z - y;
     x++;
}
else
    k = z + y;
printf("\nF1: %d %d %.2f %d", z, y, x, k);
return(k);
}

void F2(float t, int *r)
{
```

```
int y;
y = 5;
z = 0;

printf("\nF2: %d %d %.2f %d", z, y, t, *r);
if (z == 0)
{
    z = (*r) * 2;
    t = (float) z  / 3;
    printf("\nIngresa el valor: ");
    scanf("%d", r);             /* El usuario debe ingresar el valor 6 */
    printf("\nF2: %d %d %.2f %d", z, y, t, *r);
}
else
{
    z = (*r) * 2;
    printf("\nF2: %d %d %.2f %d", z, y, t, *r);
}
*r = F1(t);
}
```

El programa genera los siguientes resultados:

```
 5        7      1.40         2
 0        5      1.40         2
 4        5      1.33         6
 4        7      2.33        -3
 4        7      1.40        -3
```

Problemas suplementarios

Nota: Todos los problemas deben resolverse con la ayuda de funciones. Evita utilizar variables globales, ya que no es una buena práctica.

Problema PS4.1

Escribe un programa que, al dar como datos N números enteros $(1 \leq N \leq 500)$, obtenga el promedio de los números pares e impares.

Dato: N, NUM_1, NUM_2, NUM_N

Donde: N es una variable de tipo entero que representa el número de datos
NUM_i es una variable de tipo entero que representa el número que se ingresa.

Problema PS4.2

Escribe un programa en **C** que lea un número entero NUM y calcule el resultado de la siguiente serie:

$$1 * \frac{1}{2} / \frac{1}{3} * \frac{1}{4} / \ldots (*,/) \frac{1}{N}$$

Dato: NUM (variable de tipo entero que representa el número de términos de la serie).

Problema PS4.3

Construye un diagrama de flujo y un programa en **C** que, al recibir como dato un número entero N, calcule el factorial de dicho número.

Dato: NUM (variable de tipo entero que representa el número que se ingresa).

Problema PS4.4

Un individuo invierte en un banco un capital específico y quiere saber cuánto obtendrá al cabo de cierto tiempo, si el dinero se coloca a una determinada tasa de interés mensual. Escribe el programa correspondiente.

Datos: MES, CAP, TAS

Donde: MES es una variable de tipo entero que se utiliza para indicar el número de meses al que se colocará la inversión.
CAP es una variable de tipo real que representa el monto inicial.
TAS es una variable de tipo real que indica la tasa de interés mensual.

Problema PS4.5

Escribe un programa en **C** que, al recibir como dato un número entero N, obtenga el resultado de la siguiente serie:

$$1^1 - 2^2 + 3^3 - \ldots \pm N^N$$

Dato: N (variable de tipo entero que representa el número de términos de la serie).

Problema PS4.6

Escribe un programa en **C** que, al recibir como dato una X cualquiera, calcule el `cos(x)` utilizando la siguiente serie:

$$1 - \frac{X^2}{2!} + \frac{X^4}{4!} + \frac{X^6}{6!} + \cdots$$

La diferencia entre la serie y un nuevo término debe ser menor o igual a 0.001. Imprima el número de términos requerido para obtener esta precisión.

Dato: `X` (variable de tipo entero que representa el número que se ingresa).

Problema PS4.7

Se dice que un número `N` es **primo** si los únicos enteros positivos que lo dividen son exactamente `1` y `N`. Escribe un programa en **C** que, al recibir como dato un número entero positivo, determine si éste es un número primo.

Dato: `NUM` (variable de tipo entero que representa el número entero positivo que se ingresa).

Problema PS4.8

Se dice que un número es considerado **perfecto** si la suma de sus divisores excepto el mismo, es igual al propio número. Escriba un programa que obtenga e imprima todos los números perfectos comprendidos entre `1` y `N`.

Dato: `N` (variable de tipo entero que representa el número entero positivo que se ingresa).

Problema PS4.9

Escribe un programa en **C** que, al recibir como dato un número entero de cuatro dígitos, lo imprima en forma inversa como se muestra a continuación —el número considerado es el 9256.

6 5 2 9

Dato: `N` (variable de tipo entero que representa el número entero positivo que se ingresa).

Problema PS4.10

Escribe los resultados que se obtienen al ejecutar el siguiente programa:

Programa 4.22

```
#include <stdio.h>

/* Funciones y parámetros. */

int a, b, c, d;

int pal(int, int);                          /* Prototipo de función. */

void main(void)
{
 a = 2;
 c = 3;
 d = 5;
 a = pal(c, d);
 printf("\n%d %d %d %d", a, b, c, d);
 b = 4;
 b = pal(b, a);
 printf("\n%d %d %d %d", a, b, c, d);
}

int pal(int x, int y)
{
 int c;
 b = x * y;
 c = b + y;
 x++;
 y = y * (y + 1);
 printf("\n%d %d %d %d", b, c, x, y);
 return (x);
}
```

Problema PS4.11

Examina cuidadosamente el siguiente segmento de programa:

Programa 4.23

```
#include <stdio.h>

void trueque(int *x, int *y)
{
 int tem;
 tem = *x;
 *x = *y;
 *y = tem;
}

int suma(int x)
{
 return (x + x);
}
```

y determina cuáles de las siguientes llamadas a las funciones son correctas si `x`, `y` y `z` son variables de tipo entero.

```
1. trueque(suma (&x), &x);
2. trueque(3, 4);
3. suma(10);
4. y = suma(10);
5. z = trueque(&x, &y);
6. trueque(&x, &y);
```

Problema PS4.12

Escribe los resultados que se obtienen al ejecutar el siguiente programa:

Programa 4.24

```
#include <stdio.h>

/* Parámetros y funciones. */

int f1(void);
int f2(void);                    /* Prototipos de funciones. */
```

```
int f3(void);
int f4(void);
int K = 5;

void main(void)
{
int I;
for (I = 1; I <= 4; I++)
{
    printf("\n\nEl resultado de la función f1 es: %d", f1());
    printf("\nEl resultado de la función f2 es: %d", f2());
    printf("\nEl resultado de la función f3 es: %d", f3());
    printf("\nEl resultado de la función f4 es: %d", f4());
}
}

int f1(void)
{
 K *= K
 return (K);
}

int f2(void)
{
 int K = 3;
 K++;
 return (K);
}

int f3(void)
{
 static int K = 6;
 K += 3;
 return (K);
}

int f4(void)
{
 int K = 4;
 K = K + ::K;
 return (K);
}
```

CAPÍTULO 5

Arreglos unidimensionales

5.1. Introducción

En la práctica es frecuente que enfrentemos problemas cuya solución sería muy difícil de hallar si utilizáramos **tipos simples de datos** para resolverlos. Es decir, datos que ocupan una sola casilla de memoria. Sin embargo, muchos de estos problemas se podrían resolver fácilmente si aplicáramos en cambio **tipos estructurados de datos**, los cuales ocupan un grupo de casillas de memoria y se identifican con un nombre. Los **arreglos** que estudiaremos en este capítulo constituyen un tipo estructurado de datos.

Los datos estructurados tienen varios componentes, cada uno de los cuales puede ser un tipo simple de dato o bien un tipo estructurado de dato, pero es importante recordar que los componentes del nivel más bajo de un tipo estructurado siempre serán tipos simples de datos.

Formalmente definimos un arreglo de la siguiente manera:

"Un arreglo es una colección finita, homogénea y ordenada de elementos."

Finita, porque todo arreglo tiene un límite, es decir, se debe determinar cuál es el número máximo de elementos del arreglo. **Homogénea,** porque todos los elementos del arreglo deben ser del mismo tipo. **Ordenada,** porque se puede determinar cuál es el primer elemento, cuál el segundo, y así sucesivamente.

5.2. Arreglos unidimensionales

Formalmente definimos un arreglo unidimensional de la siguiente manera:

"Un arreglo unidimensional es una colección finita, homogénea y ordenada de datos, en la que se hace referencia a cada elemento del arreglo por medio de un índice. Este último indica la casilla en la que se encuentra el elemento."

Un **arreglo unidimensional** permite almacenar N elementos del mismo tipo (enteros, reales, caracteres, cadenas de caracteres, etc.) y acceder a ellos por medio de un índice. En los arreglos unidimensionales se distinguen dos partes fundamentales: los **componentes** y el **índice**. Los componentes hacen referencia a los elementos que se almacenan en cada una de las celdas o casillas. El índice, por su parte, especifica la forma de acceder a cada uno de estos elementos. Para hacer referencia a un *componente* de un arreglo debemos utilizar tanto el *nombre del arreglo* como el *índice* del elemento. En la figura 5.1 se puede observar la representación gráfica de un arreglo unidimensional.

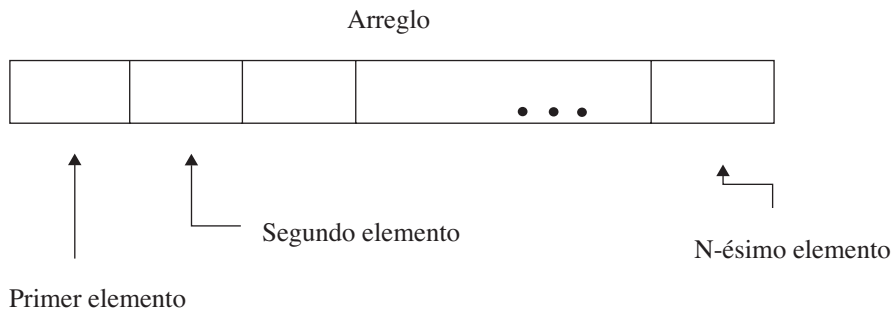


FIGURA 5.1
Representación gráfica de un arreglo unidimensional

En la figura 5.2 se muestra el *arreglo unidimensional* A que contiene 10 elementos de tipo entero. El primer índice del arreglo es el 0, el segundo, el 1, y así sucesivamente. Si queremos acceder al primer elemento del arreglo debemos escribir `A[0]`, pero si requerimos acceder al quinto elemento debemos escribir `A[4]`. Por otra parte, se puede observar que el valor de `A[7]` es 4, el de `A[3+5]` es –5, el resultado de `A[2] + A[5]` es 2, y el resultado de `A[7] * A[9]` es 32.

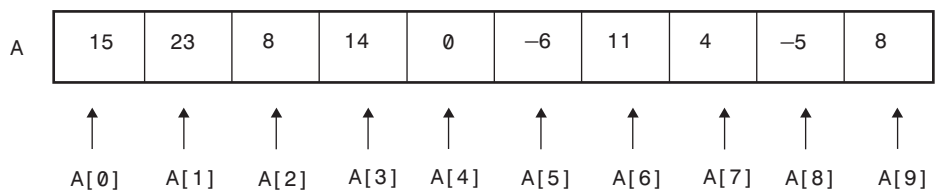


FIGURA 5.2
Índices y componentes de un arreglo unidimensional

5.3. Declaración de arreglos unidimensionales

Los arreglos ocupan espacio en memoria, que se reserva en el momento de realizar la declaración del arreglo. A continuación presentamos diferentes formas de declarar arreglos, con la explicación correspondiente.

```
void main(void)
{
 . . .
int A[10];      /* Definición de un arreglo de tipo entero de 10 elementos. */
float B[5];     /* Definición de un arreglo de tipo real de 5 elementos. */

. . .
}
```

Una vez que se definen los arreglos, sus elementos pueden recibir los valores a través de múltiples asignaciones, o bien, como ocurre frecuentemente en la práctica, a través de un ciclo. Observemos a continuación el siguiente ejemplo.

EJEMPLO 5.1

Construye un programa que, al recibir como datos un arreglo unidimensional de 100 elementos de tipo entero y un número entero, determine cuántas veces se encuentra este número dentro del arreglo.

Datos: `ARRE[100]`, `NUM` (donde `ARRE` es un arreglo unidimensional de tipo entero con capacidad para almacenar 100 valores enteros y `NUM` es una variable de tipo entero que representa el número que se buscará en el arreglo).

Programa 5.1

```
#include <stdio.h>

/* Cuenta-números.
El programa, al recibir como datos un arreglo unidimensional de tipo
➥entero y un número entero, determina cuántas veces se encuentra el
➥número en el arreglo. */

void main(void)
{
int I, NUM, CUE = 0;
int ARRE[100];                              /* Declaración del arreglo */
for (I=0; I<100; I++)
{
   printf("Ingrese el elemento %d del arreglo: ", I+1);
   scanf("%d", &ARRE[I]);            /* Lectura -asignación— del arreglo */
}
printf("\n\nIngrese el número que se va a buscar en el arreglo: ");
scanf("%d", &NUM);
for (I=0; I<100; I++)
  if (ARRE[I] == NUM)  /* Comparación del número con los elementos del
                       ➥arreglo */
       CUE++;
printf("\n\nEl %d se encuentra %d veces en el arreglo", NUM, CUE);
}
```

Otra forma de asignar valores a los componentes de un arreglo es al realizar la declaración del mismo. A continuación presentamos diferentes casos con la explicación correspondiente, junto con figuras en las que se muestran los valores que toman los diferentes componentes del arreglo.

```
. . .
int A[10] = {0};    /* Todos los componentes del arreglo se inicializan en 0. */
```

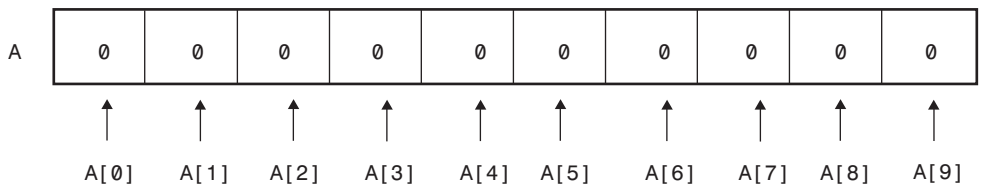


FIGURA 5.3
Declaración del arreglo unidimensional A

```
int B[5] = {5};
/* El  primer componente del arreglo se inicializa con el número 5 y el resto con
➥0. */
```

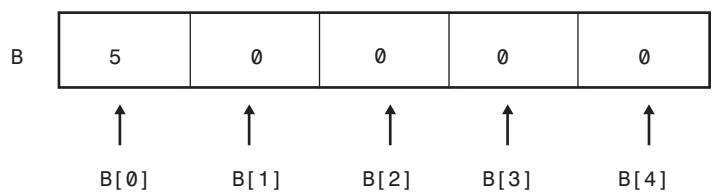


FIGURA 5.4
Declaración del arreglo unidimensional B

```
int C[5] = {6, 23, 8, 4, 11};       /* Cada componente del arreglo recibe un valor.
                                       La asignación se realiza en forma
➥consecutiva.   */
```

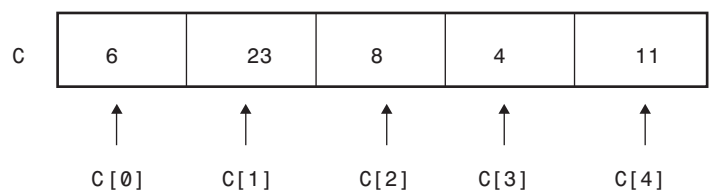


FIGURA 5.5
Declaración del arreglo unidimensional C

```
int D[5] = {6, 23, 8, 4, 11, 35};    /* Esta asignación genera un error de
➥sintaxis que se detecta en la compilación. El error ocurre porque el arreglo
➥tiene capacidad solamente para 5 componentes y hay 6 elementos que se quiere
➥asignar. */

int E[] = {33, 21, 48, 5, 11};    /* En este caso, como se omite el tamaño del
➥arreglo en la definición, se considera que el número de elementos del arreglo
➥es igual al número de elementos que existen en la lista por asignar. */
```

FIGURA 5.6
Declaración del arreglo unidimensional `E`

EJEMPLO 5.2

Los organizadores de un acto electoral de un país sudamericano solicitaron un programa de cómputo para manejar en forma electrónica el conteo de los votos. En la elección hay cinco candidatos, los cuales se representan con los valores del 1 al 5. Construye un programa en **C** que permita obtener el número de votos de cada candidato. El usuario ingresa los votos de manera desorganizada, tal y como se obtienen en una elección; el final de datos se representa con un cero. Observa la siguiente lista de ejemplo:

2 5 5 4 3 4 4 5 1 2 4 3 1 2 4 5 0

Donde: `2` representa un voto para el candidato 2, `5` un voto para el candidato 5, y así sucesivamente.

Datos: VOT_1, VOT_2, . . . , `0` (variable de tipo entero que representa el voto a un candidato).

Programa 5.2

```
#include <stdio.h>

/* Elección.
El programa almacena los votos emitidos en una elección en la que hubo cinco
➥candidatos e imprime el total de votos que obtuvo cada uno de ellos. */
```

```
void main(void)
{
int ELE[5] = {0};   /* Declaración del arreglo entero ELE de cinco
➥elementos. Todos sus elementos se inicializan en 0. */
int I, VOT;
printf("Ingresa el primer voto (0 - Para terminar): ");
scanf("%d", &VOT);
while (VOT)
{
  if ((VOT > 0) && (VOT < 6))            /* Se verifica que el voto sea
  ➥correcto. */
       ELE[VOT-1]++;       /* Los votos  se almacenan en el arreglo.
➥Recuerda que la primera posición del arreglo es 0, por esa razón a la
➥variable VOT se le descuenta 1. Los votos del primer candidato se
➥almacenan en la posición 0. */
  else
     printf("\nEl voto ingresado es incorrecto.\n");
  printf("Ingresa el siguiente voto (0 - Para terminar): ");
  scanf("%d", &VOT);
}
printf("\n\nResultados de la Elección\n");
for (I = 0; I <= 4; I++)
  printf("\nCandidato %d: %d", I+1, ELE[I]);
}
```

5.4. Apuntadores y arreglos

Un **apuntador** es una variable que contiene la dirección de otra variable y se representa por medio de los operadores de dirección (**&**) e indirección (*). El primero proporciona la dirección de un objeto y el segundo permite el acceso al objeto del cual se tiene la dirección.

Los apuntadores se utilizan mucho en el lenguaje de programación **C**, debido principalmente a que en muchos casos representan la única forma de expresar una operación determinada. Existe una relación muy estrecha entre apuntadores y arreglos. Un arreglo se pasa a una función indicando únicamente el *nombre del arreglo*, que representa el apuntador al mismo.

Para comprender mejor el concepto de apuntadores, observemos a continuación diferentes instrucciones con su explicación correspondiente. Luego de cada grupo de instrucciones se presenta una tabla en la que puedes observar los valores que van tomando las variables.

```
. . .
int X = 3, Y = 7, Z[5] = {2, 4, 6, 8, 10};
. . .
```

TABLA 5.1. Apuntadores, variables y valores

Variable	*Valor*
X	3
Y	7
Z[0]	2
Z[1]	4
Z[2]	6
Z[3]	8
Z[4]	10

```
. . .
int *IX;        /* IX representa un apuntador a un entero. */
IX = &X;        /* IX apunta a X. IX tiene la dirección de X. */
Y = *IX;        /* Y toma el valor de X. Y recibe el contenido de la
➥dirección almacenada en IX, es decir, el valor de X. Ahora Y es 3. */
*IX = 1;        /* X se modifica. Ahora X vale 1. */
. . .
```

TABLA 5.2. Apuntadores, variables y valores

Variable	*Valor*
X	1
Y	3
Z[0]	2
Z[1]	4
Z[2]	6
Z[3]	8
Z[4]	10

```
. . .
IX = &Z[2];     /* IX apunta al tercer elemento del arreglo Z. */
Y = *IX;        /* Y toma el valor de Z[2], ahora vale 6. */
*IX = 15;       /* Z[2] se modifica, ahora vale 15. */
. . .
```

TABLA 5.3. Apuntadores, variables y valores

Variable	*Valor*
X	1
Y	6
Z[0]	2
Z[1]	4
Z[2]	15
Z[3]	8
Z[4]	10

```
. . .
X = *IX + 5;       /* X se modifica, ahora vale Z[2] + 5 = 20. Recuerde que *IX
➥contiene el valor de Z[2]. */
*IX = *IX - 5;     /* Z[2] se modifica, ahora vale 10. */
. . .
```

TABLA 5.4. Apuntadores, variables y valores

Variable	*Valor*
X	20
Y	6
Z[0]	2
Z[1]	4
Z[2]	10
Z[3]	8
Z[4]	10

```
. . .
++*IX;           /* Z[2] se modifica, se incrementa en 1. Z[2] ahora vale 11. */
*IX += 1;        /* Z[2] se vuelve a modificar, ahora vale 12. Observa que
➥ambas instrucciones se pueden utilizar para hacer exactamente lo mismo. */
. . .
```

TABLA 5.5. Apuntadores, variables y valores

Variable	*Valor*
X	20
Y	6
Z[0]	2
Z[1]	4
Z[2]	12
Z[3]	8
Z[4]	10

```
. . .
X = *(IX + 1);
/* X se modifica. El apuntador IX se desplaza una posición y accede temporalmente
➥a Z[3], por lo tanto X toma este valor (8). Observa que IX no se reasigna. */
Y = *IX;         /* Y se modifica, toma el valor de Z[2] (12). */
. . .
```

TABLA 5.6. Apuntadores, variables y valores

Variable	*Valor*
X	8
Y	12
Z[0]	2
Z[1]	4
Z[2]	12
Z[3]	8
Z[4]	10

```
. . .
IX = IX + 1;
/* Observa otra forma de mover el apuntador. En este caso IX se desplaza una
➥posición, pero a diferencia del caso anterior, ahora se reasigna. IX apunta
➥ahora al cuarto elemento de Z (Z[3]). */
Y = *IX;          /* Y se modifica, toma ahora el valor de Z[3] (8). */
. . .
```

TABLA 5.7. Apuntadores, variables y valores

Variable	*Valor*
X	8
Y	12
Z[0]	2
Z[1]	4
Z[2]	12
Z[3]	8
Z[4]	10

```
. . .
IX = IX + 4;
/* IX se modifica. Observa que el apuntador se desplaza cuatro posiciones y cae
➥en una dirección que se encuentra afuera del arreglo. Esto ocasiona un error
➥que no señala el compilador de C. */
Y = *IX;          /* Y se modifica, toma el valor (basura) de una celda
➥incorrecta. Éste es un  error que no señala el compilador del lenguaje C. */
...
```

TABLA 5.8. Apuntadores, variables y valores

Variable	*Valor*
X	8
Y	**9203**
Z[0]	2
Z[1]	4
Z[2]	12
Z[3]	8
Z[4]	10

```
. . .
IX = &X;          /* IX apunta a la variable entera X. */
IX = IX + 1;      /* IX se mueve una posición y cae ahora en una celda
➥incorrecta. */
X = *IX;          /* X toma el valor (basura) de la celda a la que apunta IX. */
. . .
```

TABLA 5.9. Apuntadores, variables y valores

Variable	*Valor*
X	**20079**
Y	**9203**
Z[0]	2
Z[1]	4
Z[2]	12
Z[3]	8
Z[4]	10

EJEMPLO 5.3

En el siguiente programa se presentan todos los casos que se analizaron en la sección anterior.

Programa 5.3

```
#include <stdio.h>

/* Apuntadores, variables y valores. */

void main(void)
{
```

```
int X = 3, Y = 7, Z[5] = {2, 4, 6, 8, 10};
printf("\nX = %d \t Y = %d \t Z[0] = %d \t Z[1] = %d \t Z[3] = %d \t Z[4]
➥= %d", X, Y,

             Z[0], Z[1], Z[2], Z[3], Z[4]);

int *IX;          /* IX representa un apuntador a un entero. */
IX = &X;          /* IX apunta a X. IX tiene la dirección de X. */
Y = *IX;          /* Y toma el valor de X, ahora vale 3. */
*IX = 1;          /* X se modifica, ahora vale 1. */
printf("\nX = %d \t Y = %d \t Z[0] = %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =

             %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);

IX = &Z[2];       /* IX apunta al tercer elemento del arreglo Z. */
Y = *IX;          /* Y toma el valor de Z[2], ahora vale 6. */
*IX = 15;         /* Z[2] se modifica, ahora vale 15. */
printf("\nX = %d \t Y = %d \t Z[0] = %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =

             %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);

X = *IX + 5;      /* X se modifica, ahora vale Z[2] + 5 = 20. Recuerda que *IX
➥contiene el valor de Z[2]. */
*IX = *IX - 5;    /* Z[2] se modifica, ahora vale 10. */
printf("\nX = %d \t Y = %d \t Z[0] = %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =

             %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);

++*IX;            /* Z[2] se modifica, se incrementa en 1. Z[2] ahora vale 11. */
*IX += 1;         /* Z[2] se vuelve a modificar, ahora vale 12. */
printf("\nX = %d \t Y = %d \t Z[0] = %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =

             %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);

X = *(IX + 1);    /* X se modifica. El apuntador IX accede temporalmente a
➥Z[3], por lo tanto X toma este valor (8). Observa que IX no se reasigna */
Y = *IX;           /* Y se modifica, toma el valor de Z[2] (12). */
printf("\nX = %d \t Y = %d \t Z[0] = %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =

             %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);

IX = IX + 1;      /* IX se modifica. Observa la forma de mover el apuntador.
Ahora IX apunta al cuarto elemento de Z (Z[3]). */
Y = *IX;           /* Y se modifica, ahora vale Z[3] (8). */
printf("\nX = %d \t Y = %d \t Z[0]= %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =

             %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);
```

```
IX = IX + 4;          /* IX se modifica. Observa que el apuntador se mueve
➥4 posiciones y cae en una dirección afuera del arreglo. Esto ocasionará un
➥error. */ Y = *IX;  /* Y se modifica, toma el valor (basura) de una celda
➥incorrecta. Es un error que no señala el compilador del lenguaje C. */
printf("\nX = %d \t Y = %d \t Z[0]= %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =
              %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);

IX = &X;           /* IX apunta a la variable entera X. */
IX = IX + 1;       /* IX se mueve una posición y cae en una celda incorrecta. */
X = *IX;           /* X toma el valor (basura) de la celda a la que apunta IX.*/
printf("\nX = %d \t Y = %d \t Z[0]= %d \t Z[1] = %d \t Z[2] = %d \t Z[3] = %d
➥\t Z[4] =
              %d", X, Y, Z[0], Z[1], Z[2], Z[3], Z[4]);
}
```

5.5. Arreglos y funciones

El lenguaje de programación **C** utiliza parámetros por referencia para pasar los arreglos a las funciones. Cualquier modificación que se realice a los arreglos en las funciones afecta su valor original. En la llamada a la función sólo se debe incluir el *nombre del arreglo*, que es un apuntador. No se deben incluir los corchetes porque ocasionan un error de sintaxis.

EJEMPLO 5.4

Escribe un programa en **C** que calcule el producto de dos arreglos unidimensionales de tipo entero y almacene el resultado en otro arreglo unidimensional.

Datos: `VE1[10], VE2[10], VE3[10]` (arreglos unidimensionales de tipo entero con capacidad para 10 elementos. En `VE3` se almacena el resultado del producto de los vectores `VE1` y `VE2`).

Programa 5.4

```
#include <stdio.h>

/* Producto de vectores.
El programa calcula el producto de dos vectores y almacena el resultado
➥en otro arreglo unidimensional. */

const int MAX = 10;        /* Se define una constante para el tamaño de los
➥arreglos. */
```

```
void Lectura(int VEC[], int T);
void Imprime(int VEC[], int T);                    /* Prototipos de funciones. */
void Producto(int *X, int *Y, int *Z, int T);     /* Observa que en los
➥parámetros, para indicar que lo que se recibe es un arreglo, se puede escribir
➥VEC[] o *VEC. */

void main(void)
{
int VE1[MAX], VE2[MAX], VE3[MAX];
/* Se declaran tres arreglos de tipo entero de 10 elementos. */
Lectura(VE1, MAX);
/* Se llama a la función Lectura. Observa que el paso del arreglo a la función
➥es por referencia. Sólo se debe incluir el nombre del arreglo. */
Lectura(VE2, MAX);
Producto(VE1, VE2, VE3, MAX);
/* Se llama a la función Producto. Se pasan los nombres de los tres arreglos. */
printf("\nProducto de los Vectores");
Imprime(VE3, MAX);
}

void Lectura(int VEC[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
int I;
printf("\n");
for (I=0; I<T; I++)
{
     printf("Ingrese el elemento %d: ", I+1);
     scanf("%d", &VEC[I]);
}
}

void Imprime(int VEC[], int T)
/* La función Imprime se utiliza para imprimir un arreglo unidimensional de T
➥elementos de tipo entero. */
int I;
{
 for (I=0; I<T; I++)
     printf("\nVEC[%d]: %d", I+1, VEC[I]);
}

void Producto(int *X, int *Y, int *Z, int T)
/* Esta función se utiliza para calcular el producto de dos arreglos
➥unidimensionales de T elementos de tipo entero. */
{
int I;
for(I=0; I<T; I++)
     Z[I] = X[I] * Y[I];
}
```

EJEMPLO 5.5

En un arreglo unidimensional se almacenan las calificaciones obtenidas por un grupo de 50 alumnos en un examen. Cada calificación es un número entero comprendido entre 0 y 5. Escribe un programa que calcule, almacene e imprima la frecuencia de cada una de las calificaciones, y que además obtenga e imprima la frecuencia más alta. Si hubiera calificaciones con la misma frecuencia, debe obtener la primera ocurrencia.

Dato: `CAL[50]` (`CAL` es un arreglo unidimensional de tipo entero que almacenará las 50 calificaciones de los alumnos).

Programa 5.5

```
#include <stdio.h>

/* Frecuencia de calificaciones.
El programa, al recibir como datos las calificaciones de un grupo de 50
➥alumnos, obtiene la frecuencia de cada una de las calificaciones y además
➥escribe cuál es la frecuencia más alta. */

const int TAM = 50;

void Lectura(int *, int);
void Frecuencia(int , int, int , int);      /* Prototipos de funciones. */
void Impresion(int *, int);
void Mayor(int *, int);

void main(void)
{
int CAL[TAM], FRE[6] = {0};               /* Declaración de los arreglos. */
Lectura(CAL, TAM);                          /* Se llama a la función Lectura. */
Frecuencia(CAL, TAM, FRE, 6);
/* Se llama a la función Frecuencia, se pasan ambos arreglos. */
printf("\nFrecuencia de Calificaciones\n");
Impresión(FRE, 6);
Mayor(FRE, 6);
}

void Lectura(int VEC[], int T)
/* La función Lectura se utiliza para leer el arreglo de calificaciones. */
{
int I;
for (I=0; I<T; I++)
{
  printf("Ingrese la calificación -0:5- del alumno %d: ", I+1);
  scanf("%d", &VEC[I]);
}
```

```
}
void Impresion(int VEC[], int T)
/* La función Impresión se utiliza para imprimir el arreglo de frecuencias. */
{
int I;
for (I=0; I<T; I++)
  printf("\nVEC[%d]: %d", I, VEC[I]);
}

void Frecuencia(int A[], int P, int B[], int T)
/* Esta función calcula la frecuencia de calificaciones. */
{
int I;
for (I=0; I<P; I++)
     if ((A[I] >= 0) && (A[I] < 6))     /* Se valida que la calificación sea
        ➥correcta. */
          B[A[I]]++;     /* Observa la forma de almacenar e incrementar las
        ➥frecuencias. */
}

void Mayor(int *X, int T)
/* Esta función obtiene la primera ocurrencia de la frecuencia más alta. */
{
int I, MFRE = 0, MVAL = X[0];
for (I=1; I<T; I++)
  if (MVAL < X[I])
  {
      MFRE = I;
      MVAL = X[I];
  }
printf("\n\nMayor frecuencia de calificaciones: %d \tValor: %d", MFRE, MVAL);
}
```

Problemas resueltos

Problema PR5.1

Escribe un programa que, al recibir como dato un arreglo unidimensional de números reales, obtenga como resultado la suma del cuadrado de los números.

Dato: `VEC[100]` (arreglo unidimensional de tipo real de 100 elementos).

Programa 5.6

```
#include <stdio.h>
#include <math.h>

/* Suma-cuadrados.
El programa calcula la suma del cuadrado de los elementos de un arreglo
➥unidimensional de 100 elementos de tipo real. */

const int MAX = 100;
/* MAX se utiliza para reservar el espacio máximo que podrá ocupar el arreglo. */

void Lectura(float *, int);     /* Prototipos de funciones.  */
double Suma(float *, int);

void main(void)
{
float VEC[MAX];
double RES;
Lectura(VEC, MAX);
RES = Suma(VEC, MAX);
/* Se llama a la función Suma y se almacena el resultado en la variable RES. */
➥printf("\n\nSuma del arreglo: %.2lf", RES);
}

void Lectura(float A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo real. */
{
int I;
for (I=0; I<T; I++)
{
    printf("Ingrese el elemento %d: ", I+1);
    scanf("%f", &A[I]);
}
}

double Suma(float A[], int T)
/* La función Suma se utiliza para calcular la suma del cuadrado de los
➥componentes de un arreglo unidimensional de T elementos de tipo real. */
{
int I;
double AUX = 0.0;
for (I=0; I<T; I++)
    AUX += pow(A[I], 2);
return(AUX);
}
```

Problema PR5.2

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional *desordenado* de `N` enteros, obtenga como salida ese mismo arreglo pero sin los elementos repetidos.

Dato: `ARRE[N]` (arreglo unidimensional de tipo entero de `N` elementos, $1 \leq N \leq 100$).

Programa 5.7

```
#include <stdio.h>

/* Arreglo sin elementos repetidos.
El programa, al recibir como dato un arreglo unidimensional desordenado de N
➥elementos, obtiene como salida ese mismo arreglo pero sin los elementos
➥repetidos. */

void Lectura(int *, int);            /* Prototipos de funciones. */
void Imprime(int *, int);
void Elimina(int *, int *);
/* Observa que en el prototipo de Elimina, el segundo parámetro es por
➥referencia. Esto, porque el tamaño del arreglo puede disminuir. */

void main(void)
{
int TAM, ARRE[100];
/* Se escribe un do-while para verificar que el tamaño del arreglo que se
➥ingresa sea correcto. */
do
{
     printf("Ingrese el tamaño del arreglo: ");
    scanf("%d", &TAM);
}
while (TAM > 100  ||  TAM < 1);
Lectura(ARRE, TAM);
Elimina(ARRE, &TAM);
/* Observa que el tamaño del arreglo se pasa por referencia.*/
Imprime(ARRE, TAM);
}

void Lectura(int A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
printf("\n");
int I;
for (I=0; I<T; I++)
{
```

```
  printf("Ingrese el elemento %d: ", I+1);
  scanf("%d", &A[I]);
}
}

void Imprime(int A[], int T)
/* La función Imprime se utiliza para escribir un arreglo unidimensional, sin
➥repeticiones, de T elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
  printf("\nA[%d]: %d", I, A[I]);
}

void Elimina(int A[], int *T)
/* Esta función se utiliza para eliminar los elementos repetidos de un arreglo
➥unidimensional de T elementos de tipo entero. */
{
int I = 0, K, L;
while (I <  (*T-1))
{
   K = I + 1;
   while (K <=  (*T-1))
   {
       if (A[I] == A[K])
       {
           for (L = K; L <  (*T-1); L++)
                A[L] = A[L+1];
           *T = *T -1;
       }
       else
          K++;
   }
   I++;
}
}
```


Problema PR5.3

Escribe un programa en **C** que almacene en un arreglo unidimensional de tipo entero los primeros 100 números primos.

Programa 5.8

```
#include <stdio.h>

/* Primos.
El programa almacena en un arreglo unidimensional los primeros 100 números
➥primos. */

const int TAM = 100;

void Imprime(int, int);        /* Prototipos de funciones. */
void Primo(int, int *);

void main(void)
{
int P[TAM] = {1,2};
int FLA, J = 2, PRI = 3;
while (J <= TAM)
{
    FLA = 1;
     Primo(PRI, &FLA);       /* Se llama a la función que determina si PRI es
     ➥primo. */
    if (FLA)                 /* Si FLA es 1, entonces PRI es primo. */
    {
         P[J] = PRI;
         J++;
    }
    PRI += 2;
}
Imprime(P, TAM);
}

void Primo(int A, int *B)
/* Esta función determina si A es primo, en cuyo caso el valor de *B no se
➥altera. */
{
int DI = 3;
while (*B &&  (DI < (A / 2)))
{
   if ((A % DI) == 0)
       *B = 0;
   DI++;
}
}

void Imprime(int Primos[], int T)
/* Esta función imprime el arreglo unidimensional de números primos. */
{
int I;
for (I=0; I<T; I++)
     printf("\nPrimos[%d]:  %d", I, Primos[I]);
}
```

Problema PR5.4

Búsqueda secuencial en arreglos desordenados. La búsqueda secuencial en arreglos desordenados consiste en revisar elemento por elemento, de izquierda a derecha, hasta encontrar el dato buscado o bien hasta llegar al final del arreglo, lo que ocurra primero. Cuando el procedimiento concluye con éxito, se proporciona la posición en la cual fue encontrado el elemento. En caso contrario, se regresa a `0` para indicar que el elemento no fue localizado.

Datos: `VEC[N], ELE`

Donde: `VEC` es un arreglo unidimensional de tipo entero de `N` elementos, `1 ≤ N ≤ 100`, y `ELE` una variable de tipo entero que representa el elemento a buscar.

Programa 5.9

```
#include <stdio.h>

/* Búsqueda secuencial en arreglos desordenados. */

const int MAX=100;

void Lectura(int, int);                 /* Prototipos de funciones. */
int Busca(int *, int, int);

void main(void)
{
int RES, ELE, TAM, VEC[MAX];
do
{
    printf("Ingrese el tamaño del arreglo: ");
    scanf("%d", &TAM);
}
while (TAM>MAX || TAM<1);   /* Se verifica que el tamaño del arreglo sea
➥correcto. */
Lectura(VEC, TAM);
printf("\nIngrese el elemento a buscar: ");
scanf("%d", &ELE);
RES = Busca(VEC, TAM, ELE);   /* Se llama a la función que busca en el
➥arreglo. */
if (RES)
/* Si RES tiene un valor verdadero —diferente de 0—, se escribe la posición
➥en la que se encontró el elemento. */
    printf("\nEl elemento se encuentra en la posición %d", RES);
else
    printf("\nEl elemento no se encuentra en el arreglo");
}
```

```
void Lectura(int A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
{
   printf("Ingrese el elemento %d: ", I+1);
   scanf("%d", &A[I]);
}
}

int Busca(int A[], int T, int K)
/* Esta función localiza en el arreglo un elemento determinado. Si el elemento
➥es encontrado, regresa la posición correspondiente. En caso contrario, regresa
➥0. */
{
int I = 0, BAN = 0, RES;
while (I < T && !BAN)
  if (A[I] == K)
       BAN++;
  else
     I++;
if (BAN)
    RES = I + 1;
    /* Se asigna I+1 dado que las posiciones en el arreglo comienzan desde
    ➥cero. */
else
   RES = BAN;
return (RES);
}
```

Problema PR5.5

Búsqueda secuencial en arreglos ordenados en forma creciente. La búsqueda secuencial en arreglos ordenados en forma creciente es similar al proceso de búsqueda en arreglos desordenados. La diferencia radica en el uso de una nueva condición para controlar el proceso de búsqueda. El método se aplica revisando elemento por elemento, de izquierda a derecha, hasta encontrar el dato buscado (éxito), hasta que el elemento buscado sea menor que el elemento del arreglo con el cual se está comparando (fracaso), o bien hasta llegar al final de los datos disponibles (fracaso); siempre lo que ocurra primero. Cuando el procedimiento concluye con éxito, se proporciona la posición en la cual fue encontrado el elemento. En caso contrario, se regresa 0 para indicar que el elemento no fue localizado. Cabe destacar que si el arreglo está ordenado en forma decreciente, en el programa sólo hay que modificar la condición de *menor que* por la de *mayor que*.

Datos: VEC[N], ELE

Donde: VEC es un arreglo unidimensional de tipo entero de N elementos, 1 ≤ N ≤ 100, y ELE una variable de tipo entero que representa el elemento a buscar.

Programa 5.10

```
#include <stdio.h>

/* Búsqueda secuencial en arreglos ordenados en forma creciente. */

const int MAX=100;

void Lectura(int, int);           /* Prototipos de funciones. */
int Busca(int *, int, int);

void main(void)
{
int RES, ELE, TAM, VEC[MAX];
do
{
    printf("Ingrese el tamaño del arreglo: ");
    scanf("%d", &TAM);
}
while (TAM > MAX || TAM < 1);
/* Se verifica que el tamaño del arreglo sea correcto. */
Lectura(VEC, TAM);
printf("\nIngrese el elemento a buscar:");
scanf("%d", &ELE);
RES = Busca(VEC, TAM, ELE);     /* Se llama a la función que busca en el
                                ➥arreglo. */
if (RES)
/* Si RES tiene un valor verdadero —diferente de 0—, se escribe la
➥posición en la que se encontró al elemento. */
   printf("\nEl elemento se encuentra en la posición: %d", RES);
else
   printf("\nEl elemento no se encuentra en el arreglo");
}

void Lectura(int A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
{
   printf("Ingrese el elemento %d: ", I+1);
   scanf("%d", &A[I]);
}
}
```

```
int Busca(int A[], int T, int E)
/* Esta función se utiliza para localizar el elemento E en el arreglo
➥unidimensional A.
Si se encuentra, la función regresa la posición correspondiente. En caso
➥contrario regresa 0. */
{
int RES, I = 0, BAN = 0;
while ((I <  T) && (E >= A[I]) && !BAN)
/* Observa que se incorpora una nueva condición. */
   if (A[I] == E)
        BAN++;
   else
        I++;
if (BAN)
    RES =  I + 1;
    /* Se asigna I+1 dado que las posiciones en el arreglo comienzan des
    de cero. */
else
   RES = BAN;
return (RES);
}
```

Problema PR5.6

Búsqueda binaria. La búsqueda binaria consiste en dividir el intervalo de búsqueda en dos partes, comparando el elemento buscado con el central. En caso de ser diferentes se redefinen los extremos del intervalo, según sea el elemento central mayor o menor que el buscado, disminuyendo de esta forma el espacio de búsqueda. El proceso concluye cuando el elemento es encontrado, o bien, cuando el intervalo de búsqueda se anula. Esto implica que el elemento no se encuentra en el arreglo. Cabe destacar que el método funciona únicamente para arreglos ordenados. Con cada iteración del método, el espacio de búsqueda se reduce a la mitad, por lo tanto, el número de comparaciones que se deben realizar disminuye notablemente. Esta disminución resulta más significativa cuanto más grande es el tamaño del arreglo.

Datos: `VEC[N], ELE`

Donde: `VEC` es un arreglo unidimensional de tipo entero de `N` elementos, `1 ≤ N ≤ 100`, y `ELE` una variable de tipo entero que representa el elemento a buscar.

Programa 5.11

```
#include <stdio.h>

/* Búsqueda binaria. */

const int MAX=100;

void Lectura(int, int);                    /* Prototipos de funciones. */
int Binaria(int *, int, int);

void main(void)
{
int RES, ELE, TAM, VEC[MAX];
do
{
     printf("Ingrese el tamaño del arreglo: ");
     scanf("%d", &TAM);
}
while (TAM>MAX || TAM<1); /* Se verifica que el tamaño del arreglo sea
                          ➥correcto. */
Lectura(VEC, TAM);
printf("\nIngrese el elemento a buscar: ");
scanf("%d", &ELE);
RES = Binaria(VEC, TAM, ELE);  /* Se llama a la función que busca en el
                               ➥arreglo. */
if (RES)
/* Si RES tiene un valor verdadero —diferente de 0—, se escribe la
➥posición en la que se encontró el elemento. */
      printf("\nEl elemento se encuentra en la posición: %d", RES);
else
      printf("\nEl elemento no se encuentra en el arreglo");
}

void Lectura(int A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
{
     printf("Ingrese el elemento %d: ", I+1);
     scanf("%d", &A[I]);
}
}

int Binaria(int A[], int T, int E)
/* Esta función se utiliza para realizar una búsqueda binaria del
➥elemento E en el arreglo unidimensional A de T elementos. Si se
➥encuentra el elemento, la función regresa la posición correspondiente.
➥En caso contrario, regresa 0. */
```

```
{
int ELE, IZQ = 0, CEN, DER = T-1, BAN = 0;
while ((IZQ <= DER) && (!BAN))
{
   CEN = (IZQ + DER) / 2;
   if (E == A[CEN])
        BAN = CEN;
   else
        if (E > A[CEN])
             IZQ = CEN + 1;
        else
            DER = CEN - 1;
}
return (BAN);
}
```

Problema PR5.7

Ordenación por inserción directa (forma creciente). Éste es el método que utilizan generalmente los jugadores de cartas cuando las ordenan, de ahí que también se le conozca con el nombre de *método de la baraja*. La idea central del método consiste en insertar un elemento del arreglo en la parte izquierda del mismo que ya se encuentra ordenada. El proceso se repite desde el segundo hasta el enésimo elemento.

Observemos a continuación un ejemplo en la tabla 5.10 para ilustrar el método. Consideremos un arreglo unidimensional A de tipo entero de ocho elementos.

TABLA 5.10. Ordenación por inserción directa

Pasadas	*A[0]*	*A[1]*	*A[2]*	*A[3]*	*A[4]*	*A[5]*	*A[6]*	*A[7]*
1a.	**12**	34	22	11	54	36	19	7
2a.	**12**	**34**	22	11	54	36	19	7
3a.	**12**	**22**	**34**	11	54	36	19	7
4a.	**11**	**12**	**22**	**34**	54	36	19	7
5a.	**11**	**12**	**22**	**34**	**54**	36	19	7
6a.	**11**	**12**	**22**	**34**	**36**	**54**	19	7
7a.	**11**	**12**	**19**	**22**	**34**	**36**	**54**	7
8a.	**7**	**11**	**12**	**19**	**22**	**34**	**36**	**54**

Dato: VEC[N] (arreglo unidimensional de tipo entero de N elementos, 1 ≤ N ≤ 100).

Programa 5.12

```
#include <stdio.h>

/* Ordenación por inserción directa. */

const int MAX = 100;

void Lectura(int *, int);
void Ordena(int *, int);                /* Prototipos de funciones. */
void Imprime(int *, int);

void main(void)
{
int TAM, VEC[MAX];
do
{
    printf("Ingrese el tamaño del arreglo: ");
    scanf("%d", &TAM);
}
while (TAM>MAX || TAM<1);  /* Se verifica que el tamaño del arreglo sea
                           ➥correcto. */
Lectura(VEC, TAM);
Ordena(VEC, TAM);
Imprime(VEC, TAM);
}

void Lectura(int A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
{
  printf("Ingrese el elemento %d: ", I + 1);
  scanf("%d", &A[I]);
}
}


void Imprime(int A[], int T)
/* Esta función se utiliza para escribir un arreglo unidimensional
➥ordenado de T elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
    printf("\nA[%d]: %d", I, A[I]);
}
```

```
void Ordena(int A[], int T)
/* La función Ordena utiliza el método de inserción directa para ordenar
➥los elementos del arreglo unidimensional A. */
{
int AUX, L, I;
for (I=1; I<T; I++)
{
     AUX = A[I];
     L = I - 1;
     while ((L >= 0) && (AUX < A[L]))
     {
       A[L+1] = A[L];
       L--;
     }
     A[L+1] = AUX;
}
}
```

Problema PR5.8

Ordenación por selección directa (forma creciente). Es el mejor de los métodos simples de ordenación (intercambio —burbuja— e inserción). La idea básica del método consiste en buscar el elemento más pequeño del arreglo y colocarlo en la primera posición. Luego se busca el segundo elemento más pequeño y se coloca en la segunda posición; y así sucesivamente hasta que todos los elementos del arreglo queden ordenados.

En la tabla 5.11 se presenta un ejemplo para ilustrar al método. Consideremos un arreglo unidimensional A de tipo entero de ocho elementos.

TABLA 5.11. Ordenación por selección directa

Pasadas	*A[0]*	*A[1]*	*A[2]*	*A[3]*	*A[4]*	*A[5]*	*A[6]*	*A[7]*
1a.	12	34	22	11	54	36	19	7
2a.	**7**	34	22	11	54	36	19	12
3a.	**7**	**11**	22	34	54	36	19	12
4a.	**7**	**11**	**12**	34	54	36	19	22
5a.	**7**	**11**	**12**	**19**	54	36	34	22
6a.	**7**	**11**	**12**	**19**	**22**	36	34	54
7a.	**7**	**11**	**12**	**19**	**22**	**34**	36	54
8a.	**7**	**11**	**12**	**19**	**22**	**34**	**36**	**54**

Dato: VEC[N] (arreglo unidimensional de tipo entero de N elementos, 1 ≤ N ≤ 100).

Programa 5.13

```
#include <stdio.h>

/* Ordenación por selección directa. */

const int MAX = 100;

void Lectura(int *, int);
void Ordena(int *, int);                  /* Prototipos de funciones. */
void Imprime(int *, int);

void main(void)
{
int TAM, VEC[MAX];
do
{
  printf("Ingrese el tamaño del arreglo: ");
  scanf("%d", &TAM);
}
while (TAM>MAX || TAM<1);  /* Se verifica que el tamaño del arreglo sea
                           ➥correcto. */
Lectura(VEC, TAM);
Ordena(VEC, TAM);
Imprime(VEC, TAM);
}


void Lectura(int A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
{
   printf("Ingrese el elemento %d: ", I+1);
   scanf("%d", &A[I]);
}
}

void Imprime(int A[], int T)
/* Esta función se utiliza para escribir un arreglo unidimensional
➥ordenado de T elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
    printf("\nA[%d]: %d", I, A[I]);
}
```

5

```
void Ordena(int A[], int T)
/* La función Ordena utiliza el método de selección directa para ordenar
➥los elementos del arreglo unidimensional A. */
{
int I, J, MEN, L;
for (I=0; I < (T-1); I++)
{
   MEN = A[I];
   L = I;
   for (J=(I+1); J<T; J++)
     if (A[J] < MEN)
     {
          MEN = A[J];
          L = J;
     }
  A[L] = A[I];
  A[I] = MEN;
}
}
```

Problema PR5.9

Construye un programa que, al recibir un arreglo unidimensional de tipo entero que contiene calificaciones de exámenes de alumnos, calcule lo siguiente:

a) 'La media aritmética. Ésta se calcula como la suma de los elementos entre el número de elementos.

b) *La varianza*. Ésta se calcula como la suma de los cuadrados de las desviaciones de la media, entre el número de elementos.

c) *La desviación estándar*. Se calcula como la raíz cuadrada de la varianza.

d) *La moda*. Se calcula obteniendo el número con mayor frecuencia.

Dato: `ALU[N]` (arreglo unidimensional de tipo entero de `N` elementos, `1 ≤ N ≤ 100`).

Programa 5.14

```
#include <stdio.h>
#include <math.h>

/* Estadístico.
El programa, al recibir como dato un arreglo unidimensional de enteros
➥que contiene calificaciones, calcula la media, la varianza, la
➥desviación estándar y la moda. */
```

```
const int MAX = 100;

void Lectura(int *, int);
float Media(int *, int);
float Varianza(int *, int, float);          /* Prototipos de funciones. */
float Desviacion(float);
void Frecuencia(int *, int, int *);
int Moda(int *, int);

void main(void)
{
int TAM, MOD, ALU[MAX], FRE[11] = {0};
float MED, VAR, DES;
do
{
  printf("Ingrese el tamaño del arreglo: ");
  scanf("%d", &TAM);
}
while (TAM > MAX || TAM < 1);
/* Se verifica que el tamaño del arreglo sea correcto. */
Lectura(ALU, TAM);
MED = Media(ALU, TAM);
VAR = Varianza(ALU, TAM, MED);
DES = Desviacion(VAR);
Frecuencia(ALU, TAM, FRE);
MOD = Moda(FRE, 11);
printf("\nMedia:      %.2f", MED);
printf("\nVarianza:   %.2f", VAR);
printf("\nDesviación: %.2f", DES);
printf("\nModa:       %d", MOD);
}

void Lectura(int A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de T
➥elementos de tipo entero. */
{
int I;
for (I=0; I<T; I++)
{
  printf("Ingrese el elemento %d: ", I+1);
  scanf("%d", &A[I]);
}
}

float Media(int A[], int T)
/* Esta función se utiliza para calcular la media. */
```

```
{
int I;
float SUM = 0.0;
for (I=0; I < T; I++)
    SUM += A[I];
return (SUM / T);
}

float Varianza(int A[], int T, float M)
/* Esta función se utiliza para calcular la varianza. */
{
int I;
float SUM = 0.0;
for (I=0; I < T; I++)
    SUM += pow ((A[I] - M), 2);
return (SUM / T);
}

float Desviacion(float V)
/* Esta función se utiliza para calcular la desviación estándar. */
{
return (sqrt(V));
}

void Frecuencia(int A[], int P, int B[])
/* Esta función se utiliza para calcular la frecuencia de calificaciones.
*/
{
int I;
for (I=0; I < P; I++)
    B[A[I]]++;
}

int Moda(int A[], int T)
/* Esta función se utiliza para calcular la moda. */
{
int I, MOD = 0, VAL = A[0];
for (I=1; I<T; I++)
  if (MOD < A[I])
  {
      MOD = I;
      VAL = A[I];
  }
return (MOD);
}
```

Problema PR5.10

Analiza cuidadosamente el siguiente programa y obtén los resultados que se generan al ejecutarlo.

Programa 5.15

```
#include <stdio.h>

/* Apuntadores y arreglos */

void main(void)
{
int X = 5, Y = 8, V[5] = {1, 3, 5, 7, 9};
int *AY, *AX;
AY = &Y;
X = *AY;
*AY = V[3] + V[2];
printf("\nX=%d      Y=%d    V[0]=%d  V[1]=%d      V[2]=%d  V[3]=%d
V[4]=%d", X,
         Y, V[0], V[1], V[2], V[3], V[4]);

AX = &V[V[0]*V[1]];
X = *AX;
Y = *AX * V[1];
*AX = *AY - 3;
printf("\nX=%d      Y=%d    V[0]=%d  V[1]=%d      V[2]=%d  V[3]=%d
V[4]=%d", X,
         Y, V[0], V[1], V[2], V[3], V[4]);
}
```

Compara los resultados que hayas obtenido con los que se presentan a continuación:

```
X=8   Y=12   V[0]=1   V[1]=3   V[2]=5   V[3]=7    V[4]=9
X=7   Y=21   V[0]=1   V[1]=3   V[2]=5   V[3]=18   V[4]=9
```

Problema PR5.11

Analiza cuidadosamente el siguiente programa y obtén los resultados que se generan al ejecutarlo.

Programa 5.16

```
#include <stdio.h>

/* Apuntadores y arreglos */

void main(void)
```

```
{
int V1[4]= {2, 3, 4, 7}, V2[4]= {6};
int *AX, *AY;
AX = &V1[3];
AY = &V2[2];
V1[V2[0]-V1[2]]= *AY;
*AY= *AX - V1[0];
printf("\nV1[0]=%d V1[1]=%d V1[2]=%d V1[3]=%d \tV2[0]=%d V2[1]=%d V2[2]=%d
          V2[3]=%d", V1[0],V1[1],V1[2],V1[3],V2[0],V2[1],V2[2],V2[3]);
V2[1] = ++*AX;
V2[3] = (*AY)++;
*AX += 2;
printf("\nV1[0]=%d V1[1]=%d V1[2]=%d V1[3]=%d \tV2[0]=%d V2[1]=%d V2[2]=%d
          V2[3]=%d", V1[0],V1[1],V1[2],V1[3],V2[0],V2[1],V2[2],V2[3]);
}
```

Compara los resultados que hayas obtenido con los que se presentan a continuación:

```
V1[0]=2  V1[1]=3  V1[2]=0  V1[3]=7   V2[0]=6  V2[1]=0  V2[2]=5  V2[3]=0

V1[0]=2  V1[1]=3  V1[2]=0  V1[3]=10  V2[0]=6  V2[1]=8  V2[2]=6  V2[3]=5
```

Problema PR5.12

Analiza cuidadosamente el siguiente programa y obtén los resultados que se generan al ejecutarlo.

Programa 5.17

```
#include <stdio.h>

/* Apuntadores y arreglos */

void main(void)
{
int V1[4] = {1, 3, 5, 7}, V2[4]= {2,4};
int *AX, *AY;
AX = &V1[2];
AY = &V2[2];
V2[2] = *(AX+1);
V2[3] = *AX;
AX = AX + 1;
V1[0] =  *AX;
printf("\nV1[0]=%d V1[1]=%d V1[2]=%d V1[3]=%d \tV2[0]=%d V2[1]=%d
```

```
V1[0],V1[1],V1[2],V1[3],V2[0],V2[1],V2[2],V2[3]);
V1[2] = *AY;
V1[1] = --*AY;
AX = AX + 1;
V1[3] =  *AX;
printf("\nV1[0]=%d V1[1]=%d V1[2]=%d V1[3]=%d \tV2[0]=%d V2[1]=%d
    V2[2]=%d  V2[3]=%d", V1[0],V1[1],V1[2],V1[3],V2[0],V2[1],V2[2],V2[3]);
}
```

Compara los resultados que hayas obtenido con los que se presentan a continuación:

```
V1[0]=7  V1[1]=3  V1[2]=5  V1[3]=7       V2[0]=2  V2[1]=4  V2[2]=7  V2[3]=5

V1[0]=7  V1[1]=6  V1[2]=7  V1[3]=basura  V2[0]=2  V2[1]=4  V2[2]=6  V2[3]=5
```

Problemas suplementarios

Nota: Todos los problemas deben ser resueltos con la ayuda de funciones.

Problema PS5.1

Escribe un programa que, al dar como dato un arreglo unidimensional de números enteros, determine cuántos de ellos son positivos, cuántos negativos y cuántos nulos.

Dato: `VEC[N]` (arreglo unidimensional de tipo entero de `N` elementos, $1 \leq N \leq 100$).

Problema PS5.2

Escribe un programa que reciba como entrada un arreglo unidimensional *ordenado* de N enteros y obtenga como salida ese mismo arreglo pero sin los elementos repetidos.

Dato: `VEC[N]` (arreglo unidimensional de tipo entero de `N` elementos, $1 \leq N \leq 100$).

Problema PS5.3

Escribe un programa que almacene en un arreglo unidimensional los primeros 100 números de Fibonacci e imprima el arreglo correspondiente.

Problema PS5.4

Escribe un programa que inserte y elimine elementos en un arreglo unidimensional de tipo entero que se encuentre desordenado. Considera que no se pueden insertar elementos repetidos.

Datos: `VEC[N]`, `ELE`

Donde: `VEC` es un arreglo unidimensional de tipo entero de `N` elementos, $1 \leq N \leq 100$, y `ELE` una variable de tipo entero que expresa el número que se va a insertar o eliminar.

Problema PS5.5

Escribe un programa que inserte y elimine elementos en un arreglo unidimensional de tipo entero que se encuentre ordenado en forma creciente. Toma en cuenta que no se pueden insertar elementos repetidos.

Datos: `VEC[N]`, `ELE`

Donde: `VEC` es un arreglo unidimensional de tipo entero de `N` elementos, $1 \leq N \leq 100$, y `ELE` una variable de tipo entero que representa el número que se va a insertar o eliminar.

Problema PS5.6

En un arreglo unidimensional de tipo real se almacenan las calificaciones de un grupo de `N` alumnos que presentaron un examen de admisión a una universidad. Escribe un programa que calcule e imprima lo siguiente:

a) El promedio general del grupo.
b) El porcentaje de alumnos aprobados (todos aquellos alumnos cuyo puntaje supere los 1300 puntos).
c) El número de alumnos cuya calificación sea mayor o igual a 1500.

Dato: `ALU[N]` (arreglo unidimensional de tipo real de `N` elementos, $1 \leq N \leq 100$).

Problema PS5.7

En un arreglo unidimensional de tipo real se tienen almacenadas las toneladas mensuales de cereales cosechadas durante el año anterior en una estancia de la pampa Argentina. Escribe un programa que calcule e imprima lo siguiente:

a) El promedio anual de toneladas cosechadas.

b) ¿Cuántos meses tuvieron una cosecha superior al promedio anual?

c) ¿En qué mes se produjo el mayor número de toneladas? ¿Cuántas fueron?

Dato: `COS[12]` (arreglo unidimensional de tipo real de 12 elementos).

Problema PS5.8

Construye un programa en **C** que, al recibir como datos dos arreglos unidimensionales de tipo entero, *desordenados*, de `N` y `M` elementos respectivamente, genere un nuevo arreglo unidimensional ordenado en forma descendente de `N+M` elementos de tipo entero, mezclando los dos primeros arreglos.

Datos: `VEC1[N]`, `VEC2[M]`

Donde: `VEC1` y `VEC2` son dos arreglos unidimensionales de tipo entero de `N` y `M` elementos, respectivamente, $1 \leq N \leq 100$, $1 \leq M \leq 100$.

Problema PS5.9

Construye un programa en **C** que, al recibir como datos dos arreglos unidimensionales de tipo entero, *ordenados en forma ascendente*, de `N` y `M` elementos respectivamente, genere un nuevo arreglo unidimensional ordenado en forma ascendente de `N+M` elementos de tipo entero, mezclando los dos primeros arreglos.

Datos: `VEC1[N]`, `VEC2[M]`

Donde: `VEC1` y `VEC2` son dos arreglos unidimensionales de tipo entero de `N` y `M` elementos, respectivamente, $1 \leq N \leq 100$, $1 \leq M \leq 100$.

Problema PS5.10

Construye un programa en **C** que, al recibir como datos dos arreglos unidimensionales de tipo entero, el primero ordenado en forma ascendente y el segundo

en forma descendente, de N y M elementos respectivamente, genere un nuevo arreglo unidimensional ordenado en forma ascendente de N+M elementos de tipo entero, mezclando los dos primeros arreglos.

Datos: VEC1[N], VEC2[M]

Donde: VEC1 y VEC2 son dos arreglos unidimensionales de tipo entero de N y M elementos, respectivamente, $1 \leq N \leq 100$, $1 \leq M \leq 100$.

Problema PS5.11

Escribe un programa en el lenguaje **C** que almacene en un arreglo unidimensional los primeros 30 *números perfectos*. Un número se considera perfecto si la suma de los divisores, excepto él mismo, es igual al propio número. El **6**, por ejemplo, es un número perfecto.

Problema PS5.12

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional de tipo entero de N elementos, determine si el arreglo es *palíndrome*. Por ejemplo, el arreglo VEC que se muestra a continuación es palíndrome:

2	4	5	4	2

Dato: VEC[N] (arreglo unidimensional de tipo entero de N elementos, $1 \leq N \leq 100$).

CAPÍTULO 6

Arreglos multidimensionales

6.1. Introducción

Los arreglos que estudiamos en el capítulo anterior reciben el nombre de unidimensionales porque para acceder a un elemento del arreglo sólo tenemos que utilizar un índice. Existen también arreglos con múltiples dimensiones, a cuyos elementos se debe acceder utilizando múltiples índices. El número de dimensiones del arreglo depende tanto de las características del problema que se va a resolver, como de las facilidades del lenguaje de programación que se utilice para implementar la solución. Sin duda, los **arreglos bidimensionales** —conocidos también como matrices— son los arreglos multidimensionales más utilizados y los que trataremos principalmente en este capítulo.

6.2. Arreglos bidimensionales

Formalmente definimos un arreglo bidimensional de la siguiente manera:

> "Un arreglo bidimensional es una colección finita, homogénea y ordenada de datos, en la que se hace referencia a cada elemento del arreglo por medio de dos índices. El primero de los índices se utiliza para indicar la fila, y el segundo, para indicar la columna."

Un arreglo bidimensional permite almacenar `N x M` elementos del mismo tipo (enteros, reales, caracteres, cadenas de caracteres, etc.) y acceder a cada uno de ellos. Al igual que en los arreglos unidimensionales, se distinguen dos partes importantes: los **componentes** y los **índices**. Los primeros hacen referencia a los elementos que se almacenan en cada una de sus casillas, y los segundos, por otra parte, especifican la forma de acceder a cada uno de los elementos. Para hacer referencia a un componente de un arreglo bidimensional debemos utilizar tanto el *nombre del arreglo,* como los *índices* del elemento (fila y columna).

En la figura 6.1 se puede observar la representación gráfica de un arreglo bidimensional.

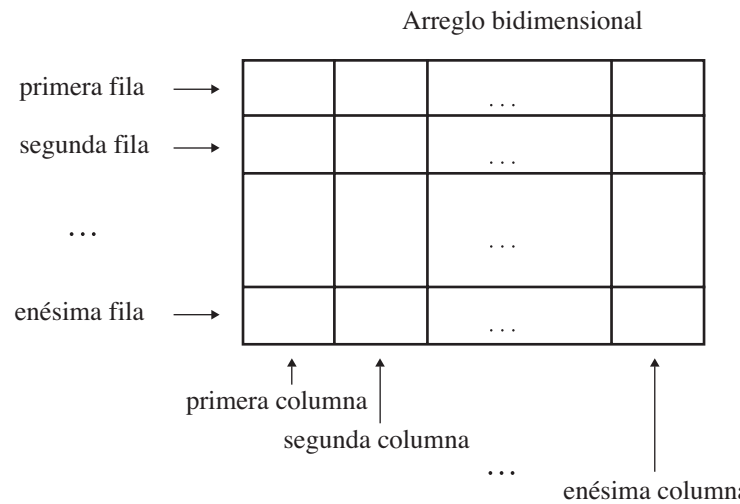


FIGURA 6.1
Representación gráfica de un arreglo bidimensional.

En la figura 6.2 se muestra el arreglo bidimensional A que contiene 18 elementos de tipo entero. Observa que el arreglo tiene tres filas y seis columnas. A cada elemento del arreglo se accede por medio de dos índices. El primero se utiliza para las filas y el segundo para las columnas. Si queremos acceder al elemento de la primera fila y la primera columna, debemos escribir `A[0][0]`; si en cambio queremos acceder al quinto elemento de la primera fila, debemos escribir `A[0][4]`. El valor de `A[1][5]` es `4`, el valor de `A[2][1 + 2]` es 0, el resultado de `A[2][2] + A[1][5]` es `9`, y el resultado de `A[0][4] * A[2][1]` es `-6`.

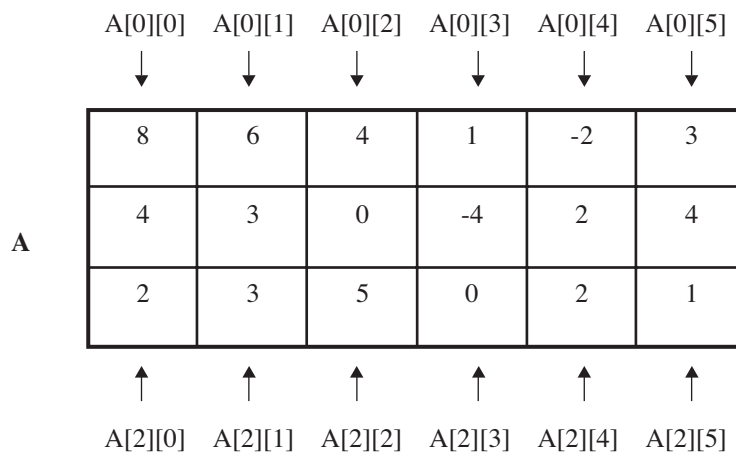


FIGURA 6.2
Índices y componentes de un arreglo bidimensional.

6.3. Declaración de arreglos bidimensionales

El espacio que los arreglos ocupan en memoria se reserva en el momento de realizar la declaración de los mismos. A continuación presentamos diferentes formas de declarar arreglos, con su explicación correspondiente.

```
void main(void)
{
 . . .
int A[5][10];    /* Declaración de un arreglo bidimensional de tipo entero de 5
                    filas y 10 columnas. */
float B[5][5];   /* Declaración de un arreglo bidimensional de tipo real de 5
                    filas y 5 columnas. */
. . .
}
```

Una vez que se definen los arreglos, sus elementos pueden recibir los valores a través de múltiples asignaciones, o bien, como ocurre frecuentemente en la práctica, por medio de un ciclo y la lectura correspondiente de los valores. Analicemos el siguiente ejemplo.

EJEMPLO 6.1

Escribe un programa en **C** que, al recibir como dato un arreglo bidimensional cuadrado de tipo entero de dimensión 10, imprima la diagonal de dicha matriz.

Dato: `MAT[10][10]` (arreglo bidimensional de tipo entero que almacena 100 elementos).

Programa 6.1

```
#include <stdio.h>

/* Diagonal principal.
El programa, al recibir como dato una matriz de tipo entero, escribe la
diagonal principal. */

const int TAM = 10;

void Lectura(int [][TAM], int);     /* Prototipo de funciones. */
void Imprime(int [][TAM], int);
/* Observa que siempre es necesario declarar el número de columnas. Si no lo
➥haces, el compilador marcará un error de sintaxis. */

void main(void)
{
int MAT[TAM][TAM];
Lectura(MAT, TAM);
Imprime(MAT, TAM);
}

void Lectura(int A[][TAM], int F)
/* La función Lectura se utiliza para leer un arreglo bidimensional. Observa
➥que sólo se debe pasar como parámetro el número de filas ya que la matriz
➥es cuadrada. */
{
int I, J;
for (I=0; I<F; I++)
    for (J=0; J<F; J++)
    {
         printf("Ingrese el elemento %d %d: ", I+1, J+1);
         scanf("%d", &A[I][J]);
    }
}
```

```
void Imprime(int A[][TAM], int F)
/* La función Imprime se utiliza para escribir un arreglo bidimensional
➥cuadrado de F filas y columnas. */
{
int I, J;
for (I=0; I<F; I++)
   for (J=0; J<TAM; J++)
      if (I == J)
          printf("\nDiagonal %d %d: %d ", I, J, A[I][J]);
}
```

Otra forma de asignar valores a los componentes del arreglo es al realizar la definición del mismo. A continuación presentamos diferentes casos con su explicación correspondiente y una figura en la que se pueden observar los valores que toman los diferentes componentes del arreglo bidimensional.

```
. . .
int A[3][6] = {0};    /* Todos los componentes del arreglo se inicializan con 0. */
```

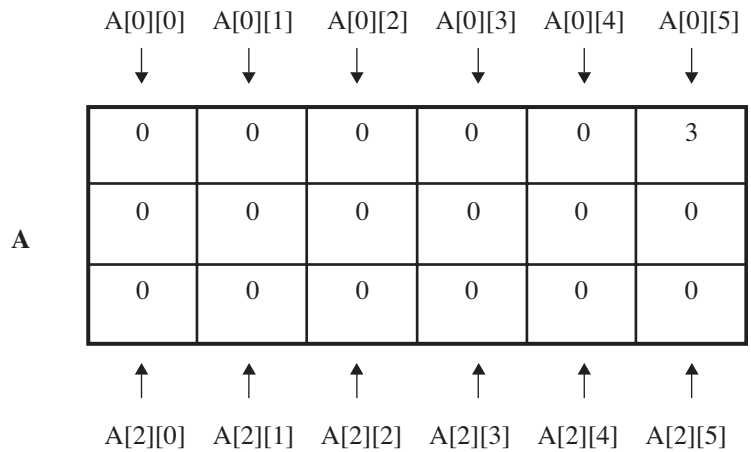


FIGURA 6.3
Declaración del arreglo bidimensional A.

```
int B[3][6] = {3,4,6,8};    /* Los primeros cuatro componentes de la primera fila
➥se inicializan con los valores: 3,4,6 y 8. El resto con 0. */
```

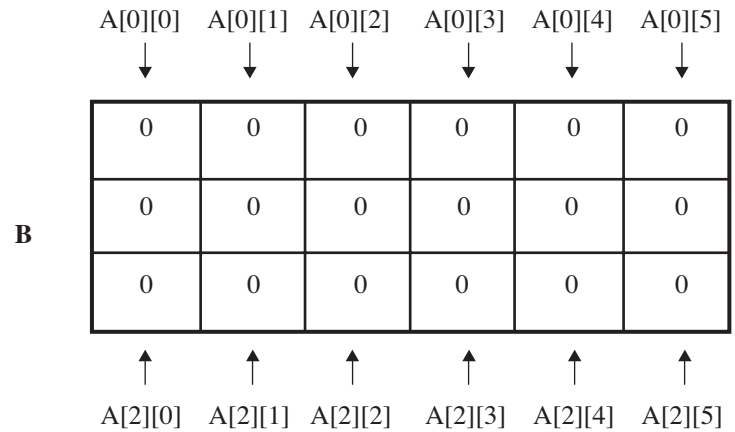


FIGURA 6.4
Declaración del arreglo bidimensional B.

```
int C[3][6] = {6, 23, 8, 4, 11, 33, 21, 0, 6, 10, 23, 4, 8, 2, 1, 6, 9, 15};
/* Cada componente del arreglo recibe un valor. La asignación se realiza fila a
➥fila.*/

int C[3][6] = {{6, 23, 8, 4, 11, 33}, {21, 0, 6, 10, 23, 4}, {8, 2, 1, 6, 9, 15}};
/* La asignación anterior también se puede realizar de esta forma. */
```

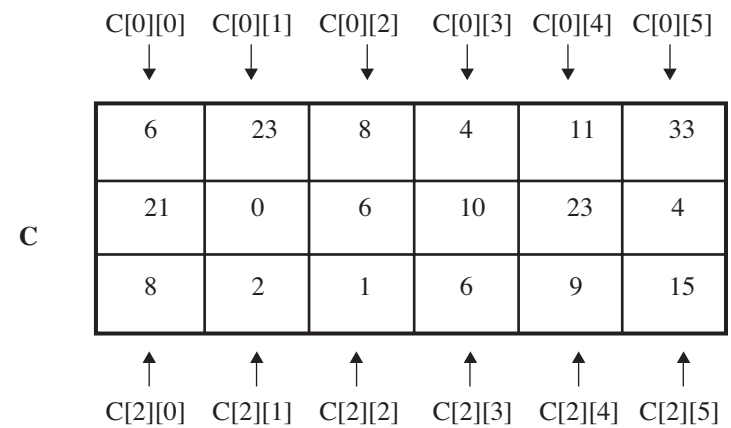


FIGURA 6.5
Declaración del arreglo bidimensional C.

```
int C[3][6] = {{6, 23, 8, 4, 11, 35, 8}};      /* Esta declaración genera un error
➥de sintaxis, ya que la fila tiene espacio para seis elementos y se asignan
➥siete. /*
```

Observemos a continuación otro ejemplo.

EJEMPLO 6.2

Escribe un programa en **C** que, al recibir dos arreglos bidimensionales `MA` y `MB` de `MxN` elementos cada uno, calcule la suma de ambos arreglos, almacene el resultado en otro arreglo bidimensional e imprima, además, el resultado obtenido.

Datos: `MA[M][N]`, `MB[M][N]` (arreglos bidimensionales de tipo entero de `MxN` elementos, `1 ≤ M ≤ 50` y `1 ≤ N ≤ 50`).

Programa 6.2

```
clude <stdio.h>

/* Suma matrices.
El programa, al recibir como datos dos arreglos bidimensionales del mismo
➥tamaño, calcula la suma de ambos y la almacena en un tercer arreglo
➥bidimensional. */

const int MAX = 50;

/* Prototipo de funciones. */
void Lectura(int [][MAX], int, int);
void Suma(int [][MAX], int [][MAX], int [][MAX], int, int);
void Imprime(int [][MAX], int, int);

void main(void)
{
int MA[MAX][MAX], MB[MAX][MAX], MC[MAX][MAX];
/* Declaración de los tres arreglos */
int FIL, COL;
do
{
    printf("Ingrese el número de filas de los arreglos: ");
    scanf("%d", &FIL);
}
while (FIL> MAX || FIL< 1);
/* Se verifica que el número de filas sea correcto. */
do
{
    printf("Ingrese el número de columnas de los arreglos: ");
    scanf("%d", &COL);
}
while (COL > MAX || COL < 1);
/* Se verifica que el número de columnas sea correcto. */
printf("\nLectura del Arreglo MA\n");
Lectura(MA, FIL, COL);
```

```
printf("\nLectura del Arreglo MB\n");
Lectura(MB, FIL, COL);
Suma (MA, MB, MC, FIL, COL);
printf("\nImpresión del Arreglo MC\n");
Imprime (MC, FIL, COL);
}

void Lectura(int A[][MAX], int F, int C)
/* La función Lectura se utiliza para leer un arreglo bidimensional entero de F
➥filas y C columnas. */
{
int I, J;
for (I=0; I<F; I++)
    for (J=0; J<C; J++)
    {
         printf("Ingrese el elemento %d %d: ", I+1, J+1);
         scanf("%d", &A[I][J]);
    }
}

void Suma(int M1[][MAX],int M2[][MAX],int M3[][MAX], int F, int C)
/* La función Suma se utiliza para sumar los arreglos y almacenar el resultado
➥en un tercer arreglo bidimensional. */
{
int I, J;
for (I=0; I<F; I++)
    for (J=0; J<C; J++)
         M3[I][J]= M1[I][J] + M2[I][J];
}

void Imprime(int A[][MAX], int F, int C)
/* La función Imprime se utiliza para escribir un arreglo bidimensional de tipo
➥entero de F filas y C columnas. */
{
int I, J;
for (I=0; I<F; I++)
    for (J=0; J<C; J++)
        printf("\nElemento %d %d: %d ", I, J, A[I][J]);
}
```

6.4. Arreglos de más de dos dimensiones

Los arreglos de más de dos dimensiones son similares a los bidimensionales, excepto porque cada elemento se debe referenciar por medio de tres o más índices. Los arreglos de tres dimensiones se conocen como tridimensionales. Cabe destacar que los arreglos de más de tres dimensiones se utilizan muy poco en la práctica.

En la figura 6.6 se muestra el arreglo tridimensional **A** que contiene 18 elementos de tipo entero. Observa que el arreglo tiene tres filas, dos columnas y tres planos de pro-

fundidad. A cada elemento del arreglo se accede por medio de tres índices. El primer índice se utiliza para las filas, el segundo para las columnas y el tercero para la profundidad. Si queremos acceder al elemento de la primera fila, la primera columna y el primer plano de profundidad debemos escribir `A[0][0][0]`; si en cambio queremos acceder al elemento de la tercera fila, la segunda columna y la segunda profundidad escribimos `A[2][1][1]`. El valor de `A[1][1][2]` es 2, el de `A[2][0][2]` es 3, el resultado de `A[1][1][1] + A[0][1][2]` es 4, y el de `A[1][1][1] * A[2][1][0]` es 12.

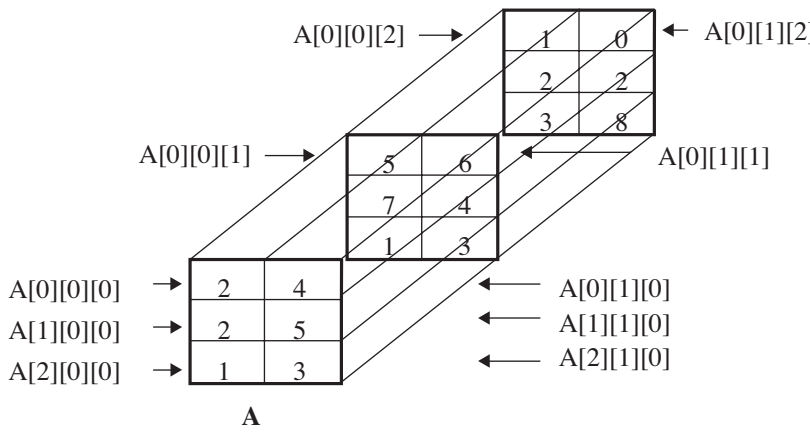


FIGURA 6.6
Índices y componentes de un arreglo tridimensional.

6.5. Declaración de arreglos tridimensionales

La declaración de los arreglos tridimensionales, y en general de aquellos de más de dos dimensiones, es similar a la de los arreglos bidimensionales. El espacio que los arreglos ocupan en memoria se reserva en el momento de realizar la declaración de los mismos. Observemos a continuación las siguientes declaraciones.

```
    void main(void)
    {
     . . .
    int A[5][10][3];      /* Declaración de un arreglo tridimensional de tipo
➥entero de 5 filas, 10 columnas y 3 planos de profundidad. */
    float B[5][5][6];     /* Declaración de un arreglo tridimensional de tipo
➥real de 5 filas, 5 columnas y 6 planos de profundidad. */

    . . .
    }
```

Una vez que se definen los arreglos, sus elementos pueden recibir los valores a través de múltiples asignaciones, o bien, como mencionamos anteriormente, a través de un ciclo y la correspondiente lectura de valores. Veamos el siguiente ejemplo.

EJEMPLO 6.3

En una universidad se almacena información sobre el número de alumnos que han ingresado a sus ocho diferentes carreras en los dos semestres lectivos, en los últimos cinco años. Escribe un programa en **C** que calcule lo siguiente:

a) El año en que ingresó el mayor número de alumnos a la universidad.

b) La carrera que recibió el mayor número de alumnos el último año.

c) ¿En qué año la carrera de Ingeniería en Computación recibió el mayor número de alumnos?

Dato: `UNI[8][2][5]` (arreglo tridimensional de tipo entero que almacena información sobre el ingreso de alumnos a una universidad).

Observa además que las carreras de la universidad tienen un valor numérico asociado:

1. Contabilidad.
2. Administración.
3. Economía.
4. Relaciones Internacionales.
5. Matemáticas.
6. Ingeniería en Computación.
7. Ingeniería Industrial.
8. Ingeniería en Telemática.

Programa 6.3

```
#include <stdio.h>

/* Universidad.
El programa, al recibir información sobre el número de alumnos que han ingresado
➥a sus ocho diferentes carreras en los dos semestres lectivos de los últimos
➥cinco años, obtiene información útil para el departamento de escolar. */

const int F = 8, C = 2, P = 5;
/* Se declaran constantes para la fila, la columna y la profundidad. */
void Lectura(int [][C][P], int, int, int);
```

```
void Funcion1(int [][C][P], int, int, int);
void Funcion2(int [][C][P], int, int, int);
void Funcion3(int [][C][P], int, int, int);
/* Prototipo de funciones. Cada prototipo de función corresponde a uno de los
➥incisos. */

void main(void)
{
int UNI[F][C][P];
Lectura(UNI, F, C, P);
Funcion1(UNI, F, C, P);
Funcion2(UNI, F, C, P);
Fu     Funcion3(UNI, 6, C, P);
/* Se coloca el 6 como parámetro ya que es la clave de la Ingeniería en
➥Computación. */
}

void Lectura(int A[][C][P], int FI, int CO, int PR)
/* La función Lectura se utiliza para leer un arreglo tridimensional de tipo
➥entero de FI filas, CO columnas y PR profundidad. Observa que al ser
➥tridimensional se necesitan tres ciclos para recorrer el arreglo. */
{
int K, I, J;
for (K=0; K<PR; K++)
    for (I=0; I<FI; I++)
        for (J=0; J<CO; J++)
        {
            printf("Año: %d\tCarrera: %d\tSemestre: %d  ", K+1, I+1, J+1);
            scanf("%d", &A[I][J][K]);
        }
}

void Funcion1(int A[][C][P],int FI, int CO, int PR)
/* Esta función se utiliza para determinar el año en el que ingresó el mayor
➥número de alumnos a la universidad. Observa que el primer ciclo externo
➥corresponde a los años. */
{
int K, I, J, MAY = 0, AO = -1, SUM;
for (K=0; K<PR; K++)
/* Por cada año se suma el ingreso de los dos semestres de las ocho carreras.
*/
{
    SUM = 0;
    for (I=0; I<FI; I++)
         for (J=0; J<CO; J++)
              SUM += A[I][J][K];
    if (SUM > MAY)
    {
         MAY = SUM;
         AO = K;
    }
```

```
}
printf("\n\nAño con mayor ingreso de alumnos: %d    Alumnos: %d", AO+1, MAY);
}

void Funcion2(int A[][C][P],int FI, int CO, int PR)
/* Esta función se utiliza para determinar la carrera que recibió el mayor
➥número de alumnos el último año. Observa que no se utiliza un ciclo para los
➥planos de la profundidad, ya que es un dato (PR). */
{
int I, J, MAY = 0, CAR = -1, SUM;
for (I=0; I<FI; I++)
{
     SUM = 0;
     for (J=0; J<CO; J++)
         SUM += A[I][J][PR-1];
     if (SUM > MAY)
     {
          MAY = SUM;
          CAR = I;
     }
}
printf("\n\nCarrera con mayor número de alumnos: %d  Alumnos: %d", CAR+1,
     MAY);
}

void Funcion3(int A[][C][P],int FI, int CO, int PR)
/* Esta función se utiliza para determinar el año en el que la carrera
➥Ingeniería en Computación recibió el mayor número de alumnos. Observa que no
➥se utiliza un ciclo para trabajar con las filas, ya que es un dato (FI). */
{
int K, J, MAY = 0, AO = -1, SUM;
for (K=0; K<PR; K++)
{
     SUM = 0;
     for (J=0; J<CO; J++)
          SUM += A[FI-1][J][K];
     if (SUM > MAY)
     {
          MAY = SUM;
          AO = K;
     }
}
printf("\n\nAño con mayor ingreso de alumnos: %d   Alumnos: %d", AO+1, MAY);
}
```

A continuación presentamos una serie de problemas resueltos y después de éstos veremos problemas suplementarios cuidadosamente seleccionados.

Problemas resueltos

Problema PR6.1

Escribe un programa en **C** que, al recibir como dato un arreglo bidimensional (matriz) cuadrado, determine si el mismo es **simétrico**. Recuerda que se considera simétrico si cumple la siguiente condición: `A[I][J] = A[J][I]`.

Dato: `MAT[N][N]` (donde `MAT` es un arreglo bidimensional de tipo entero de `NxN` elementos, `1 ≤ N ≤ 100`).

Programa 6.4

```
#include <stdio.h>

/* Simétrico.
El programa, al recibir como dato un arreglo bidimensional cuadrado, determina
si el mismo es simétrico. */

const int MAX = 100;

void Lectura(int [][MAX], int);      /* Prototipos de funciones. */
int Simetrico(int [][MAX], int);

void main(void)
{
int MAT[MAX][MAX], N, RES;
do
{
     printf("Ingrese el tamaño del arreglo: ");
     scanf("%d", &N);
}
while (N > MAX || N < 1);     /* Se verifica que el tamaño del arreglo sea
                              válido. */
Lectura (MAT, N);
RES = Simetrico(MAT, N);
if (RES)
     printf("\nEl arreglo bidimensional es simétrico");
else
     printf("\nEl arreglo bidimensional no es simétrico");
}

void Lectura(int A[][MAX], int T)
/* La función Lectura se utiliza para leer un arreglo bidimensional cuadrado de
tipo entero de T filas y T columnas. */
{
```

```
int I, J;
for (I=0; I<T; I++)
    for (J=0; J<T; J++)
    {
         printf("Fila: %d\tColumna: %d", I+1, J+1);
         scanf("%d", &A[I][J]);
    }
}

int Simetrico (int A[][MAX], int T)
/* La función Simétrico se utiliza para determinar si el arreglo bidimensional
➥cuadrado es simétrico. Si es simétrico regresa 1, en caso contrario, 0.
➥Observa que en el segundo ciclo solamente se recorre la matriz triangular
➥inferior, sin la diagonal principal. */
{
int I = 0, J, F = 1;
while ((I < T) && F)
{
    J = 0;
    while ((J < I) && F)
       if (A[I][J] == A[J][I])
            J++;
       else
          F = 0;
    I++;
 }
 return (F);
 }
```

Problema PR6.2

A la clase de Estructuras de Datos del profesor Serrano asiste un grupo numeroso de alumnos. El profesor Serrano es muy exigente y aplica cuatro exámenes durante el semestre. Escribe un programa en **C** que resuelva lo siguiente:

a) El promedio de calificaciones de cada alumno.

b) El promedio del grupo en cada examen.

c) El examen que tuvo el mayor promedio de calificación.

Dato: `ALU [N, 4]` (donde `ALU` es un arreglo bidimensional de tipo real de `N` filas y cuatro columnas que almacena calificaciones de alumnos, $1 \leq N \leq 50$).

Programa 6.5

```
#include <stdio.h>

/* Alumnos.
El programa, al recibir un arreglo bidimensional que contiene información
➥sobre calificaciones de alumnos en cuatro materias diferentes, obtiene
➥resultados estadísticos. */

const int MAX = 50;
const int EXA = 4;

void Lectura(float [MAX][EXA], int);
void Funcion1(float [MAX][EXA], int);          /* Prototipos de funciones. */
void Funcion2(float [MAX][EXA], int);

void main(void)
{
int NAL;
float ALU[MAX][EXA];
do
{
      printf("Ingrese el número de alumnos del grupo: ");
      scanf("%d", &NAL);
}     /* Se verifica que el número de alumnos del grupo sea válido. */
while (NAL > MAX ¦¦ NAL < 1);
Lectura(ALU, NAL);
Funcion1(ALU, NAL);
Funcion2(ALU, NAL);
}

void Lectura(float A[][EXA], int N)
/* La función Lectura se utiliza para leer un arreglo bidimensional de tipo
real de N filas y EXA columnas. */
{
int I, J;
for (I=0; I<N; I++)
    for (J=0; J<EXA; J++)
    {
         printf("Ingrese la calificación %d del alumno %d: ", J+1, I+1);
         scanf("%f", &A[I][J]);
    }
}

void Funcion1(float A[][EXA], int T)
/* Esta función se utiliza para obtener el promedio de cada estudiante. */
{
int I, J;
float SUM, PRO;
for (I=0; I<T; I++)
{
```

```
     SUM = 0;
     for (J=0; J<EXA; J++)
          SUM += A[I][J];
     PRO = SUM / EXA;
     printf("\nEl promedio del alumno %d es: %5.2f", I+1, PRO);
}
}

void Funcion2(float A[][EXA], int T)
/* Esta función se utiliza tanto para obtener el promedio de cada examen, así
como también el examen que obtuvo el promedio más alto. */
{
int I, J, MAY;
float SUM, PRO, MPRO = 0;
printf("\n");
for (J=0; J<EXA; J++)
{
     SUM = 0;
     for (I=0; I<T; I++)
          SUM += A[I][J];
     PRO = SUM / T;
     if (PRO > MPRO)
     {
          MPRO = PRO;
          MAY = J;
     }
     printf("\nEl promedio del examen %d es: %f", J+1, PRO);
}
printf("\n\nEl examen con mayor promedio es: %d \t Promedio: %5.2f", MAY+1,
➥MPRO);
}
```

Problema PR6.3

Escribe un programa en **C** que intercambie las `N` columnas de un arreglo bidimensional. Los elementos de la primera columna se intercambian con los de la última, los de la segunda con los de la penúltima, y así sucesivamente.

Dato: `MAT [M][N]` (arreglo bidimensional de tipo real de `M` filas y `N` columnas, $1 \leq M \leq 50$ y $1 \leq N \leq 50$).

Programa 6.6

```
#include <stdio.h>

/* Intercambia.
El programa intercambia las columnas de un arreglo bidimensional. Los
➥elementos de la primera columna se intercambian con los de la última,
➥los de la segunda con los de la penúltima, y así sucesivamente. */

const int MAX = 50;

void Lectura(float [][MAX], int, int);
void Intercambia(float [][MAX], int, int);       /* Prototipos de funciones. */
void Imprime(float [][MAX], int, int);

void main(void)
{
int F, C;
float MAT[MAX][MAX];
do
{
      printf("Ingrese el número de filas: ");
      scanf("%d", & F);
}
while (F > MAX || F < 1);  /* Se verifica que el número de filas sea correcto. */
do
{
      printf("Ingrese el número de columnas: ");
      scanf("%d", &C);
}
while (C > MAX || C < 1);  /* Se verifica que el número de columnas sea
                           ➥correcto. */
Lectura(MAT, F, C);
Intercambia(MAT, F, C);
Imprime(MAT, F, C);
}

void Lectura(float A[][MAX], int F, int C)
/* La función Lectura se utiliza para leer un arreglo bidimensional de tipo
➥real de F filas y C columnas. */
{
int I, J;
for (I=0; I< F; I++)
    for (J=0; J<C; J++)
    {
         printf("Ingrese el elemento %d %d:  ", I+1, J+1);
         scanf("%f", &A[I][J]);
    }
}
void Intercambia(float A[][MAX], int F, int C)
/* Esta función se utiliza para intercambiar las columnas del arreglo
```

```
bidimensional. Observa que el índice correspondiente a las columnas sólo se
mueve hasta la mitad del arreglo. */
{
int I, J;
float AUX;
/* Observa que en esta función el índice I se utiliza para las columnas, y el
índice J para las filas. */
for (I=0; I < (C / 2); I++)
       for (J=0; J< F; J++)
       {
             AUX = A[J][I];
             A[J][I] = A[J][C-I-1];
             A[J][C-I-1]=AUX;
       }
}

void Imprime(float A[][MAX], int F, int C)
/* La función Imprime se utiliza para escribir un arreglo bidimensional de tipo
➥real de F filas y C columnas. */
{
int I, J;
for (I=0; I< F; I++)
      for (J=0; J<C; J++)
           printf("\nElemento %d %d: %5.2f", I+1, J+1, A[I][J]);
}
```

Problema PR6.4

En la fábrica de lácteos TREGAR, localizada en Gobernador Crespo, Santa Fe, Argentina, se proporcionan las ventas mensuales de sus 15 principales productos de la siguiente manera:

```
1,  5, 150
1,  5,  50
1,  8, 100
1,  8,  30
2,  5,  50
   . . .
-1, -1,  -1
```

Observa que el primer valor de la transacción corresponde al mes, el segundo al producto y el tercero a la cantidad vendida del mismo. La primera transacción indica que en el mes **1** se vendieron **150** unidades del producto **5**. Observa que en un mismo mes se pueden tener diferentes ventas de un mismo producto (transacción 2).

Se conocen además los precios de venta de cada uno de los productos. Escribe un programa en **C** que calcule lo siguiente:

a) El monto de venta anual de cada uno de los productos.

b) El monto total de ventas de la fábrica de lácteos.

c) El tipo de producto que más se ha vendido y el monto de las ventas.

Datos: MES_1, PRO_1, CAN_1
MES_2, PRO_2, CAN_2
. . .
-1, -1, -1
COS_1, COS_2, ..., COS_{15}

Donde: MES_i es una variable de tipo entero que hace referencia al mes de producción de la transacción i, $1 \leq MES \leq 12$.
PRO_i es una variable de tipo entero que indica el tipo de producto, $1 \leq PRO \leq 15$.
CAN_i es una variable de tipo entero que representa las ventas del producto PRO en la transacción i.
COS_i es una variable de tipo real que representa el costo de producto i, $1 \leq i \leq 15$.

Programa 6.7

```
#include <stdio.h>

/* Fábrica de lácteos.
El programa, al recibir como datos las ventas mensuales de diferentes
➥productos, obtiene información estadística valiosa para la empresa. */

void Lectura1(int [15][12]);
void Lectura2(float, int);
void Funcion1(int [][12], int, int, float *, float *);  /* Prototipos de
➥funciones. */
void Funcion2(float *, int);
void Funcion3(float *, int);

void main(void)
{
int FAB[15][12] = {0};                /* Inicialización en 0 del arreglo FAB. */
float COS[15], VEN[15];
Lectura1(FAB);
Lectura2(COS, 15);
Funcion1(FAB, 15, 12, COS, VEN);
```

```
Funcion2(VEN, 15);
Funcion3(VEN, 15);
}
void Lectura1(int A[][12])
/* Esta función se utiliza para almacenar en el arreglo bidimensional las
➥diferentes transacciones que representan las ventas de los diferentes
➥productos. El fin de datos está dado por –1. */
{
int MES, PRO, CAN;
printf("\nIngrese mes, tipo de producto y cantidad vendida: ");
scanf("%d %d %d", &MES, &PRO, &CAN);
while (MES!= -1 && PRO!= -1 && CAN!=-1)
{
     A[MES-1][PRO-1] += CAN;
     printf("Ingrese mes, tipo de producto y cantidad vendida: ");
     scanf("%d %d %d", &MES, &PRO, &CAN);
}
}

void Lectura2(float A[], int N)
/* Esta función se utiliza para leer los precios de venta de los diferentes
➥productos. */
{
int I;
for (I=0; I<N; I++)
{
     printf("Ingrese costo del producto %d: ", I+1);
     scanf("%f", &A[I]);
}
}

void Funcion1(int A[][12], int F, int C, float V1[], float V2[])
/* Esta función se utiliza para calcular el monto de venta anual de cada uno
➥de los productos. Observa que el resultado se almacena en un arreglo
➥unidimensional que se utilizará posteriormente. */
{
int I, J, SUM;
printf("\n");
for (I=0; I< F; I++)
{
     SUM = 0;
     for (J=0; J<C; J++)
            SUM += A[I][J];
     V2[I] = V1[I] * SUM;
     printf("\nTotal de ventas del producto %d: %8.2f", I+1, V2[I]);
}
}

void Funcion2(float A[], int C)
/* Esta función se utiliza para calcular el monto total de ventas de la fábrica. */
{
```

```
int I;
float SUM = 0.0;
for (I=0; I<C; I++)
       SUM += A[I];
printf("\n\nTotal de ventas de la fábrica: %.2f", SUM);
}

void Funcion3(float A[], int C)
/* Esta función se utiliza para obtener el tipo de producto que más se ha vendido
➥y el monto de las ventas de dicho producto. */
{
int I, TPR = 0;
float VEN = A[0];
for (I=1; I<C; I++)
      if (VEN < A[I])
      {
           TPR = I;
           VEN = A[I];
      }
printf("\n\nTipo de Producto más vendido: %d \t   Ventas: %.2f", TPR + 1, VEN);
}
```

Problema PR6.5

Escribe un programa en **C** que, al recibir como dato una matriz, calcule su traspuesta. La traspuesta de una matriz se obtiene al escribir las filas de la matriz como columnas.

Dato: `MAT[M][N]` (arreglo bidimensional de tipo entero de `M` filas y `N` columnas, `1 ≤ M ≤ 50` y `1 ≤ N ≤ 50`).

Programa 6.8

```
#include <stdio.h>

/* Traspuesta.
El programa calcula la traspuesta de una matriz. */

const int MAX = 50;

void Lectura(int [][MAX], int, int);
void Traspuesta(int [][MAX],int [][MAX], int, int);  /* Prototipos de funciones. */
void Imprime(int [][MAX], int, int);

void main(void)
{
```

```
int MAT[MAX][MAX], TRA[MAX][MAX];
int FIL, COL;
do
{
      printf("Ingrese el número de filas de la matriz: ");
      scanf("%d", &FIL);
}
while (FIL > MAX || FIL < 1);
/* Se verifica que el número de filas sea correcto. */
do
{
      printf("Ingrese el número de columnas de la matriz: ");
      scanf("%d", &COL);
}
while (COL > MAX || COL < 1);
/* Se verifica que el número de columnas sea correcto. */
Lectura(MAT, FIL, COL);
Traspuesta(MAT, TRA, FIL, COL);
Imprime(TRA, COL, FIL);
}

void Lectura(int A[][MAX], int F, int C)
/* Esta función se utiliza para una matriz de tipo entero de F filas y C
➥columnas. */
{
int I, J;
for (I=0; I<F; I++)
       for (J=0; J<C; J++)
       {
             printf("Ingrese el elemento %d %d: ", I+1, J+1);
             scanf("%d", &A[I][J]);
       }
}

void Traspuesta(int M1[][MAX],int M2[][MAX],int F, int C)
/* Esta función se utiliza para calcular la traspuesta. */
{
int I, J;
for (I=0; I< F; I++)
      for (J=0; J<C; J++)
           M2[J][I] = M1[I][J];
}

void Imprime(int A[][MAX], int F, int C)
/* Esta función se utiliza para escribir una matriz de tipo entero de F filas
➥y C columnas —en este caso la traspuesta. */
{
int I, J;
for (I=0; I<F; I++)
     for (J=0; J<C; J++)
          printf("\nElemento %d %d: %d ", I+1, J+1, A[I][J]);
}
```

Problema PR6.6

Escribe un programa en **C** que, al recibir como dato un arreglo bidimensional cuadrado (matriz), asigne valores a un arreglo unidimensional aplicando el siguiente criterio:

$$B[I] = \begin{cases} \sum_{J=0}^{I} MAT[I][J] & Si(I\%3) = 1 \\ \prod_{J=I}^{N-1} MAT[J][I] & Si(I\%3) = 2 \\ (\prod_{J=0}^{N-1} MAT[J][I]/(\sum_{J=0}^{I} MAT[J][I]) & \textit{De otra forma} \end{cases}$$

Dato: `MAT[N][N]` (arreglo bidimensional cuadrado de tipo entero, `1 ≤ N ≤ 50`).

Programa 6.9

```
#include <stdio.h>

/* Asigna.
El programa, al recibir un arreglo bidimensional cuadrado, asigna elementos en
➥función del módulo (residuo) a un arreglo unidimensional. */

void Lectura(int [][10], int);
void Calcula(int [][10], float [], int);
float Mod0(int [][10], int, int);
float Mod1(int [][10], int);                    /* Prototipos de funciones. */
float Mod2(int [][10], int, int);
void Imprime(float [10], int);

void main(void)
{
int MAT[10][10],TAM;
float VEC[10];
do
{
      printf("Ingrese el tamaño de la matriz: ");
      scanf("%d", &TAM);
}
while (TAM >10 || TAM < 1);
Lectura(MAT, TAM);
```

```
Calcula(MAT, VEC, TAM);
Imprime(VEC, TAM);
}

void Lectura(int A[][10], int N)
/* La función Lectura se utiliza para leer un arreglo bidimensional cuadrado
➥de tipo entero. */
{
int I, J;
for (I=0; I<N; I++)
     for (J=0; J<N; J++)
     {
           printf("Ingrese el elemento %d %d: ", I+1, J+1);
           scanf("%d", &A[I][J]);
     }
}

void Calcula(int A[][10],float B[], int N)
/* Esta función se utiliza tanto para calcular el módulo entre el índice del
➥arreglo unidimensional y 3, como para llamar a las funciones
➥correspondientes para resolver el problema. */
{
int I;
for (I=0; I<N; I++)
    switch (I%3)
    {
        case 1: B[I] = Mod1 (A,I);
           break;
        case 2: B[I] = Mod2 (A,I,N);
           break;
        default: B[I] = Mod0 (A,I,N);
           break;
    }
}

float Mod0 (int A[][10],int K, int M)
/* Esta función calcula el cociente entre una productoria y una sumatoria. */
{
int I;
float PRO = 1.0, SUM  = 0.0;
for (I=0; I<M; I++)
{
      PRO *= A[I][K];
      SUM += A[I][K];
}
return (PRO / SUM);
}
float Mod1(int A[][10], int N)
/* Esta función obtiene el resultado de una sumatoria. */
{
```

```
int I;
float SUM = 0.0;
for (I=0; I<=N; I++)
     SUM += A[N][I];
return (SUM);
}

float Mod2 (int A[][10],int N, int M)
/* Esta función obtiene el resultado de la productoria. */
{
int I;
float PRO = 1.0;
for (I=N; I<M; I++)
     PRO *= A[I][N];
return (PRO);
}

void Imprime(float B[], int N)
/* Esta función se utiliza para escribir un arreglo unidimensional de tipo
➥real de N elementos. */
{
int I;
for (I=0; I<N; I++)
      printf("\nElemento %d: %.2f ", I, B[I]);
}
```

Problema PR6.7

Escribe un programa en **C** que genere un cuadrado mágico (CM). Un CM se representa por medio de una matriz cuadrada de orden `N`, impar, y contiene los números comprendidos entre `1` y `N*N`. En un CM la suma de cualquiera de las filas, columnas y diagonales principales siempre es la misma. El cuadrado mágico se genera aplicando los siguientes criterios:

1. El primer número (1) se coloca en la celda central de la primera fila.
2. El siguiente número se coloca en la celda de la fila anterior y columna posterior.
3. La fila anterior al primero es el último. La columna posterior a la última es la primera.
4. Si el número es un sucesor de un múltiplo de `N`, no aplique la regla 2. Coloque el número en la celda de la misma columna de la fila posterior.

Programa 6.10

```
#include <stdio.h>

/* Cuadrado mágico.
El programa genera un cuadrado mágico siguiendo los criterios enunciados
➥anteriormente. */

const int MAX = 50;

void Cuadrado(int [][MAX], int);
void Imprime(int [][MAX], int);           /* Prototipos de funciones. */

void main(void)
{
int CMA[MAX][MAX],TAM;
do
{
     printf("Ingrese el tamaño impar de la matriz: ");
     scanf("%d", &TAM);
}
while ((TAM > MAX || TAM < 1) && (TAM % 2));
/* Se verifica el tamaño del arreglo y el orden (impar) del mismo. */
Cuadrado(CMA, TAM);
Imprime(CMA, TAM);
}

void Cuadrado(int A[][MAX],int N)
/* Esta función se utiliza para formar el cuadrado mágico. */
{
int I = 1, FIL = 0, COL = N / 2, NUM = N * N;
while (I <= NUM)
{
    A[FIL][COL] = I;
    if (I%N != 0)
    {
          FIL = (FIL - 1 + N) % N;
          COL = (COL + 1) % N;
    }
    else
        FIL++;
    I++;
}
}


void Imprime(int A[][MAX], int N)
/* Esta función se utiliza para escribir el cuadrado mágico. */
{
int I, J;
for (I=0; I<N; I++)
     for (J=0; J<N; J++)
           printf("\nElemento %d %d: %d",I+1, J+1, A[I][J]);
}
```

Problema PR6.8

Escribe un programa en **C** que, al recibir como datos los valores mensuales del año 2004 de las acciones de los cinco fondos de inversión que maneja una casa de bolsa de la Ciudad de México, y por otra parte los precios de esas acciones al 31 de diciembre del año 2003, realice lo siguiente:

a) Calcule el rendimiento anual de los cinco fondos de inversión.

b) Obtenga el promedio anual de las acciones de cada uno de los fondos de inversión.

c) Obtenga el fondo que obtuvo el mayor rendimiento, así como el que obtuvo el peor rendimiento. Escribe además los porcentajes correspondientes.

Dato: `FON [5][12], PRE[5]`

Donde: `FON` es un arreglo bidimensional de tipo real de cinco filas y 12 columnas, que almacena el valor mensual de las acciones de los cinco fondos de inversión.

`PRE` es un arreglo unidimensional de tipo real de cinco elementos, que almacena el precio de las acciones de los fondos al 31 de diciembre del año 2003.

Programa 6.11

```
#include <stdio.h>

/* Casa de bolsa.
El programa, al recibir como datos los precios mensuales de las acciones de sus
➥cinco fondos de inversión, además del precio de las acciones al 31 de diciembre
➥del 2003, genera información estadística importante para la empresa. */

void LecturaM(float [][12], int, int);
void LecturaV(float *, int);
void F1(float [][12], int, int, float *, float *);   /* Prototipos de funciones. */
void F2(float [][12], int, int);
void F3(float *, int);

void main(void)
{
float FON[5][12], PRE[5], REN[5];
/* REN es un arreglo unidimensional de tipo real que se utilizará para almacenar
➥el rendimiento anual de los fondos de inversión. */
```

```
LecturaM(FON, 5, 12);
LecturaV(PRE, 5);
F1(FON, 5, 12, PRE, REN);
F2(FON, 5, 12);
F3(REN, 5);
}

void LecturaM(float A[][12], int F, int C)
/* Esta función se utiliza para leer un arreglo bidimensional de tipo real de F
➥filas y C columnas. */
{
int I, J;
for (I=0; I<F; I++)
     for (J=0; J<C; J++)
     {
           printf("Precio fondo %d\t mes %d:  ", I+1, J+1);
           scanf("%f", &A[I][J]);
     }
}

void LecturaV(float A[], int T)
/* Esta función se utiliza para leer un arreglo unidimensional de tipo real de T
➥elementos. */
{
int I;
printf("\n");
for (I=0; I<T; I++)
{
      printf("Precio Fondo %d al 31/12/2003:  ", I+1);
      scanf("%f", &A[I]);
}
}

void F1(float A[][12], int F, int C, float B[], float V[])
{
/* La función F1 se utiliza para calcular el rendimiento anual de los fondos de
➥inversión. El resultado se almacena en el arreglo unidimensional V. */
int I;
printf("\nRENDIMIENTOS ANUALES DE LOS FONDOS");
for(I=0; I<F; I++)
{
     V[I] = (A[I][C-1] - B[I]) / B[I] * 100;
     printf("\nFondo %d:   %.2f", I+1, V[I]);
}
}

void F2(float A[][12], int F, int C)
{
/* Esta función calcula el promedio anualizado de las acciones de los diferentes
➥fondos. */
```

```
int I, J;
float SUM, PRO;
printf("\n\nPROMEDIO ANUALIZADO DE LAS ACCIONES DE LOS FONDOS");
for(I=0; I<R; I++)
{
     SUM = 0;
     for(J=0; J<C; J++)
         SUM += A[I][J];
     PRO = SUM / C;
     printf("\nFondo %d: %.2f", I+1, PRO);
}
}

void F3(float A[], int F)
/* Esta función permite calcular los fondos con el mejor y peor rendimiento. */
{
float ME = A[0], PE = A[0];
int M = 0, P = 0, I;
for (I=1; I<F; I++)
{
      if (A[I] > ME)
      {
            ME = A[I];
            M = I;
      }
      if (A[I] < PE)
      {
            PE = A[I];
            P = I;
      }
}
printf("\n\nMEJOR Y PEOR FONDO DE INVERSION");
printf("\nMejor fondo: %d\tRendimiento: %6.2f", M+1, ME);
printf("\nPeor fondo: %d\tRendimiento: %6.2f", P+1, PE);
}
```


Problema PR6.9

En el arreglo tridimensional LLU se almacenan las lluvias mensuales registradas en milímetros, en las 24 provincias de Argentina, durante los últimos 10 años. Escribe un programa en **C** que permita resolver lo siguiente:

a) La provincia que tuvo el mayor registro de precipitación pluvial durante los últimos 10 años.

b) La provincia que tuvo el menor registro de lluvias en el último año.

c) El mes que tuvo el mayor registro de lluvias en la provincia 18 en el quinto año.

Nota: En todos los casos se debe escribir además el registro de lluvias correspondiente.

Dato: `LLU[24][12][10]` (arreglo tridimensional de tipo real que almacena las lluvias mensuales en las 24 provincias durante los últimos 10 años).

Programa 6.12

```
#include <stdio.h>

/* Lluvias.
El programa, al recibir como dato un arreglo tridimensional que contiene
➥información sobre lluvias, genera información estadística. */

const int PRO = 24;
const int MES = 12;
const int AÑO = 10;

void Lectura(float [PRO][MES][AÑO], int, int, int);
void Funcion1(float [PRO][MES][AÑO], int, int, int);
void Funcion2(float [PRO][MES][AÑO], int, int, int);  /* Prototipos de funciones. */
void Funcion3(float [PRO][MES][AÑO], int, int, int);

void main(void)
{
float LLU[PRO][MES][AÑO];
Lectura(LLU, PRO, MES, AÑO);
Funcion1(LLU, PRO, MES, AÑO);
Funcion2(LLU, PRO, MES, AÑO);
Funcion3(LLU, 18, MES, 5);
}

void Lectur (float A[][MES][AÑO], int F, int C, int P)
/* Esta función se utiliza para leer un arreglo tridimensional de tipo real de
➥F filas, C columnas y P planos de profundidad. */
{
int K, I, J;
for (K=0; K<P; K++)
     for (I=0; I<F; I++)
         for (J=0; J<C; J++)
         {
               printf("Año: %d\tProvincia: %d\tMes: %d", K+1, I+1, J+1);
               scanf("%f", &A[I][J][K]);
         }
}

void Funcion1(float A[][MES][AÑO],int F, int C, int P)
/* Esta función se utiliza para localizar la provincia que tuvo el mayor registro
➥de precipitación pluvial en los últimos 10 años. Escribe además el registro
➥correspondiente. */
```

```
{
int I, K, J, EMAY = -1;
float ELLU = -1.0, SUM;
for (I=0; I<F; I++)
{
      SUM = 0.0;
      for (K=0; K<P; K++)
            for (J=0; J<C; J++)
                  SUM += A[I][J][K];
      SUM /= P * C;
      if (SUM > ELLU)
      {
          ELLU = SUM;
          EMAY = I;
      }
}
printf("\n\nProvincia con mayor registro de lluvias: %d", EMAY+1);
printf("\nRegistro: %.2f", ELLU);
}

void Funcion2(float A[][MES][AÑO],int F, int C, int P)
/* Esta función se utiliza para localizar la provincia que tuvo el menor registro
➥de lluvias en el último año. Escribe además el registro correspondiente. */
{
int I, J, EMEN;
float ELLU = 1000, SUM;
for (I=0; I<F; I++)
{
      SUM = 0;
      for (J=0; J<C; J++)
            SUM += A[I][J][P-1];
      SUM /= C;
      if (SUM < ELLU)
      {
             ELLU = SUM;
             EMEN = I;
      }
}
printf("\n\nProvincia con menor registro anual de lluvias en el último año: %d",
       EMEN+1);
printf("\nRegistro anual: %.2f", ELLU);
}

void Funcion3(float A[][MES][AÑO],int F, int C, int P)
/* Esta función se utiliza para localizar el mes con mayor registro de lluvias en
➥la provincia 18 en el quinto año. Escribe además el registro correspondiente. */
{
int J, EMES = -1;
float ELLU = -1.0;
```

```
for (J=0; J<C; J++)
{
      if (A[F-1][J][P-1] > ELLU)
      {
            ELLU = A[F-1][J][P-1];
            EMES = J;
      }
}
printf("\n\nMes: %d Lluvias: %.2f", EMES+1, ELLU);
}
```

Problema PR6.10

En el arreglo tridimensional `PRO` se almacenan las ventas mensuales de los últimos ocho años de los tres departamentos de una empresa textil: *hilos, lanas y licra*. Escribe un programa en **C** que obtenga lo siguiente:

a) Las ventas totales de la empresa en el segundo año.

b) El departamento que tuvo las mayores ventas en el último año, incluyendo también el importe de las ventas.

c) El departamento, mes y año con la mayor venta, incluyendo el importe de las ventas.

Dato: `PRO[12][3][8]` (donde `PRO` es un arreglo tridimensional de tipo real que almacena las ventas mensuales de los tres departamentos en los últimos ocho años).

Programa 6.13

```
#include <stdio.h>

/* Empresa textil.
El programa, al recibir un arreglo tridimensional que contiene información
➥sobre las ventas mensuales de tres departamentos en los últimos ocho años,
➥genera información estadística valiosa para la empresa. */

const int MES = 12;
const int DEP =  3;
const int AÑO = 8;

void Lectura(float [MES][DEP][AÑO], int, int, int);
void Funcion1(float [MES][DEP][AÑO], int, int, int);
void Funcion2(float [MES][DEP][AÑO], int, int, int);   /* Prototipos
➥de funciones. */
```

```
void Funcion3(float [MES][DEP][AÑO], int, int, int);

void main(void)
{
float PRO[MES][DEP][AÑO];
Lectura(PRO, MES, DEP, AÑO);
Funcion1(PRO, MES, DEP, 2);
Funcion2(PRO, MES, DEP, AÑO);
Funcion3(PRO, MES, DEP, AÑO);
}

void Lectura(float A[][DEP][AÑO], int F, int C, int P)
/* La función Lectura se utiliza para leer un arreglo tridimensional de tipo
➥real de F filas, C columnas y P planos de profundidad. */
{
int K, I, J;
for (K=0; K<P; K++)
      for (I=0; I<F; I++)
          for (J=0; J<C; J++)
          {
               printf("Año: %d\tMes: %d\tDepartamento: %d  ", K+1, I+1, J+1);
               scanf("%f", &A[I][J][K]);
          }
}

void Funcion1(float A[][DEP][AÑO],int F, int C, int P)
/* Esta función se utiliza para obtener las ventas totales de la empresa
➥en el segundo año. */
{
int I, J;
float SUM = 0.0;
for (I=0; I<F; I++)
    for (J=0; J<C; J++)
         SUM += A[I][J][P-1];
printf("\n\nVentas totales de la empresa en el segundo año: %.2f", SUM);
}

void Funcion2(float A[][DEP][AÑO],int F, int C, int P)
/* Esta función se utiliza para obtener el departamento que tuvo las mayores
➥ventas en el último año. Genera además el importe de las ventas. */
{
int I, J;
float SUM1 = 0, SUM2 = 0, SUM3 = 0;
for (I=0; I<F; I++)
    for (J=0; J<C; J++)
          switch (J+1)
          {
              case 1: SUM1 += A[I][J][P-1];
                 break;
              case 2: SUM2 += A[I][J][P-1];
                 break;
```

```
                case 3: SUM3 += A[I][J][P-1];
                   break;
             }
if (SUM1 > SUM2)
     if (SUM1 > SUM3)
     {
          printf("\n\nDepartamento con mayores ventas en el último año: Hilos");
          printf("  Ventas:%.2f", SUM1);
     }
     else
     {
          printf("\n\nDepartamento con mayores ventas en el último año: Licra");
          printf("  Ventas:%.2f", SUM3);
     }
else
    if (SUM2 > SUM3)
    {
          printf("\n\nDepartamento con mayores ventas en el último año: Lanas");
          printf("  Ventas:%.2f", SUM2);
    }
    else
    {
          printf("\n\nDepartamento con mayores ventas en el último año: Licra");
          printf("  Ventas:%.2f", SUM3);
    }
}

void Funcion3(float A[][DEP][AÑO],int F, int C, int P)
/* Esta función se utiliza para obtener el departamento, mes y año con la mayor
➥venta. Escribe también el monto de las ventas. */
{
int K, I, J, DE, ME, AN;
float VEN = -1.0;
for (K=0; K<P; K++)
    for (I=0; I< F; I++)
        for (J=0; J<C; J++)
            if (A[I][J][K] > VEN)
            {
                 VEN = A[I][J][K];
                 DE = J;
                 ME = I;
                 AN = K;
            }
printf("\n\nDepartamento: %d\tMes: %d\tAño: %d", DE+1, ME+1, AN+1);
printf("\tVentas: %.2f", VEN);
}
```

Problemas suplementarios

Problema PS6.1

Escribe un programa en **C** que coloque un 1 en las diagonales principales de una matriz cuadrada. El resto se debe completar con 0. Observa la figura 6.7, que muestra cómo debe quedar la matriz.

1	0	0	1
0	1	1	0
0	1	1	0
1	0	0	1

FIGURA 6.7
Matriz cuadrada.

Dato: `MAT[N][N]` (arreglo bidimensional cuadrado de `NxN` elementos, `1 ≤ N ≤ 100`).

Problema PS6.2

Construye un programa que, al recibir los montos de ventas mensuales de cinco departamentos de una fábrica, proporcione la siguiente información:

a) Las ventas mensuales de la fábrica, incluido el monto anual.
b) El departamento que tuvo la mayor venta en el mes de julio, incluyendo el monto de la venta.
c) El mes en el que se obtuvieron las mayores y menores ventas del departamento 3.

Dato: `VEN[5][12]` (arreglo bidimensional de tipo real que almacena las ventas mensuales de cinco departamentos de una fábrica).

Problema PS6.3

Escribe un programa que intercambie las `M` filas de un arreglo bidimensional. Los elementos de la primera fila se intercambian con los de la última fila, los del segundo con los de la penúltima, y así sucesivamente.

Dato: `MAT [M][N]` (arreglo bidimensional de tipo real de `M` filas y `N` columnas, $1 \leq M \leq 50$ y $1 \leq N \leq 50$).

Problema PS6.4

Construye un programa en **C** que, al recibir una matriz cuadrada impar, determine si la misma se puede considerar un cuadrado mágico. Los criterios para formar un cuadrado mágico se especifican en el problema PR6.7.

Dato: `CUA[N][N]` (arreglo bidimensional cuadrado impar de `NxN` elementos, $1 \leq N \leq 100$).

Problema PS6.5

Escribe un programa que, al recibir como datos dos arreglos bidimensionales `A[M][N]` y `B[N][P]`, calcule el producto de dichos arreglos y almacene el resultado en el arreglo bidimensional `C[M][P]`.

Datos: `A[M][N], B[N][P]` (arreglos bidimensionales reales de `MxN` y `NxP` elementos, respectivamente, $1 \leq M \leq 50$, $1 \leq N \leq 50$, $1 \leq P \leq 50$).

Problema PS6.6

Escribe un programa en **C** que, al recibir como datos dos arreglos bidimensionales enteros `A[M][N]` y `B[N][M]`, calcule la suma de `A` más la traspuesta de `B` ($A + B^T$) y almacene el resultado en el arreglo bidimensional `C`.

Datos: `A[M][N], B[N][M]` (arreglos bidimensionales enteros de `MxN` y `NxM` elementos, respectivamente, $1 \leq M \leq 50$, $1 \leq N \leq 50$).

Problema PS6.7

Escribe un programa en **C** que, al recibir como dato un arreglo bidimensional de tipo entero, recorra este arreglo en forma de espiral. Observa la siguiente figura.

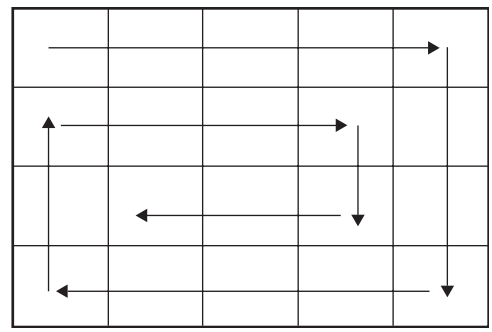


FIGURA 6.8
Recorrido en forma de espiral en una matriz.

Dato: `MAT [M][N]` (arreglo bidimensional de tipo real de `M` filas y `N` columnas, `1 ≤ M ≤ 50` y `1 ≤ N ≤ 50`).

Problema PS6.8

Escribe un programa en **C** que, al recibir como dato un arreglo bidimensional de tipo entero, recorra este arreglo columna a columna, tal como se observa en la siguiente figura.

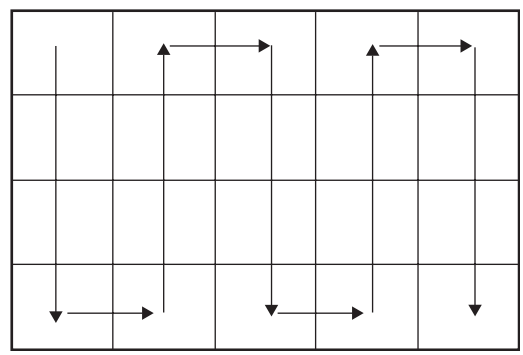


FIGURA 6.9
Recorrido columna a columna en una matriz.

Dato: `MAT [M][N]` (arreglo bidimensional de tipo real de `M` filas y `N` columnas, `1 ≤ M ≤ 50` y `1 ≤ N ≤ 50`).

Problema PS6.9

Escribe un programa en **C** que permita resolver *el problema de las ocho reinas*. Éste es un problema muy conocido por los amantes del ajedrez. Consiste en colocar ocho reinas en un tablero de ajedrez vacío (8×8) de forma que ninguna de ellas pueda atacar a las restantes. Es decir, ningún par de reinas puede estar en la misma fila, columna o diagonal.

Problema PS6.10

Escribe un programa en **C** que permita resolver *el problema del recorrido del caballo*. Al igual que el problema anterior, éste también es muy conocido por los aficionados al ajedrez. El problema consiste en recorrer en forma completa el tablero de ajedrez (8×8) con el caballo, tocando solamente una vez cada casilla. En total el recorrido completo consiste de 64 movimientos.

Problema PS6.11

En la Secretaría de Turismo de México se almacena información sobre el número de visitantes mensuales de los 10 principales centros turísticos del país, en los últimos cinco años. Construye un programa en **C** que proporcione la siguiente información:

a) El total de visitantes a cada uno de los centros turísticos.
b) Los centros turísticos más y menos visitados en los últimos cinco años, junto con el número de visitantes a cada uno de estos centros.
c) El mes del último año con mayor y menor afluencia turística, junto con el número de visitantes.

Dato: `SEC[10][12][5]` (arreglo tridimensional de tipo entero que contiene información sobre los visitantes a los centros turísticos más importantes del país).

Problema PS6.12

En la Federación Mexicana de Fútbol se lleva información sobre el número de asistentes mensuales en los diez estadios más importantes del país durante los últimos cinco años. Construye un programa que genere la siguiente información:

a) El número de asistentes del último año en cada estadio, ordenados de mayor a menor.

b) Los estadios que tuvieron la mayor y menor afluencia de público en los últimos cinco años, junto con el número de asistentes.

c) El mes del último año en que cada estadio tuvo la mayor afluencia de público, junto con el número de asistentes.

Dato: `FUT[7][12][5]` (arreglo tridimensional de tipo entero que contiene información sobre asistentes a los estadios de fútbol más importantes del país).

CAPÍTULO 7

Caracteres y cadenas de caracteres

7.1. Introducción

Los datos que una computadora puede procesar se clasifican en **simples** y **estructurados**. Esta clasificación se origina en el número de celdas o casillas de memoria que se necesitan para almacenar un dato. Los datos simples tienen la particularidad de ocupar una sola casilla —posición— de memoria. Los enteros, los reales y los **caracteres** son ejemplos de tipos de datos simples.

Por otra parte, los datos estructurados ocupan un **grupo** de casillas de memoria. Un dato estructurado tiene varios componentes, que pueden ser tipos de datos simples, o bien, estructurados. Los componentes del nivel más bajo de un tipo estructurado son siempre tipos de datos simples. Los **arreglos** —capítulos 5 y 6—, los **registros** —capítulo 8— y las **cadenas de caracteres** son ejemplos de tipos de datos estructurados.

En este capítulo estudiaremos los **caracteres** y las **cadenas de caracteres**.

7.2. Caracteres

Un **caracter*** es un tipo de dato simple que representa un número, una letra o cualquier caracter especial disponible del teclado de la máquina. Cuando se asigna un caracter a una variable tipo char, éste siempre se debe escribir entre apóstrofos ‘ ‘.

EJEMPLO 7.1

En el siguiente programa podemos observar diferentes formas para declarar un caracter, así como las funciones de entrada (**getchar** y **scanf**) y salida (**putchar** y **printf**) que permiten trabajar con ellos. Las funciones pertenecen a la biblioteca estándar de entrada/salida `stdio.h`.

Programa 7.1

```
#include <stdio.h>

/* Funciones para el manejo de caracteres de la biblioteca stdio.h */

void main(void)
{
char p1, p2, p3 = '$';
/* Declaración de las variables tipo caracter p1, p2 y p3. Observa que a p3 se le
➥asigna el símbolo $. */
printf("\nIngrese un caracter: ");
p1=getchar();         /* Se utiliza la función getchar para leer un caracter. */
putchar(p1);          /* Se utiliza la función putchar para escribir un
➥caracter. */
printf("\n");
fflush(stdin);
/* Luego de leer un caracter siempre es conveniente escribir la función fflush
➥para limpiar el búfer, porque generalmente queda con basura y genera un error
➥en la ejecución del programa. El error se produce porque cuando se ingresa un
➥dato se oprime el return y luego cuando volvemos a leer un caracter o una
➥cadena de caracteres se toma a ese return como el nuevo dato ingresado. */

printf("\nEl caracter p3 es: ");
putchar(p3);
/* Se utiliza la función putchar para escribir el caracter almacenado en p3. */
printf("\n");
```

* La palabra carácter, invariablemente debe ir acentuada; pero por consideración a los usuarios de informática, que suelen usarla sin acento, en este libro la aplicaremos de esta manera.

```
printf("\nIngrese otro caracter: ");
fflush(stdin);
scanf("%c", &p2);
/* Se puede utilizar scanf con el formato de variable %c para leer un caracter. */
printf("%c", p2);
/* Se puede utilizar printf con el formato de variable %c para escribir un
➥caracter. */
}
```

EJEMPLO 7.2

En la tabla 7.1 se presenta un resumen de las funciones más importantes para el manejo de caracteres de la biblioteca **ctype.h**.

TABLA 7.1. Funciones de la biblioteca `ctype.h`

Función	*Explicación*
`isdigit(p)`	Regresa 1 si `p` es un dígito y 0 en caso contrario.
`isalpha(p)`	Regresa 1 si `p` es una letra y 0 en caso contrario.
`islower(p)`	Regresa 1 si `p` es una letra minúscula y 0 en caso contrario.
`isupper(p)`	Regresa 1 si `p` es una letra mayúscula y 0 en caso contrario.
`tolower(p)`	Convierte de mayúscula a minúscula. Si la letra es minúscula no modifica su valor.
`toupper(p)`	Convierte de minúscula a mayúscula. Si la letra es mayúscula no modifica su valor.

Veamos en el siguiente programa la aplicación de las principales funciones de esta biblioteca.

Programa 7.2

```
# include <stdio.h>
# include <ctype.h>

/* Funciones para el manejo de caracteres de la biblioteca ctype.h. */

void main(void)
{
```

```
char p1;
printf("\nIngrese un caracter para analizar si éste es un dígito: ");
p1 = getchar();
if (isdigit (p1))
/* La función isdigit regresa 1 si p1 es un dígito y 0 en caso contrario. */
    printf("%c es un dígito \n", p1);
else
    printf("%c no es un dígito \n", p1);

fflush(stdin);
printf("\nIngrese un caracter para examinar si éste es una letra: ");
p1 = getchar();
if (isalpha (p1))
/* La función isalpha regresa 1 si p1 es una letra y 0 en caso contrario. */
    printf("%c es una letra \n", p1);
else
    printf("%c no es una letra \n", p1);

fflush(stdin);
printf("\nIngrese un caracter para examinar si éste es una letra minúscula: ");
p1 = getchar();
if (isalpha (p1))
    if (islower (p1))
       /* La función islower regresa 1 si p1 es una letra minúscula y 0 en caso
       ➥contrario.
       La función isupper, por otra parte, regresa 1 si p1 es una letra mayúscula
       ➥y 0 en caso contrario. */
       printf("%c es una letra minúscula \n", p1);
    else
       printf("%c no es una letra minúscula \n", p1);
else
   printf("%c no es una letra \n", p1);

fflush(stdin);
printf("\nIngrese una letra para convertirla de mayúscula a minúscula: ");
p1 = getchar();
if (isalpha (p1))
     if (isupper(p1))
          printf("%c fue convertida de mayúscula a minúscula \n", tolower(p1));
          /* La función tolower convierte de mayúscula a minúscula. Si la
          ➥letra es minúscula no la modifica. La función toupper, por otra parte,
          ➥convierte de minúscula a mayúscula. Si la letra es mayúscula no la
          ➥modifica.  */
      else
         printf("%c es una letra minúscula \n", p1);
else
    printf("%c no es una letra \n", p1);
}
```

7.3. Cadenas de caracteres

Una **cadena de caracteres** es un tipo de datos estructurado compuesto por **caracteres**. En el lenguaje de programación **C**, una cadena de caracteres se define como un arreglo de caracteres que termina con el caracter nulo (‘\0’). El acceso a una cadena se realiza por medio de un apuntador que señala al primer caracter de la cadena. Cuando se asigna una cadena de caracteres a una variable de tipo char, ésta se debe escribir entre comillas “ ”. Observemos a continuación los siguientes ejemplos.

EJEMPLO 7.3

En el siguiente programa podemos observar diferentes maneras para declarar cadenas de caracteres y asignarles valores. Estudiaremos las funciones de entrada y salida de la biblioteca `stdio.h`: `gets`, `scanf`, `puts` y `printf`.

Programa 7.3

```
#include <stdio.h>

/* Funciones para el manejo de cadenas de caracteres de la biblioteca stdio.h. */

void main(void)
{
char *cad0 = "Buenos días";   /* En este caso se asignan 11 caracteres más el
➥caracter de terminación '\0' a la posición de memoria a la que apunta la
➥variable cad0 —apuntador del tipo cadena de caracteres. */

char cad1[20] = "Hola";         /* Se asignan cuatro caracteres más el caracter
➥de terminación a la variable tipo char cad1. Observa que cad1 tiene espacio
➥para 20 caracteres.*/

char cad2[] = "México";         /* En este caso se asignan seis caracteres (más
➥el caracter de terminación) a la variable cad2. Observa que cad2 no tiene espacio
➥reservado como cad1; por lo tanto, acepta cualquier número de caracteres. */

char cad3[] = {'B', 'i', 'e', 'n', 'v', 'e', 'n', 'i', 'd', 'o', '\0'};
/* Observa otra forma de asignación de valores a la variable cad3. */

char cad4[20], cad5[20], cad6[20];

printf("\nLa cadena cad0 es: ");
puts(cad0);
/* La función puts es la más apropiada para escribir cadenas de caracteres.
➥Observa que esta función baja automáticamente una línea después de imprimir
➥la cadena. */
```

```
printf("\nLa cadena cad1 es: ");
printf("%s", cad1);
/* La función printf, con el formato de variable %s, también se puede utilizar
➥para escribir cadenas de caracteres. Baja automáticamente una línea después
➥de escribir la cadena.*/

printf("\nLa cadena cad2 es: ");
puts(cad2);
printf("\nLa cadena cad3 es: ");
puts(cad3);

printf("\nIngrese una línea de texto —se lee con gets—: \n");
/* La función gets es la más apropiada para leer cadenas de caracteres. */
gets(cad4);
printf("\nLa cadena cad4 es: ");
puts(cad4);
fflush(stdin);

printf("\nIngrese una línea de texto —se lee con scanf—: \n");
scanf("%s", cad5);
/* La función scanf, con el formato de variable %s, también se puede utilizar
➥para leer una cadena de caracteres, aunque con algunas restricciones. Si la
➥cadena está formada por varias palabras sólo lee la primera. Por ejemplo, si
➥queremos ingresar la cadena "Buenos días", sólo lee la palabra "Buenos", por
➥ello esta función únicamente es útil si conocemos con anticipación que la
➥cadena que vamos a leer está formada por una sola palabra. */
printf("\nLa cadena cad5 es: ");
printf("%s", cad5);
fflush(stdin);

char p;
int i = 0;
/* La declaración de variables siempre se debe realizar en la parte inicial del
➥programa. En este caso se colocan en esta sección (char p e int i = 0) para
➥que puedas observar la relación directa con las líneas de programación que se
➥muestran a continuación. */
printf("\nIngrese una línea de texto —se lee cada caracter con getchar—: \n");
➥/* Se utiliza la función getchar para leer caracteres de la línea de texto y
➥asignarlos a la variable de tipo cadena de caracteres cad6. Observa que se leen
➥caracteres mientras no se encuentre al caracter que indica fin de línea '\n'. */

while ((p = getchar())!= '\n')
   cad6[i++] = p;
cad6[i] = '\0';
/* Al final de la cadena se incorpora el caracter de terminación NULL para
➥indicar el fin de la misma. */
printf("\nLa cadena cad6 es: ");
puts(cad6);
}
```

EJEMPLO 7.4

En el siguiente programa podemos observar algunas formas particulares, correctas e incorrectas, para declarar cadenas de caracteres y asignarles valores.

Programa 7.4

```
#include <stdio.h>

/* Declaración de cadenas de caracteres y asignación de valores. */

void main(void)
{
char *cad0;
cad0 = "Argentina";          /* La declaración y la asignación son correctas. */
puts(cad0);

cad0 = "Brasil";
/* Correcto. Se modifica el contenido de la posición en memoria a la que apunta
➥la variable cad0 —apuntador de tipo cadena de caracteres. */
puts(cad0);

char *cad1;
gets(*cad1);    gets(cad1);
/* Incorrecto. Ambas lecturas generan un error en la ejecución del programa.
➥Para que un apuntador de tipo cadena de caracteres se pueda utilizar con la
➥función de lectura gets, es necesario inicializarlo como se hace en la siguiente
➥instrucción. */

char *cad1 = "";
gets(cad1);
/* Correcto. Primero se le asigna un valor a la posición de memoria a la que
➥apunta cad1. Luego podemos modificar el contenido de esta posición de memoria
➥utilizando la función gets. */

char cad1[];
/* Incorrecto. Se genera un error en la compilación del programa, porque no
➥se reserva el espacio correspondiente. */

char cad2[20] = "México";    /* Correcto. */
puts(cad2);
gets(cad2);
/* El valor de una cadena (declarada como cadena[longitud]) se puede modificar
➥por medio de lecturas o utilizando funciones de la biblioteca string.h
(ejemplo 7.6). */
puts(cad2);
```

```
cad2[10] = "Guatemala";
/* Incorrecto. Observa cuidadosamente el caso anterior y analiza la diferencia
➥que existe con éste. Aquí se produce un error en la compilación del programa,
➥al tratar de asignar la cadena de caracteres "Guatemala" al caracter 11 de la
➥cadena. */
}
```

EJEMPLO 7.5

En el siguiente programa podemos observar la aplicación de algunas funciones (`atoi, atof, strtod, atol, strtol`) para el manejo de caracteres de la biblioteca `stdlib.h`.

Programa 7.5

```
#include <stdio.h>
#include <stdlib.h>

/* Funciones para el manejo de caracteres de la biblioteca stdlib.h. */

void main(void)
{
int i;
double d;
long l;
char cad0[20], *cad1;

printf("\nIngrese una cadena de caracteres: ");
gets(cad0);
i = atoi(cad0);
/* La función atoi convierte una cadena de caracteres que contiene números
a ➥un valor de tipo entero. Si la cadena comienza con otro caracter o no
➥contiene números, regresa 0 o el valor queda indefinido. */
printf("\n%s \t %d", cad0, i+3);
/* Se imprime el valor de i+3 para demostrar que i ya fue convertido a un
➥entero.*/

printf("\nIngrese una cadena de caracteres: ");
gets(cad0);
d = atof(cad0);
/* La función atof convierte una cadena de caracteres que contiene números
➥reales a un valor de tipo double. Si la cadena comienza con otro caracter
➥o no contiene números, regresa 0 o el valor queda indefinido. */
printf("\n%s \t %.2lf ", cad0, d+1.50);

d = strtod(cad0, &cad1);
```

```
/* La función strtod convierte una cadena de caracteres que contiene números
➥reales a un valor de tipo double. El resto de la cadena se almacena en el
➥segundo argumento de la función, &cad1, un apuntador de tipo cadena de
➥caracteres. Si la cadena no contiene números o comienza con otro caracter,
➥regresa 0 o el valor queda indefinido. */
printf("\n%s \t %.2lf", cad0, d+1.50);
puts(cad1);

l = atol(cad0);
/* La función atol convierte una cadena de caracteres que contiene números a
➥un valor de tipo long. Si la cadena no contiene números o comienza con
➥otro caracter, regresa 0 o el valor queda indefinido. */
printf("\n%s \t %ld ", cad0, l+10);

l = strtol(cad0, &cad1, 0);
/* La función strtol convierte una cadena de caracteres que contiene números a
➥un valor de tipo long. El resto de la cadena se almacena en el otro argumento
➥de la función, &cad1. El tercer argumento se utiliza para indicar que la
➥cadena puede estar en formato octal, decimal o hexadecimal. Si la cadena no
➥contiene números o comienza con otro caracter, regresa 0 o el valor queda
➥indefinido. */
printf("\n%s \t %ld", cad0, l+10);
puts(cad1);
}
```

En la siguiente tabla se muestran los resultados que arroja este programa.

TABLA 7.2. Funciones de la biblioteca `stdlib.h`

Corrida	*Función*	`cad0`	`cad1`	**`int`** `(i+3)`	**`double`** `(d+1.50)`	**`long`** `(l+2)`
1	atoi(cad)	7sst		10		
1	atof(cad)	7sst			8.5	
1	strtod(cad, &cad1)	7sst	sst		8.5	
1	atol(cad)	7sst				9
1	strtol(cad, &cad1, 0)	7sst	sst			9
2	atoi(cad)	7.89sst 30		10		
2	atof(cad)	7.89sst 30			9.39	
2	strtod(cad, &cad1)	7.89sst 30	sst 30		9.39	
2	atol(cad)	7.89sst 30				9
2	strtol(cad, &cad1, 0)	7.89sst 30	.89sst 30			9

(*Continúa*)

TABLA 7.2. (Continuación)

Corrida	*Función*	**cad0**	**cad1**	**int** (i+**3**)	**double** (d+**1.50**)	**long** (l+**2**)
3	atoi(cad)	s77		3		
3	atof(cad)	s77			1.50	
3	strtod(cad, &cad1)	s77	s77		1.50	
3	atol(cad)	s77				2
3	strtol(cad, &cad1, 0)	s77	s77			2

EJEMPLO 7.6

En el siguiente programa podemos observar la aplicación de las principales funciones (`strcpy`, `strncpy`, `strcat`, `strncat`) para el manejo de cadenas de caracteres de la biblioteca `string.h`.

Programa 7.6

```
#include <stdio.h>
#include <string.h>

/* Funciones de la biblioteca string.h para el manejo de cadenas de
caracteres. */

void main(void)
{
char *cad0 = "Hola México";
char cad1[20], cad2[20], cad3[20] = ", buenos días!!!";

strcpy(cad1, cad0);
/* La función strcpy permite copiar una cadena de caracteres completa. En este
➥caso se copia la cadena cad0 a cad1. Si el espacio reservado para cad1 es
➥menor que el de cad0, se genera un error en la ejecución del programa. */
printf("\nPrueba de la función strcpy. Se copia la cadena cad0 a cad1:
➥%s\n", cad1);

strcpy(cad1, cad3);
printf("\nPrueba de la función strcpy. Se copia la cadena cad3 a cad1:
➥%s\n", cad1);

strcpy(cad1, "XX");
printf("\nPrueba de la función strcpy. Se copia la cadena XX a cad1:
➥%s\n", cad1);

strncpy(cad2, cad0, 4);
```

```
cad2[4] = '\0';
/* La función strncpy permite copiar un número determinado de caracteres a
➥otra cadena de caracteres. En este caso se copian 4 caracteres de la cadena
➥cad0 —segundo argumento— a cad2 —primer argumento. Siempre se debe
➥incorporar al final de la cadena el caracter de terminación. Si el espacio
➥reservado para cad2 es menor que lo que se pretende copiar, se genera
➥un error en la ejecución del programa. */
printf("\nPrueba de la función strncpy. Se copian 4 caracteres de cad0 a
➥cad2: %s\n",
                cad2);

strncpy(cad2, cad3, 3);
cad2[3] = '\0';
printf("\nPrueba de la función strncpy. Se copian 3 caracteres de cad3 a
➥cad2: %s\n",
                cad2);

strcat(cad0, cad3);
/* La función strcat permite incorporar una cadena de caracteres a otra
➥cadena dada. En este caso se agrega la cadena cad3 a cad0. Si el espacio
➥reservado para cad0 es menor a lo que se debe almacenar se genera un error
➥de ejecución. */
printf("\nPrueba de la función strcat. Se incorpora la cadena cad3 a cad0:
➥%s\n", cad0);

strcat(cad1, " YY");
printf("\nPrueba de la función strcat. Se incorpora la cadena YY a cad1:
➥%s\n", cad1);

strcat(cad2, " ");
strncat(cad2, cad0, 4);
printf("\nPrueba de la función strncat. Se incorporan 4 caracteres de cad0
➥a cad2:
                 %s\n", cad2);
/* La función strncat permite incorporar un número determinado de caracteres
➥a una cadena. En este caso se agregan cuatro caracteres de la cadena cad0
➥a cad2. Si el espacio de cad2 es menor a lo que se debe almacenar ocurre
➥un error de ejecución. */

cad0 = strstr(cad0, "México");
printf("\nPrueba de la función strstr. Se trata de localizar la cadena
➥México dentro de cad0: %s\n", cad0);
/* La función strstr se utiliza para localizar una cadena de caracteres dentro
➥de otra cadena. Si la encuentra, regresa un apuntador al inicio de la
➥primera ocurrencia de la cadena localizada. De otra forma, regresa NULL. */

cad0 = strstr(cad0, "Guatemala");
printf("\nPrueba de la función strstr. Se trata de localizar la cadena
➥Guatemala dentro de cad0: %s\n", cad0);
}
```

En la siguiente tabla se muestran los resultados del programa.

TABLA 7.3. Funciones de la biblioteca `string.h`

Función	cad0	cad1	cad2	cad3
	Hola México			, buenos días!!!
strcpy(cad1, cad0)		Hola México		
strcpy(cad1, cad3)		, buenos días!!!		
strcpy(cad1,"XX")		XX		
strncpy(cad2, cad3, 3)			, b	
strncpy(cad2, cad0, 4)			Hola	
strcat(cad0, cad3)	Hola México, buenos días!!!			
strcat(cad1, " YY")		XX YY		
strcat(cad2, " ")			Hola	
strncat(cad2, cad0, 4)				Hola Hola
cad0 = strstr(cad0, "México")	México, buenos días!!!			
cad0 = strstr	(NULL)			
(cad0, "Guatemala")				

EJEMPLO 7.7

En el siguiente programa podemos observar la aplicación de otras funciones importantes (`strcmp`, `strlen`, `strchr`) para el manejo de cadenas de caracteres de la biblioteca `string.h`.

Programa 7.7

```
#include <stdio.h>
#include <string.h>

/* Otras funciones de la biblioteca string.h para el manejo de cadenas. */

void main(void)
{
int i;
char cad0[20] = "Hola México";
```

```
char cad1[20] = "Hola Guatemala";
char cad2[20] = "Hola Venezuela";
char cad3[20] = "Hola México";
char *c, c3;

i = strcmp(cad0, cad1);
/* La función strcmp permite comparar dos cadenas de caracteres. Si la
➥primera cadena —en este caso cad0— es mayor a la segunda —cad1—,
➥regresa un valor positivo; si es menor, un valor negativo y de otra forma,
➥0. */
printf("\nResultado de la comparación —cad0 y cad1—: %d", i);

i = strcmp(cad0, cad2);
printf("\nResultado de la comparación —cad0 y cad2—: %d", i);

i = strcmp(cad0, cad3);
printf("\nResultado de la comparación —cad0 y cad3—: %d", i);

i = strlen(cad0);
/* La función strlen obtiene la longitud —el número de caracteres— de
➥una cadena. */
printf("\nLongitud cadena cad0: %d", i);

i = strlen(cad1);
printf("\nLongitud cadena cad1: %d", i);

c = strchr(cad1, 'G');          /* c es un apuntador de tipo caracter.
*/
/* La función strchr busca la posición en la que se encuentra un
➥determinado caracter en la cadena de caracteres. Si lo encuentra regresa
➥un apuntador a la primera ocurrencia del caracter en la cadena, de otra
➥forma regresa NULL. */
if (c != NULL)
{
      c3 = *c;       /* c3 toma el contenido de la celda de memoria a la
➥que apunta c.*/
      printf("\nEl valor de c3 es:  %c", c3);
}

c = strchr(cad2, 'V');
if (c != NULL)
{
      c3 = *c;
      printf("\nEl valor de c3 es:  %c", c3);
}
}
```

En la tabla 7.4 se muestran los resultados que arroja el programa.

TABLA 7.4. Otras funciones de la biblioteca `string.h`

Función	`cad0`	`cad1`	`cad2`	`icad3`	`i`	`c3`
	`Hola México`	`Hola Guatemala`	`Hola Venezuela`	`Hola México`		
`i = strcmp(cad0, cad3)`					`6`	
`i =strcmp(cad0, cad3)`					`-9`	
`i = strcmp(cad0, cad3`					`0`	
`i = strlen(cad0)`					`11`	
`i = strlen(cad0)`					`14`	
`c = strchr(cad1, 'G')` `c3 = *c`						`G`
`c = strchr(cad1, 'V')` `c3 = *c`						`V`

7.4. Cadenas de caracteres y arreglos

Una **cadena de caracteres** es un tipo de datos estructurado compuesto por caracteres. En el lenguaje de programación **C** una cadena se define como un arreglo de caracteres que termina con el caracter nulo (`'\0'`). Por otra parte, un **arreglo unidimensional** se define como una colección finita, homogénea y ordenada de datos, en la que se hace referencia a cada elemento del arreglo por medio de un índice. El índice se utiliza para indicar la columna correspondiente. Finalmente, un **arreglo bidimensional** se define como una colección finita, homogénea y ordenada de datos, en la que se hace referencia a cada elemento del arreglo por medio de dos índices. El primer índice se utiliza para indicar la fila y el segundo para indicar la columna.

Las cadenas de caracteres se pueden almacenar fácilmente en cualquier tipo de arreglo. Utilizando una representación de este tipo se pueden resolver de manera eficiente una gran cantidad de problemas. La única característica importante que debemos considerar es la siguiente: *"Dado que una cadena se define como un arreglo unidimensional, si queremos almacenar cadenas de caracteres en arreglos unidimensionales debemos trabajar de forma similar a como lo hacemos*

con arreglos bidimensionales". En las filas almacenamos las cadenas y en las columnas los caracteres de cada cadena. Es decir, si queremos almacenar un grupo de 10 cadenas de caracteres de 30 caracteres como máximo en el arreglo unidimensional `CAR`, éste lo debemos declarar de la siguiente forma:

```
char CAR[10][30];
```

El primer índice se utiliza para indicar la fila y el segundo para señalar el caracter de la cadena. Si, en cambio, quisiéramos almacenar cadenas de caracteres en arreglos bidimensionales, tendríamos que trabajar de forma similar a como lo hacemos con arreglos tridimensionales. Sin embargo, esta representación es muy poco utilizada. Observemos a continuación el siguiente ejemplo:

EJEMPLO 7.8

Escribe un programa en C que, al recibir como dato un arreglo unidimensional de tipo cadena de caracteres, determine el número de minúsculas y mayúsculas que hay en cada cadena.

Dato: `FRA[N][M]` (donde `FRA` representa el arreglo unidimensional de cadena de caracteres, `1 ≤ N ≤ 20, 1 ≤ M ≤ 50`).

Programa 7.8

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/* Minúsculas y mayúsculas.
El programa, al recibir como dato un arreglo unidimensional de tipo
➥cadena de caracteres, determina el número de minúsculas y mayúsculas
➥que hay en cada cadena. */

void minymay(char cad);          /* Prototipo de función. */

void main(void)
{
int i, n;
char FRA[20][50];
/* Observa cómo se declara el arreglo unidimensional de cadena de
➥caracteres. */

printf("\nIngrese el número de filas del arreglo: ");
scanf("%d", &n);
```

```
for (i=0; i<n; i++)
{
      /* Para cada fila se lee la cadena correspondiente. */
      printf("Ingrese la línea %d de texto: ", i+1);
      fflush(stdin);
      gets(FRA[i]);
}
printf("\n\n");
for (i=0; i<n; i++)
   minymay(FRA[i]);
}

void minymay(char *cadena)
/* Esta función se utiliza para calcular el número de minúsculas
➥y mayúsculas que hay en cada cadena. */
{
int i = 0, mi = 0, ma = 0;
while(cadena[i] != '\0')
{
     if (islower(cadena[i]))
          mi++;
     else
        if (isupper(cadena[i]))
             ma++;
     i++;
}
printf("\n\nNúmero de letras minúsculas: %d", mi);
printf("\nNúmero de letras mayúsculas: %d", ma);
}
```

Problemas resueltos

Problema PR7.1

Escribe un programa en **C** que, al recibir como datos una cadena de caracteres y un caracter, determine cuántas veces se encuentra el caracter en la cadena.

Datos: `cad[50]`, `car` (donde `cad` representa una cadena de 50 caracteres como máximo y `car` el caracter).

Programa 7.9

```
#include <stdio.h>

/* Cuenta caracteres.
El programa, al recibir como datos una cadena de caracteres y un caracter,
➥cuenta cuántas veces se encuentra el caracter en la cadena. */

int cuenta(char *, char);                /* Prototipo de función. */

void main(void)
{
char car, cad[50];
int res;
printf("\nIngrese la cadena de caracteres: ");
gets(cad);
fflush(stdin);
printf("\nIngrese el caracter: ");
car = getchar();
res = cuenta(cad, car);
printf("\n\n%c se encuentra %d veces en la cadena %s", car, res, cad);
}

int cuenta(char *cad, char car)
/* Esta función se utiliza para obtener el número de veces que se encuentra
➥el caracter en la cadena. */
{
int i = 0, r = 0;
while (cad[i] != '\0')
{
    if (cad[i] == car)
         r++;
    i++;
}
  return (r);
 }
```

Problema PR7.2

Escribe un programa en **C** que, al recibir como datos cadenas de caracteres que contienen reales, obtenga la suma y el promedio de dichos números.

Datos: $cad_1[10], cad_2[10], cad_3[10], \ldots, s$

Donde: $cad_i[10]$ representa la cadena `i` de `10` caracteres como máximo.

Nota: Observa que antes de leer cada cadena se le pregunta al usuario si desea ingresarla. Si su respuesta es afirmativa —S— entonces se lee, de lo contrario ya no se ingresan más cadenas de caracteres.

Programa 7.10

```
#include <stdio.h>
#include <stdlib.h>

/* Suma y promedio.
El programa, al recibir como datos varias cadenas de caracteres que
➥contienen reales, los suma y obtiene el promedio de los mismos. */

void main(void)
{
char c, cad[10];
int i = 0;
float sum = 0.0;
printf("\nDesea ingresar una cadena de caracteres (S/N)? ");
c = getchar();
while (c == 'S')
{
   printf("\nIngrese la cadena de caracteres: ");
   fflush(stdin);
   gets(cad);
   i++,
   sum += atof(cad);
   printf("\nDesea ingresar otra cadena de caracteres (S/N)? ");
   c = getchar();
}
printf("\nSuma: %.2f", sum);
printf("\nPromedio: %.2f", sum / i);
 }
```

Problema PR7.3

Escribe un programa en **C** que, al recibir como datos una cadena de caracteres y una posición de la cadena, determine si el caracter correspondiente a la posición dada es una letra minúscula.

Datos: `cad[50]`, `n` (donde `cad` representa una cadena de 50 caracteres y `n` una variable de tipo entero que representa la posición en la cadena).

Programa 7.11

```
# include <stdio.h>
# include <ctype.h>

/* Verifica.
El programa, al recibir como datos una cadena de caracteres y una posición es-
pecífica en la cadena, determina si el caracter correspondiente es una letra
minúscula. */

void main(void)
{
char p, cad[50];
int n;
printf("\nIngrese la cadena de caracteres (máximo 50): ");
gets(cad);
printf("\nIngrese la posición en la cadena que desea verificar: ");
scanf("%d", &n);
if ((n >= 0) && (n < 50))
{
        p = cad[n-1];
        if (islower(p))
          printf("\n%c es una letra minúscula", p);
        else
         printf("\n%c no es una letra minúscula", p);
}
else
        printf("\nEl valor ingresado de n es incorrecto");
}
```

Problema PR7.4

Escribe un programa en **C** que determine el número de letras minúsculas y mayúsculas que existen en una frase.

Dato: `cad[50]` (donde `cad` representa la cadena —frase— de 50 caracteres).

Programa 7.12

```
#include <stdio.h>
#include <ctype.h>

/* Cuenta letras minúsculas y mayúsculas.
El programa, al recibir como dato una frase, determina el número de letras
➥minúsculas y mayúsculas que existen en la frase. */

void main(void)
{
char cad[50];
int i = 0, mi = 0, ma = 0;
```

```
printf("\nIngrese la cadena de caracteres (máximo 50 caracteres): ");
gets(cad);
while(cad[i] != '\0')
{
   if (islower (cad[i]))
        mi++;
   else
      if (isupper (cad[i]))
           ma++;
   i++;
}
printf("\n\nNúmero de letras minúsculas: %d", mi);
printf("\nNúmero de letras mayúsculas: %d", ma);
}
```

Problema PR7.5

Escriba un programa en **C** que, al recibir como dato una cadena de caracteres, determine la longitud de la misma sin utilizar la función **`strlen`**.

Dato: `cad[50]` (donde `cad` representa la cadena de 50 caracteres).

Programa 7.13

```
#include <stdio.h>

/* Calcula longitud.
El programa calcula la longitud de la cadena sin utilizar la función strlen. */

int cuenta(char *);             /* Prototipo de función. */

void main(void)
{
int i;
char cad[50];
printf("\nIngrese la cadena de caracteres: ");
gets(cad);
i = cuenta(cad);
printf("\nLongitud de la cadena: %d", i);
}

int cuenta(char *cadena)
/* La función calcula la longitud de la cadena. */
{
int c = 0;
while (!cadena[c] == '\0')
   c++;
return (c);
}
```

La recursividad constituye una alternativa para resolver este problema. A continuación se muestra la solución sugerida.

Programa 7.14

```
#include <stdio.h>

/* Calcula longitud en forma recursiva.
El programa calcula de manera recursiva la longitud de la cadena sin utilizar
➥la función strlen. */

int cuenta(char *);                /* Prototipo de función. */

void main(void)
{
int i;
char cad[50];
printf("\nIngrese la cadena de caracteres: ");
gets(cad);
i = cuenta(cad);
printf("\nLongitud de la cadena: %d", i);
 }

int cuenta(char *cadena)
/* Esta función calcula la longitud de la cadena en forma recursiva. Es
➥importante tener conocimientos tanto de pilas como de recursividad para
➥comprender la solución propuesta, aunque ésta sea muy simple. Observa que
➥mientras no lleguemos al último caracter de la cadena, incrementamos la
➥cuenta en uno y llamamos a la función con el siguiente caracter.  */
{
if (cadena[0] == '\0')
    return 0;
else
    return (1 + cuenta (&cadena[1]));
}
```

Problema PR7.6

Escribe un programa en **C** que, al recibir como dato una cadena de caracteres formada por números y letras, en ese orden, imprima en forma sucesiva cada letra tantas veces como lo indique el número que le precede. Por ejemplo, si la cadena es la siguiente:

```
3p6c4a5q
```

El programa debe imprimir:

```
pppccccccaaaaqqqqq
```

Restricción: Los números están formados por un solo dígito (0...9).

Dato: `cad[50]` (donde `cad` representa la cadena de 50 caracteres).

Programa 7.15

```
# include <stdio.h>
# include <ctype.h>

/* Decodifica.
El programa decodifica una cadena de caracteres compuesta por números y
➥letras. */

void interpreta(char *);                    /* Prototipo de función. */

void main(void)
{
char cad[50];
printf("\nIngrese la cadena de caracteres: ");
gets(cad);
interpreta(cad);
}

void interpreta(char *cadena)
/* Esta función se utiliza para decodificar la cadena de caracteres. */
{
int i = 0, j, k;
while (cad[i] != '\0')
{
   if (isalpha (cad[i]))   /* Se utiliza isalpha para observar si el caracter
                           ➥es una letra. */
    {
         k = cad[i - 1] - 48;
         /* En la variable entera k se almacena el ascii del número —convertido
         ➥en caracter— que nos interesa, menos 48 que corresponde al ascii
         ➥del dígito 0. */
         for (j = 0; j < k; j++)
              putchar(cad[i]);
    }
    i++;
}
```

Problema PR7.7

Escribe un programa en **C** que, al recibir como datos dos cadenas de caracteres, determine cuántas veces se encuentra la segunda cadena en la primera. Por ejemplo, si la primera cadena es la siguiente:

```
sasaasassassssassas
```

y la segunda cadena es:

```
sas
```

el programa debe regresar: `5`

Datos: `cad0[50], cad1[50]` (donde `cad0` y `cad1` representan las cadenas de 50 caracteres como máximo)

Programa 7.16

```
#include <stdio.h>
#include <string.h>

/* Cuenta cadenas.
El programa, al recibir dos cadenas de caracteres, calcula e imprime cuántas
➥veces se encuentra la segunda cadena en la primera. */

void main(void)
{
char cad1[50], cad2[50], *cad0 = "";
int i = 0;
printf("\n Ingrese la primera cadena de caracteres: ");
gets(cad1);
printf("\n Ingrese la cadena a buscar: ");
gets(cad2);
strcpy(cad0, cad1);             /* Se copia la cadena original a cad0. */
cad0 = strstr (cad0, cad2);
/* En cad0 se asigna el apuntador a la primera ocurrencia de la cadena cad2.
➥Si no existe se almacena NULL.*/
while (cad0 != NULL)
{
    i++;
    cad0 = strstr (cad0 + 1, cad2);
   /* Se modifica nuevamente la cadena, moviendo el apuntador una
➥posición. */
}
printf("\nEl número de veces que aparece la segunda cadena es: %d", i);
}
```

Problema PR7.8

Escribe un programa que, al recibir como dato una línea de texto —cadena de caracteres—, escriba esa línea en forma inversa. Por ejemplo, si la línea de texto dice:

```
Hola México
```

El programa debe escribir:

```
ocixéM aloH
```

Dato: `fra[50]` (donde `fra` representa la cadena de 50 caracteres como máximo)

Programa 7.17

```
#include <stdio.h>
#include <string.h>

/* Cadena invertida.
El programa obtiene la cadena invertida. */

char * inverso(char *);                        /* Prototipo de función. */

void main(void)
{
char fra[50], aux[50];
printf("\nIngrese la línea de texto: ");
gets(fra);
strcpy(aux, inverso(fra));    /* Se copia a aux el resultado de la función
➥inverso. */
printf("\nEscribe la línea de texto en forma inversa: ");
puts(aux);
}

char * inverso(char *cadena)
/* La función calcula el inverso de una cadena y regresa el resultado al
➥programa principal. */
{
int i = 0, j, lon;
char cad;
lon = strlen(cadena);
j = lon-1;
while (i < ((lon - 1) / 2))
/* Observa que el reemplazo de los caracteres se debe realizar solamente
➥hasta la mitad de la cadena. */
{
   cad = cadena[i];
   cadena[i] = cadena[j];
   cadena[j] = cad;
   i++;
   j--;
}
return (cadena);
}
```

En el siguiente programa se presenta otra forma de resolver el problema, pero ahora de manera recursiva.

Programa 7.18

```
#include <stdio.h>

/* Cadena invertida resuelta en forma recursiva. */

void inverso(char *);             /* Prototipo de función. */

void main(void)
{
char fra[50];
printf("\nIngrese la línea de texto: ");
gets(fra);
printf("\nEscribe la línea de texto en forma inversa: ");
inverso(fra);
}

void inverso(char *cadena)
/* La función inverso obtiene precisamente el inverso de la cadena. La solución
➥presentada es simple, pero para comprenderla es necesario tener conocimientos
➥tanto de pilas como de recursividad. Observa que mientras no se encuentre el
➥caracter de terminación de la cadena, se llama a la función recursiva con
➥el apuntador al siguiente caracter de la cadena. Por otra parte, queda
➥pendiente de ejecutar —almacenado en una pila— el caracter al cual apunta
➥*cadena. */
{
if (cadena[0]!= '\0')
{
     inverso(&cadena[1]);
     putchar(cadena[0]);
}
}
```

Problema PR7.9

Escribe un programa en **C** que, al recibir como dato una cadena de caracteres, determine cuántas palabras se encuentran en dicha cadena. Cada palabra se separa por medio de un espacio en blanco. Por ejemplo, si la cadena es la siguiente:

```
México es la novena economía del mundo
```

El programa debe escribir que hay siete palabras.

Dato: `fra[50]` (donde `fra` representa la cadena de 50 caracteres como máximo).

Programa 7.19

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

/* Cuenta palabras.
El programa calcula el número de palabras que hay en la cadena de caracteres. */

int cuentap(char *);                    /* Prototipo de función. */

void main(void)
{
int i;
char fra[50];
printf("\nIngrese la línea de texto: ");
gets(fra);
strcat(fra," ");              /* Se agrega un espacio en blanco al final de la
                              ➥cadena. */
i = cuentap(fra);
printf("\nLa línea de texto tiene %d palabras", i);
}

int cuentap(char *cad)
{
/* La función cuenta el número de palabras que hay en la cadena de
➥caracteres. */
char *cad0 = "";
int i = 0;
cad0 = strstr(cad," ");       /* Se localiza el primer espacio en blanco en la
                              ➥cadena. */
while (strcmp(cad, " "))
{
    strcpy(cad, cad0);
    i++;
    cad0 = strstr (cad + 1," ");
    /* Se busca un espacio en blanco a partir de la siguiente posición. */
}
return (i);
}
```

Problema PR7.10

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional de tipo cadena de caracteres, encuentre la cadena de mayor longitud sin utilizar la función `strlen`, y que imprima tanto la cadena como el número de caracteres de la misma.

Dato: `FRA[n][m]` (donde `FRA` representa el arreglo unidimensional de cadena de caracteres, `1 ≤ n ≤ 20, 1 ≤ m ≤ 50`).

Programa 7.20

```
#include <stdio.h>
#include <string.h>

int longitud(char cad);         /*  Prototipo de función. */

void main(void)
{
int i, n, l = -1, p, t;
char cad[50], FRA[20][50];
printf("\nIngrese el número de filas del arreglo: ");
scanf("%d", &n);
for (i=0; i<n; i++)
{
    printf("Ingrese la línea %d de texto. Máximo 50 caracteres: ", i+1);
    fflush(stdin);
    gets(FRA[i]);      /* Se lee la cadena de caracteres dentro del ciclo. */
}
printf("\n");
for (i=0; i<n; i++)
{
    strcpy(cad, FRA[I]);
    t = longitud (cad);
    if (t > l)
    {
          l = t;
          p = i;
    }
}
printf("\nLa cadena con mayor longitud es: ");
puts(FRA[p]);
printf("\nLongitud: %d", l);
}

int longitud(char *cadena)
/* Esta función calcula la longitud de la cadena. Es idéntica a la función
➥cuenta del programa 7.13. */
{
int cue = 0;
while (! cadena[cue] ==  '\0')
   cue++;
return (cue);
}
```

Problema PR7.11

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional de tipo cadena de caracteres, intercambie las filas del arreglo: *la última con la primera, la penúltima con la segunda, y así sucesivamente*.

Dato: `FRA[n][m]` (donde `FRA` representa el arreglo unidimensional de cadena de caracteres, $1 \leq n \leq 20$, $1 \leq m \leq 30$).

Programa 7.21

```
#include <stdio.h>
#include <string.h>

void intercambia(char FRA[][30], int);      /* Prototipo de función. */

void main(void)
{
int i, n;
char FRA[20][30];
printf("\nIngrese el número de filas del arreglo: ");
scanf("%d", &n);
for (i=0; i<n; i++)
{
     printf("Ingrese la línea de texto número %d: ", i+1);
     fflush(stdin);
     gets(FRA[i]);
}
printf("\n\n");
intercambia(FRA, n);
for (i=0; i<n; i++)
{
     printf("Impresión de la línea de texto %d: ", i+1);
     puts(FRA[i]);
}
}

void intercambia(char FRA[][30], int n)
/* Esta función intercambia las filas del arreglo. */
{
int i, j;
j = n - 1;
char cad[30];
for (i=0; i < (n/2); i++)
{
     strcpy(cad, FRA[i]);
     strcpy(FRA[i], FRA[j]);
     strcpy(FRA[j], cad);
     j--
}
}
```

Problemas suplementarios

Problema PS7.1

Escribe un programa en **C** que, al recibir como dato una cadena de caracteres, imprima todos los caracteres impares de la cadena.

Dato: `cad[50]` (donde `cad` representa la cadena de 50 caracteres como máximo).

Problema PS7.2

Desarrolla un programa en **C** que, al recibir como dato una cadena de caracteres, escriba solamente los dígitos que se encuentren en las posiciones pares.

Dato: `cad[50]` (donde `cad` representa la cadena de 50 caracteres como máximo).

Problema PS7.3

Escribe un programa en **C** que, al recibir como dato una cadena de caracteres cuya longitud máxima sea 30, complete dicha cadena con el caracter - si la cadena no alcanza el máximo correspondiente. Por ejemplo, si recibe la cadena:

Estructuras de Datos

el programa debería modificar e imprimir la cadena:

Estructuras de Datos ——————

Dato: `cad[30]` (donde `cad` representa la cadena de 30 caracteres como máximo).

Problema PS7.4

Construye un programa que, al recibir como dato una cadena de caracteres que exprese una fecha en formato (dd/mm/aa), genere otra cadena con la misma fecha pero con formato (dd de *nombre del mes* de aaaa). Por ejemplo, si la fecha se ingresa de esta forma:

`06/08/05`

la nueva cadena debe indicar lo siguiente:

```
06 de Agosto de 2005
```

Dato: `cad[30]` (donde `cad` representa la cadena de 30 caracteres como máximo).

Problema PS7.5

Escribe un programa que, al recibir como dato una cadena de caracteres, convierta el primer caracter de cada palabra si ésta fuera una letra, de minúscula a mayúscula. Por ejemplo, si la cadena es la siguiente:

```
Estructuras de datos, año 2003, edición 2
```

el programa debe imprimir:

```
Estructuras De Datos, Año 2003, Edición 2
```

Dato: `cad[50]` (donde `cad` representa la cadena de 50 caracteres).

Problema PS7.6

Escribe un programa en **C** que, al recibir como datos cadenas de caracteres, determine cuál es la de mayor longitud.

Datos: $cad_1[20]$, $cad_2[20]$, $cad_3[20]$, ..., S

Donde: cad_i representa la cadena `i` de 20 caracteres como máximo.

Nota: Observa que antes de leer cada cadena se le pregunta al usuario si desea ingresarla. Si su respuesta es afirmativa —S—, entonces se lee, de lo contrario ya no se ingresan más cadenas de caracteres.

Problema PS7.7

Desarrolla un programa en **C** que, al recibir como dato un número telefónico en formato de cadena, lo convierta y escriba de la siguiente manera:

```
Número telefónico: 5256284000

Nueva cadena: (52)-5-6284000
```

Dato: `cad[30]` (donde `cad` representa la cadena de caracteres).

Problema PS7.8

Escribe un programa en **C** que, al recibir como datos dos cadenas de caracteres, forme una tercera cadena intercalando las palabras de las cadenas recibidas. Por ejemplo, si las cadenas son las siguientes:

```
aa ab ac af ap ar
   ap bc bd be
```

el programa debe generar una cadena como la siguiente:

```
aa ap ab bc ac bd af be ap ar
```

Datos: `cad1[50], cad2[50]` (donde `cad1` y `cad2` representan las cadenas de 50 caracteres como máximo).

Problema PS7.9

Escribe un programa en **C** que, al recibir como dato una cadena de caracteres, imprima la cadena en forma inversa. Por ejemplo, si la cadena es la siguiente:

```
mundo del economía novena la es México
```

el programa debe imprimirla de esta forma:

```
México es la novena economía del mundo
```

Dato: `cad[50]` (donde `cad` representa la cadena de 50 caracteres como máximo).

Problema PS7.10

Desarrolla un programa en **C** que, al recibir como datos varias cadenas de caracteres, escriba sólo aquellas que tengan al inicio la fecha del día de hoy. Todas las cadenas tienen el siguiente formato:

```
06/08/2005 Cadena
```

Datos: cad_1`[50], `cad_2`[50], `cad_3`[50], ..., S`

Donde: cad_i representa la cadena `i` de 50 caracteres como máximo.

Nota: Observa que antes de leer cada cadena se le pregunta al usuario si desea ingresarla. Si su respuesta es afirmativa —S—, entonces se lee, de lo contrario ya no se ingresan más cadenas de caracteres.

Problema PS7.11

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional de cadenas de caracteres, imprima la cadena que tiene el mayor número de vocales.

Dato: `ARC[10][50]` (donde `ARC` representa un arreglo de cadena de caracteres de 10 filas y cada cadena puede tener 50 caracteres como máximo).

Problema PS7.12

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional de cadenas de caracteres, imprima la cadena que tiene el mayor número de letras mayúsculas.

Dato: `ARC[10][50]` (donde `ARC` representa un arreglo de cadena de caracteres de 10 filas y cada cadena puede tener 50 caracteres como máximo).

Problema PS7.13

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional de cadenas de caracteres, imprima el número de palabras que hay en cada cadena.

Dato: `ARC[10][50]` (donde `ARC` representa un arreglo de cadena de caracteres de 10 filas y cada cadena puede tener 50 caracteres como máximo).

Problema PS7.14

Escribe un programa en **C** que, al recibir como dato un arreglo unidimensional de cadenas de caracteres, imprima la frecuencia con que aparecen las palabras en función de la longitud de las mismas. Por ejemplo, si el arreglo almacena las siguientes cadenas de caracteres —tomadas del libro *El amor en los tiempos de cólera*, de Gabriel García Márquez—:

```
Era inevitable, el olor de las almendras amargas le recordaba siempre el
destino de los amores contrariados. El doctor Juvenal Urbino lo percibió
desde que entró en la casa todavía en penumbras, adonde había acudido de
 urgencia a ocuparse de un caso que para él había dejado de ser urgente
                        desde hacía muchos años.
```

El programa debe imprimir lo siguiente:

```
Longitud de la palabra    Frecuencia

      1                        1
      2                       15
      3                        6
      4                        5
      5                        6
      6                        6
      7                        8
      8                        3
      9                        3
      10                       1
      11                       0
      12                       1
```

Dato: `ARC[20][80]` (donde `ARC` representa un arreglo de cadena de caracteres de 20 filas y cada cadena puede tener 80 caracteres como máximo)

CAPÍTULO 8

Estructuras y uniones

8.1. Introducción

Cuando estudiamos arreglos en los capítulos 5 y 6, observamos que representan un tipo de datos estructurado y permiten resolver un gran número de problemas en forma efectiva. Definimos a los arreglos como una colección finita, homogénea y ordenada de elementos. En este capítulo estudiaremos dos tipos de datos estructurados que se distinguen fundamentalmente de los arreglos porque sus elementos pueden ser **heterogéneos**, es decir, pueden pertenecer —aunque no necesariamente— a tipos de datos diferentes. Estas estructuras de datos reciben el nombre de **estructuras** y **uniones**.

8.2. Estructuras

Las **estructuras,** conocidas generalmente con el nombre de **registros**, representan un tipo de datos estructurado. Se utilizan tanto para resolver problemas que involucran tipos de datos estructurados, heterogéneos, como para almacenar información en archivos —como veremos en el siguiente capítulo. Las **estructuras** tienen varios componentes, cada uno de los cuales puede constituir a su vez un tipo de datos simple o estructurado. Sin embargo, los componentes del nivel más bajo de un tipo estructurado, siempre son tipos de datos simples. Formalmente definimos a una estructura de la siguiente manera:

"Una estructura es una colección de elementos finita y heterogénea."

Finita porque se puede determinar el número de componentes y **heterogénea** porque todos los elementos pueden ser de tipos de datos diferentes. Cada componente de la estructura se denomina **campo** y se identifica con un **nombre único**. Los campos de una estructura pueden ser de tipos de datos diferentes como ya hemos mencionado, simples o estructurados; por lo tanto, también podrían ser nuevamente una estructura. Para hacer referencia a un *campo* de una estructura siempre debemos utilizar tanto el *nombre de la variable tipo estructura* como el *nombre del campo*. En la figura 8.1 se muestra la representación gráfica de una estructura.

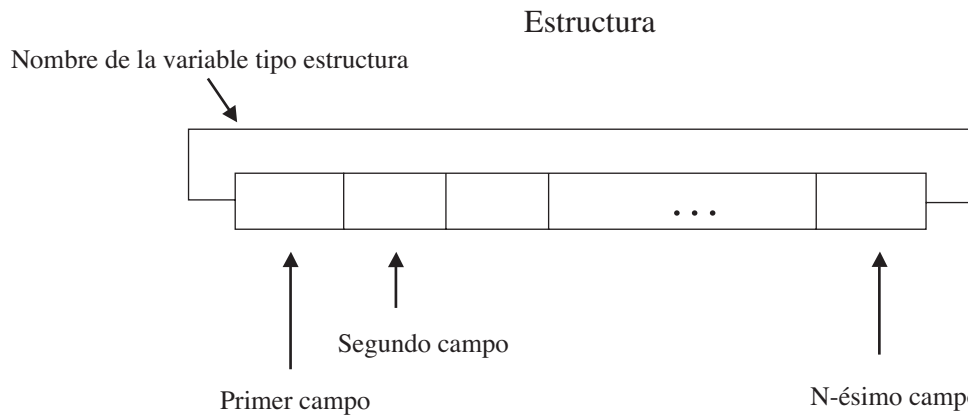


FIGURA 8.1
Representación gráfica de una estructura

EJEMPLO 8.1

Consideremos que por cada alumno de una universidad debemos almacenar la siguiente información:

- Matrícula del alumno (entero).
- Nombre del alumno (cadena de caracteres).
- Carrera del alumno (cadena de caracteres).
- Promedio del alumno (real).
- Dirección del alumno (cadena de caracteres).

La estructura de datos adecuada para almacenar esta información es la **estructura**. Cabe aclarar que no es posible utilizar un arreglo para resolver este problema, porque sus componentes deben ser del mismo tipo de datos. En la figura 8.2 se muestra la representación gráfica de este ejemplo.

Alumno				
Matrícula	Nombre	Carrera	Promedio	Domicilio

FIGURA 8.2
Representación gráfica de la estructura del ejemplo 8.1

El primer campo de la estructura es `Matrícula`; el segundo, `Nombre`; el tercero, `Carrera`, y así sucesivamente. Si queremos acceder entonces al primer campo de la estructura debemos escribir `variable-de-tipo-estructura-Alumno.Matrícula`. Si en cambio queremos hacer referencia al domicilio del alumno escribimos `variable-de-tipo-estructura-Alumno.Domicilio`.

8.2.1. Declaración de estructuras

Observemos a continuación diferentes formas de declarar estructuras con la explicación correspondiente.

EJEMPLO 8.2

En el siguiente programa podemos observar la forma en que se declara la estructura del ejemplo 8.1, así como también diferentes formas en que los campos reciben valores.

Programa 8.1

```
#include <stdio.h>
#include <string.h>

/* Estructuras-1.
El programa muestra la manera en que se declara una estructura, así como la
➥forma en que se tiene acceso a los campos de las variables de tipo estructura
➥tanto para asignación de valores como para lectura y escritura. */

struct alumno                          /* Declaración de la estructura. */
{
   int matricula;
   char nombre[20];
   char carrera[20];                   /* Campos de la estructura. */
   float promedio;
   char direccion[20];
};        /* Observa que la declaración de una estructura termina con punto y
          ➥coma. */

void main(void)
{
/* Observa que las variables de tipo estructura se declaran como cualquier otra
➥variable. a1, a2 y a3 son variables de tipo estructura alumno. */
➥struct alumno a1 = {120, "María", "Contabilidad", 8.9, "Querétaro"}, a2, a3;
/* Los campos de a1 reciben valores directamente. */

char nom[20], car[20], dir[20];
int mat;
float pro;

/* Los campos de a2 reciben valores por medio de una lectura. */
printf("\nIngrese la matrícula del alumno 2: ");
scanf("%d", &a2.matricula);
fflush(stdin);
printf("Ingrese el nombre del alumno 2:");
gets(a2.nombre);
printf("Ingrese la carrera del alumno 2: ");
gets(a2.carrera);
printf("Ingrese el promedio del alumno 2: ");
scanf("%f", &a2.promedio);
fflush(stdin);
printf("Ingrese la dirección del alumno 2: ");
gets(a2.direccion);

/* Los campos de a3 reciben valores por medio de asignaciones. */
printf("\nIngrese la matrícula del alumno 3: ");
scanf("%d", &mat);
a3.matricula = mat;
fflush(stdin);
printf("Ingrese el nombre del alumno 3: ");
gets(nom);
```

```
strcpy(a3.nombre, nom);
printf("Ingrese la carrera del alumno 3: ");
gets(car);
strcpy(a3.carrera, car);
printf("Ingrese el promedio del alumno 3: ");
scanf("%f", &pro);
a3.promedio = pro;
fflush(stdin);
printf("Ingrese la dirección del alumno 3: ");
gets(dir);
strcpy(a3.direccion, dir);

/* Observe la forma en que se imprimen los campos de a1 y a2. */
printf("\nDatos del alumno 1\n");
printf("%d\n", a1.matricula);
puts(a1.nombre);
puts(a1.carrera);
printf("%.2f\n", a1.promedio);
puts(a1.direccion);

printf("\nDatos del alumno 2\n");
printf("%d\n", a2.matricula);
puts(a2.nombre);
puts(a2.carrera);
printf("%.2f\n", a2.promedio);
puts(a2.direccion);

/* Observa otra forma de escribir los campos de la variable de tipo estructura
a3. */
printf("\nDatos del alumno 3\n");
printf("%d \t %s \t %s \t %.2f \t %s", a3.matricula, a3.nombre, a3.carrera,
        a3.promedio, a3.direccion);
}
```

EJEMPLO 8.3

En el siguiente programa podemos observar diferentes formas en que los campos de las variables declaradas como **apuntadores de una estructura** reciben valores, así como también el acceso a los campos de estas variables.

Programa 8.2

```
#include <string.h>

/* Estructuras-2.
El programa muestra la manera en que se declara una estructura, así como la
➥forma en que se tiene acceso a los campos de los apuntadores de tipo estructura
➥tanto para lectura como para escritura. Se utiliza además una función que
➥recibe como parámetro un apuntador de tipo estructura. */
struct alumno                          /* Declaración de la estructura. */
{
```

```
   int matricula;
   char nombre[20];
   char carrera[20];                    /* Campos de la estructura alumno. */
   float promedio;
   char direccion[20];
};

void Lectura(struct alumno *);          /* Prototipo de función. */

void main(void)
{
struct alumno a0 = {120, "María", "Contabilidad", 8.9, "Querétaro"};
struct alumno *a3, *a4, *a5, a6;
/* Observa que las variables *a3, *a4 y *a5 se declaran como apuntadores de
➥tipo estructura alumno. a6 es una variable de tipo estructura alumno. */

a3 = &a0;          /* En este caso al apuntador de tipo estructura alumno a3
➥se le asigna la dirección de la variable de tipo estructura alumno, a0. */

a4 = new (struct alumno);
/* Nota que al apuntador a4 es necesario asignarle una dirección de memoria.
➥Para tener acceso a los campos de un apuntador de tipo estructura, utiliza uno
➥de los dos formatos siguientes:

                                apuntador->campo
                                     o bien
                               (*apuntador).campo

En la lectura de los campos de la variable a4 se utilizan como ejemplo ambos
➥formatos. */
printf("\nIngrese la matrícula del alumno 4: ");
scanf("%d", &(*a4).matricula);
fflush(stdin);
printf("Ingrese el nombre del alumno 4: ");
gets(a4->nombre);
printf("Ingrese la carrera del alumno 4: ");
gets((*a4).carrera);
printf("Ingrese promedio del alumno 4: ");
scanf("%f", &a4->promedio);
fflush(stdin);
printf("Ingrese la dirección del alumno 4: ");
gets(a4->direccion);

a5 = new (struct alumno);
Lectura(a5);     /* En este caso se pasa el apuntador de tipo estructura alumno
a5 a la función Lectura. */

Lectura(&a6);  /* En este caso se pasa la variable de tipo estructura alumno a6,
➥a la función Lectura. Observa que en este caso debemos utilizar el operador de
➥dirección para preceder a la variable. */
printf("\nDatos del alumno 3\n");
/* Observa la forma de escribir los campos de los apuntadores de tipo
➥estructura. */
```

```
printf("%d\t%s\t%s\t%.2f\t%s", a3->matricula, a3->nombre, a3->carrera,
     ➥a3->promedio, a3->direccion);

printf("\nDatos del alumno 4\n");
printf("%d\t%s\t%s\t%.2f\t%s", a4->matricula, a4->nombre, a4->carrera,
     ➥a4->promedio, a4->direccion);

printf("\nDatos del alumno 5\n");
printf("%d\t%s\t%s\t%f\t%s", a5->matricula, a5->nombre, a5->carrera,
     ➥a5->promedio, a5->direccion);

printf("\nDatos del alumno 6\n");
/* Observa la forma de escribir los campos de la variable tipo estructura. */
printf("%d\t%s\t%s\t%.2f\t%s", a6.matricula, a6.nombre, a6.carrera,
     ➥a6.promedio, a6.direccion);
}

void Lectura(struct alumno *a)
/* Esta función permite leer los campos de un apuntador de tipo estructura
➥alumno. */
{
printf("\nIngrese la matrícula del alumno: ");
scanf("%d", &(*a).matricula);
fflush(stdin);
printf("Ingrese el nombre del alumno: ");
gets(a->nombre);
fflush(stdin);
printf("Ingrese la carrera del alumno: ");
gets((*a).carrera);
printf("Ingrese el promedio del alumno: ");
scanf("%f", &a->promedio);
fflush(stdin);
printf("Ingrese la dirección del alumno: ");
gets(a->direccion);
}
```

8.2.2. Creación de sinónimos o alias

La instrucción `typedef` permite al usuario definir **alias** o **sinónimos**, es decir, nuevos tipos de datos equivalentes a los ya existentes. El objetivo de esta instrucción consiste en utilizar nombres más apropiados y más cortos para los tipos de datos, puesto que evitamos escribir la palabra struct en la declaración de las variables.

La instrucción `typedef` se puede utilizar tanto con tipos de datos simples como con estructurados. Con los tipos de datos simples su uso no resulta muy práctico, más bien es redundante. Por ejemplo, si tenemos que declarar cinco variables de tipo entero, `C1, C2, C3, C4` y `C5`, para un problema en particular, lo hacemos de esta forma:

```
    . . .
    int C1, C2, C3, C4, C5;
    . . .
```

Si en cambio utilizáramos la instrucción `typedef`, tendríamos que escribir las siguientes instrucciones:

```
. . .
typedef int contador;        /* Se declara un tipo de datos definido por el
                             ➥usuario,
                             contador en este caso, equivalente al tipo de dato
                             ➥int. */
   . . .

   void main(void)
   . . .
   contador C1, C2, C3, C4, C5;
   /* Posteriormente, ya sea en el programa principal o en una función,
   ➥declaramos las variables C1, C2, C3, C4 y C5 como de tipo contador. */
   . . .
```

En los tipos de datos estructurados, específicamente en las *estructuras*, su uso es importante ya que elimina la necesidad de escribir reiteradamente la palabra `struct` cada vez que hacemos referencia a una variable o apuntador de tipo estructura. Observemos a continuación la modificación que realizamos al programa 8.2.

```
. . .
typedef struct            /* Declaración de la estructura utilizando typedef.*/
{
   int matricula;
   char nombre[20];
   char carrera[20];
   float promedio;
   char direccion[20];
} alumno;                 /* alumno es el nuevo tipo de datos creado por el
                          ➥usuario. */

void Lectura (alumno *);  /* Prototipo de función. Observa que al haber creado
➥el tipo de datos definido por el usuario alumno, se elimina la necesidad de
➥escribir la palabra struct antes de alumno en el parámetro. */

void main(void)
{
alumno a0 = {120, "María", "Contabilidad", 8.9, "Querétaro"}, *a3, *a4, *a5, a6;
/* En este caso se evita escribir la palabra struct en la declaración de las
➥variables tipo alumno. */
. . .
```

8.2.3. Estructuras anidadas

Las **estructuras** representan un tipo de datos estructurado, que tiene por lo tanto varios componentes. Cada uno de estos componentes puede a su vez ser un tipo de datos simple o estructurado. Las **estructuras anidadas** se presentan cuando en la declaración de una estructura, por lo menos uno de sus componentes es una estructura. Observemos el siguiente ejemplo.

EJEMPLO 8.4

Consideremos que en una empresa requieren almacenar la siguiente información de cada empleado:

- Nombre del empleado (cadena de caracteres).
- Departamento de la empresa (cadena de caracteres).
- Sueldo (real).
- Domicilio
  - Calle (cadena de caracteres).
  - Número (entero).
  - Código Postal (entero).
  - Localidad (cadena de caracteres).

A continuación **se observa** la representación gráfica de esta estructura:

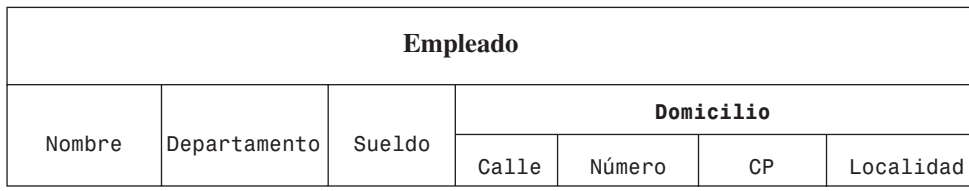


FIGURA 8.3
Representación gráfica de una estructura anidada

El programa muestra la manera como se declara una estructura anidada, así como la forma de acceso a los campos de cada una de las variables o apuntadores de tipo estructura, tanto para lectura como para escritura. Observa que para la lectura de los datos de algunas variables y apuntadores de tipo estructura se utiliza una función.

Programa 8.3

```
#include <stdio.h>
#include <string.h>

/* Estructuras-3.
El programa muestra la manera en que se declara una estructura anidada, así
➥como la forma de acceso a los campos de las variables o apuntadores de tipo
➥estructura, tanto para lectura como para escritura. Se utiliza además una
➥función que recibe como parámetro un apuntador de tipo estructura. */

typedef struct        /* Declaración de la estructura domicilio utilizando
➥un typedef. */
{
   char calle[20];
   int numero;
   int cp;
   char localidad[20];
} domicilio;

struct empleado    /* Declaración de la estructura anidada empleado. */

{
   char nombre[20];
   char departamento[20];
   float sueldo;
       domicilio direccion;        /* direccion es un campo de tipo estructura
                                   ➥domicilio de la estructura empleado. */
};

void Lectura(struct empleado *a)
/* Función que permite leer los campos de un apuntador de tipo estructura
➥empleado. */
{
printf("\nIngrese el nombre del empleado: ");
gets(a->nombre);
fflush(stdin);
printf("Ingrese el departamento de la empresa: ");
gets(a->departamento);
printf("Ingrese el sueldo del empleado: ");
scanf("%f", &a->sueldo);
fflush(stdin);
printf("—-Ingrese la dirección del empleado—-");
printf("\n\tCalle: ");
gets(a->direccion.calle);
printf("\tNúmero: ");
scanf("%d", &a->direccion.numero);
printf("\tCódigo Postal: ");
scanf("%d", &a->direccion.cp);
fflush(stdin);
printf("\tLocalidad: ");
```

```
gets(a->direccion.localidad);
}

void main(void)
{
struct empleado e0 = {"Arturo", "Compras", 15500.75, "San Jerónimo", 120,
     ➥3490, "Toluca"};
struct empleado *e1, *e2, e3, e4;
/* Se declaran diferentes variables y apuntadores de la estructura empleado
➥para que el lector pueda apreciar también las diferentes formas en que los
➥campos reciben valores. */

/* En el programa principal se leen los campos de una variable, e3, y un
➥apuntador de tipo estructura, *e1. */
e1 = new (struct empleado);
printf("\nIngrese el nombre del empleado 1: ");
scanf("%s", &(*e1).nombre);
fflush(stdin);
printf("Ingrese el departamento de la empresa: ");
gets(e1->departamento);
printf("Ingrese el sueldo del empleado: ");
scanf("%f", &e1->sueldo);
printf("—-Ingrese la dirección del empleado—-");
printf("\n\tCalle: ");
fflush(stdin);
gets(e1->dirección, calle);
printf("\tNúmero: ");
scanf("%d", &e1->direccion.numero);
printf("\tCódigo Postal: ");
scanf("%d", &e1->direccion.cp);
printf("\tLocalidad: ");
fflush(stdin);
gets(e1->direccion.localidad);

printf("\nIngrese el nombre del empleado 3: ");
scanf("%s", &e3.nombre);
fflush(stdin);
printf("Ingrese el departamento de la empresa: ");
gets(e3.departamento);
printf("Ingrese el sueldo del empleado: ");
scanf("%f", &e3.sueldo);
printf("—-Ingrese la dirección del empleado—-");
printf("\n\tCalle: ");
fflush(stdin);
gets(e3.direccion.calle);
printf("\tNúmero: ");
scanf("%d", &e3.direccion.numero);
printf("\tCódigo Postal: ");
scanf("%d", &e3.direccion.cp);
printf("\tLocalidad: ");
fflush(stdin);
gets(e3.direccion.localidad);
```

```
/* En la función Lectura se leen los campos de una variable, e4, y un apuntador
➥de tipo estructura, *e2. */
e2 = new (struct empleado);
Lectura(e2);

Lectura(&e4);

printf("\nDatos del empleado 1\n");
printf("%s\t%s\t%.2f\t%s\t%d\t%d\t%s",       e1->nombre,      e1->departamento,
➥e1>sueldo,
e1->direccion.calle,        e1->direccion.numero,        e1->direccion.cp,
➥e1->direccion.localidad);

printf("\nDatos del empleado 4n");
printf("%s\t%s\t%.2f\t%s\t%d\t%d\t%s", e4.nombre, e4.departamento, e4.sueldo,
➥e4.direccion.calle, e4.direccion.numero, e4.direccion.cp, e4.direccion.localidad);
}
```

8.2.4. Estructuras con arreglos

Existen numerosos casos en la vida real en los que para resolver un problema de manera eficiente necesitamos utilizar **estructuras combinadas con arreglos**. Observemos el siguiente ejemplo, en el que uno de los campos de la estructura es a su vez otro arreglo.

EJEMPLO 8.5

En una escuela almacenan la información de sus alumnos utilizando arreglos unidimensionales. La siguiente información de cada alumno se guarda en una estructura:

- Matrícula (entero).
- Nombre y apellido (cadena de caracteres).
- Promedios de las materias (arreglo unidimensional de reales).

Dato: `ARRE[N]` (donde `ARRE` es un arreglo unidimensional de tipo `ALUMNO`, $1 \leq N \leq 100$).

Escribe un programa en **C** que obtenga lo siguiente:

a) La matrícula y el promedio de cada alumno.
b) Las matrículas de los alumnos cuya calificación en la tercera materia sea mayor a 9.
c) El promedio general de la materia 4.

Programa 8.4

```
#include <stdio.h>
#include <string.h>

/* Escuela.
El programa genera información estadística de los alumnos de una escuela. */

typedef struct            /* Declaración de la estructura alumno utilizando un
                          ➥typedef. */
{
  int matricula;
  char nombre[30];
  float cal[5];
  /* Observa que el campo de la estructura alumno es un arreglo
  ➥unidimensional. */
} alumno;

void Lectura(alumno, int T);
void F1(alumno *, int TAM);               /* Prototipos de funciones. */
void F2(alumno *, int TAM);
void F3(alumno *, int TAM);

void main(void)
{
alumno ARRE[50];     /* Se declara un arreglo unidimensional de tipo alumno. */
int TAM;
do
{
  printf("Ingrese el tamaño del arreglo: ");
  scanf("%d", &TAM);
}
while (TAM > 50 ¦¦ TAM < 1);  /* Se verifica que el tamaño del arreglo sea
                              ➥correcto. */
Lectura(ARRE, TAM);
F1(ARRE, TAM);
F2(ARRE, TAM);
F3(ARRE, TAM);
}

void Lectura(alumno A[], int T)
/* La función Lectura se utiliza para leer un arreglo unidimensional de tipo
➥estructura alumno de T elementos. */
{
int I, J;
for (I=0; I<T; I++)
{
     printf("\nIngrese los datos del alumno %d", I+1);
     printf("\nIngrese la matrícula del alumno: ");
     scanf("%d", &A[I].matricula);
```

```
        fflush(stdin);
        printf("Ingrese el nombre del alumno:");
        gets(A[I].nombre);
        for (J=0; J<5; J++)
        {
            printf("\tIngrese la calificación %d  del alumno %d: ", J+1, I+1);
            scanf("%f", &A[I].cal[J]);
        }
    }
    }

    void F1(alumno A[], int T)
    /* La función F1 obtiene la matrícula y el promedio de cada alumno. */
    {
    int I, J;
    float SUM, PRO;
    for (I=0; I<T; I++)
    {
        printf("\nMatrícula del alumno:  %d", A[I].matricula);
        SUM = 0.0;
        for (J=0; J<5; J++)
             SUM = SUM + A[I].cal[J];
        PRO = SUM / 5;
        printf("\t\tPromedio: %.2f", PRO);
    }
    }

    void F2(alumno A[], int T)
    /* La función F2 obtiene las matrículas de los alumnos cuya calificación en la
    ➥tercera materia es mayor a 9. */
    {
    int I;
    printf("\nAlumnos con calificación en la tercera materia > 9");
    for (I=0; I<T; I++)
        if (A[I].cal[2] > 9)
            printf("\nMatrícula del alumno: %d", A[I].matricula);
    }

    void F3(alumno A[], int T)
    /* Esta función obtiene el promedio general del grupo de la materia 4. */
    {
    int I;
    float PRO, SUM = 0.0;
    for (I=0; I<T; I++)
        SUM = SUM + A[I].cal[3];
    PRO = SUM / T;
    printf("\n\nPromedio de la materia 4: %.2f", PRO);
    }
```

8.3. Uniones

Las **uniones** representan también un tipo de datos estructurado. Son similares a las **estructuras**. Sin embargo, se distinguen fundamentalmente de éstas porque sus miembros comparten el mismo espacio de almacenamiento en la memoria interna rápida de la computadora. Son muy útiles para ahorrar memoria. Sin embargo, es necesario considerar que sólo pueden utilizarse en aquellas aplicaciones en que sus componentes no reciban valores al mismo tiempo. Es decir, sólo uno de sus componentes puede recibir valor a la vez. El espacio de memoria reservado para una unión corresponde a la capacidad del campo de mayor tamaño.

Formalmente definimos una **unión** de la siguiente manera:

"Una unión es una colección de elementos finita y heterogénea, en la cual sólo uno de sus componentes puede recibir valor a la vez."

8.3.1. Declaración de uniones

La declaración de uniones es similar a la de estructuras. Observemos a continuación en el siguiente ejemplo la forma de declarar uniones.

EJEMPLO 8.6

Supongamos que debemos almacenar la siguiente información de cada alumno de una universidad:

- Matrícula del alumno (entero).
- Nombre del alumno (cadena de caracteres).
- Carrera del alumno (cadena de caracteres).
- Promedio del alumno (real).
    - Teléfono celular (cadena de caracteres).
    - Correo electrónico (cadena de caracteres).

El programa muestra la manera en que se declara una unión, así como la forma en que se tiene acceso a los campos de cada una de las variables de tipo unión, tanto para lectura como para escritura. Observa que en algunas variables de tipo estructura se utiliza una función para la lectura de los datos.

Programa 8.5

```
#include <stdio.h>
#include <string.h>

/* Uniones.
El programa muestra la manera como se declara una unión, así como la forma de
➥acceso a los campos de las variables de tipo unión tanto para asignación
➥de valores como para lectura y escritura. */

union datos                /* Declaración de una unión. */
{
   char celular[15];
   char correo[20];
};

typedef struct             /* Declaración de una estructura utilizando typedef. */
{
   int matricula;
   char nombre[20];
   char carrera[20];
   float promedio;
   union datos personales;
   /* Observa que uno de los campos de la estructura alumno es una unión. */
} alumno;

void Lectura(alumno a);      /*  Prototipo de función. */

void main(void)
{
alumno a1 = {120, "María", "Contabilidad", 8.9, "5-158-40-50"}, a2, a3;
/* Observa que sólo el primer componente de una unión puede recibir valores por
➥medio de este tipo de asignaciones. */

/* Para que puedas observar las diferentes formas en que los campos de las
➥variables de tipo estructura alumno reciben valores, ingresamos los valores
➥de los campos de tres formas diferentes. Los campos de a1 reciben valores
➥directamente, los campos de a2 se leen en el programa principal, y los campos
➥de a3 reciben valores a través de una función. */
printf("Alumno 2\n");
printf("Ingrese la matrícula: ");
scanf("%d", &a2.matricula);
fflush(stdin);
printf("Ingrese el nombre: ");
gets(a2.nombre);
fflush(stdin);
printf("Ingrese la carrera: ");
gets(a2.carrera);
printf("Ingrese el promedio: ");
scanf("%f", &a2.promedio);
fflush(stdin);
```

```
printf("Ingrese el correo electrónico: ");
gets(a2.personales.correo);
/* Observa que en la variable a2 de tipo estructura alumno el segundo campo de la
➥unión recibe un valor. */

printf("Alumno 3\n");
Lectura(&a3);  /* Se llama a una función para leer los campos de la variable a3. */

/* Impresión de resultados. */
printf("\nDatos del alumno 1\n");
printf("%d\n", a1.matricula);
puts(a1.nombre);
puts(a1.carrera);
printf("%.2f\n", a1.promedio);
puts(a1.personales.celular);
/* Observa que escribe el valor del teléfono celular asignado. */
➥puts(a1.personales.correo);      }
/* Observa que si tratamos de imprimir el campo correo, escribe basura. */

strcpy(a0.personales.correo, "hgimenez@hotmail.com");
/* Se ingresa ahora un valor al segundo campo de la unión de la variable a0. */

puts(a0.personales.celular);
/* Ahora escribe basura en el campo del teléfono celular. */
puts(a0.personales.correo);
/* Escribe el contenido del campo (hgimenez@hotmail.com). */

printf("\nDatos del alumno 2\n");
printf("%d\n", a2.matricula);
puts(a2.nombre);
puts(a2.carrera);
printf("%.2f\n", a2.promedio);
puts(a2.personales.celular);        /* Escribe basura. */
puts(a2.personales.correo);         /* Escribe el contenido del segundo campo. */

printf("Ingrese el teléfono celular del alumno 2: ");
fflush(stdin);
gets(a2.personales.celular);

puts(a2.personales.celular);        /* Escribe el teléfono celular ingresado. */
puts(a2.personales.correo);         /* Ahora escribe basura. */

printf("\nDatos del alumno 3\n");
printf("%d\n", a3.matricula);
puts(a3.nombre);
puts(a3.carrera);
printf("%.2f\n", a3.promedio);
puts(a3.personales.celular);
puts(a3.personales.correo);         /* Escribe basura. */
}
```

```
void Lectura(alumno *a)
/* La función Lectura se utiliza para leer los campos de una variable de tipo
➥estructura alumno. */
{
printf("\nIngrese la matrícula: ");
scanf("%d", &(*a).matricula);
fflush(stdin);
printf("Ingrese el nombre: ");
gets(a->nombre);
fflush(stdin);
printf("Ingrese la carrera: ");
gets((*a).carrera);
printf("Ingrese el promedio: ");
scanf("%f", &a->promedio);
printf("Ingrese el teléfono celular: ");
fflush(stdin);
gets(a->personales.celular);
}
```

Problemas resueltos

Problema PR8.1

Una comercializadora farmacéutica distribuye productos a distintas farmacias de la Ciudad de México. Para ello almacena en un arreglo unidimensional, *ordenado de menor a mayor en función de la clave*, toda la información relativa a sus productos:

- Clave del producto (entero).
- Nombre del producto (cadena de caracteres).
- Existencia (entero).
- Precio unitario (real).

Dato: `INV [N]` (donde `INV` es un arreglo unidimensional de tipo `PRODUCTO` de `N` elementos, $1 \leq N \leq 100$).

Realice un programa en **C** que construya los siguientes módulos:

a) Ventas. El módulo registra la venta de diferentes productos a un cliente —farmacia—. Obtiene el total de la venta y actualiza el inventario correspondiente. El fin de datos para la venta de un cliente es 0.

b) Reabastecimiento. Este módulo permite incorporar productos —cantidades— al inventario. El fin de datos es 0.

c) Nuevos Productos. El módulo permite incorporar nuevos productos al inventario. Los productos se encuentran ordenados en el arreglo por su clave. El fin de datos es 0.

d) Inventario. El módulo permite imprimir el inventario completo.

En la siguiente figura se muestra la representación gráfica de la estructura de datos necesaria para resolver este problema.

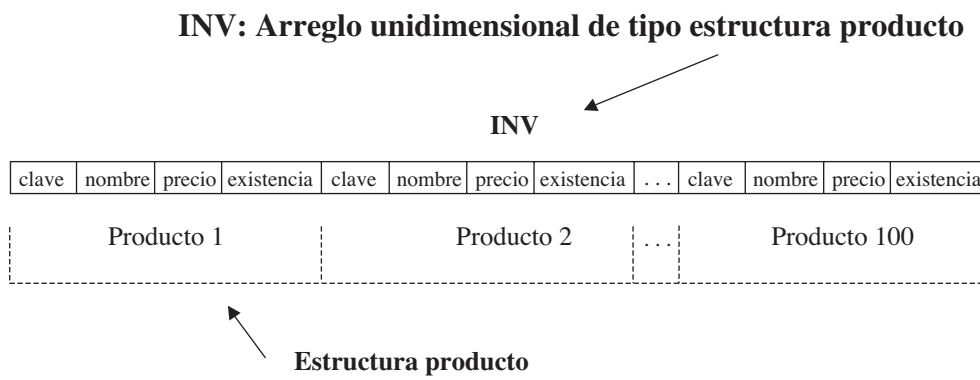


FIGURA 8.4

Representación gráfica de la estructura de datos necesaria para el problema PR8.1

Programa 8.6

```
#include <stdio.h>
#include <string.h>

/* Comercializadora farmacéutica.
El programa maneja información sobre ventas, inventario, reabastecimiento y
➥nuevos productos de una comercializadora farmacéutica. */

typedef struct                /* Declaración de la estructura producto. */
{
   int clave;
   char nombre[15];
   float precio;
   int existencia;
} producto;

void Lectura(producto *, int);          /*  Prototipos de funciones. */
void Ventas(producto *, int);
void Reabastecimiento(producto *, int);
void Nuevos_Productos(producto *, int *);
void Inventario(producto *, int);
```

```
void main(void)
{
producto INV[100];
/* Se declara un arreglo unidimensional de tipo estructura producto. */
int TAM, OPE;
do
{
   printf("Ingrese el número de productos: ");
   scanf("%d", &TAM);
}
while (TAM > 100  ||  TAM < 1);
/* Se verifica que el número de productos ingresados sea correcto. */
Lectura(INV, TAM);
printf("\nIngrese operación a realizar. \n\t\t1 - Ventas \n\t\t 2 -
➥Reabastecimiento \n\t\t
  3 - Nuevos Productos \n\t\t 4 - Inventario \n\t\t   0 - Salir: ");
scanf("%d", &OPE);
while (OPE)
{
   switch (OPE)
   {
      case 1: Ventas(INV, TAM);
         break;
      case 2: Reabastecimiento(INV, TAM);
         break;
      case 3: Nuevos_Productos(INV, &TAM);
         /* Se pasa el parámetro por referencia, porque se puede modificar el
         ➥número
            de elementos del arreglo en la función. */
         break;
      case 4: Inventario(INV, TAM);
         break;
   };
   printf("\nIngrese operación a realizar. \n\t\t1 - Ventas \n\t\t 2 -
   ➥Reabastecimiento
       \n\t\t 3 - Nuevos Productos \n\t\t 4 - Inventario \n\t\t   0 - Salir: ");
   scanf("%d", &OPE);
}
}

void Lectura(producto A[], int T)
/* Esta función se utiliza para leer un arreglo unidimensional de tipo
➥estructura producto de T elementos. */
{
int I;
for (I=0; I<T; I++)
{
   printf("\nIngrese información del producto %d", I+1);
   printf("\n\tClave: ");
   scanf("%d", &A[I].clave);
   fflush(stdin);
```

```
   printf("\tNombre:");
   gets(A[I].nombre);
   printf("\tPrecio:");
   scanf("%f", &A[I].precio);
   printf("\tExistencia: ");
   scanf("%d", &A[I].existencia);
}
}

void Ventas(producto A[], int T)
/* Esta función se utiliza para manejar las venta a un cliente. Se ingresan
➥productos y cantidades, el fin de datos está dado por el cero. Además de
➥obtener el total de las ventas, se actualiza el inventario. */
{
int CLA, CAN, I, RES;
float TOT, PAR;
printf("\nIngrese clave del producto -0 para salir-: ");
scanf("%d", &CLA);
TOT = 0.0;
while (CLA)
{
  printf("\tCantidad: ");
  scanf("%d", &CAN);
  I = 0;
  while ((I < T) && (A[I].clave < CLA))
  /* Se realiza una búsqueda para localizar la clave del producto. */
     I++;
  if ((I == T) || (A[I].clave > CLA))
       printf("\nLa clave del producto es incorrecta");
  else
       if (A[I].existencia >= CAN)
         /* Se analiza si el stock es suficiente para satisfacer el pedido. */
       {
            A[I].existencia -= CAN;    /* Se actualiza el stock del producto. */
            PAR = A[I].precio * CAN;
            TOT += PAR;
       }
       else
        {
          printf("\nNo existe en inventario la cantidad solicitada. Solo hay %d",
                   A[I].existencia);
           printf(" \nLos lleva 1 - Si   0 – No?: ");
           scanf("%d", &RES);
           if (RES)
           {
                 PAR = A[I].precio * A[I].existencia;
                 A[I].existencia = 0;          /* El stock queda en cero. */
                TOT += PAR;
           }
  }
```

```
  printf("\nIngrese la siguiente clave del producto -0 para salir-:);
  scanf("%d", &CLA);
}
printf("\nTotal de la venta:  %f", TOT);
}

void Reabastecimiento(producto A[], int T)
/* Esta función se utiliza para reabastecer al inventario. */
{
int CLA,CAN,I;
printf("\nIngrese clave del producto -0 para salir-: ");
scanf("%d", &CLA);
while (CLA)
{
  I = 0;
  while ((I < T) && (A[I].clave < CLA))
     I++;
  if ((I==T) ¦¦ (A[I].clave > CLA))
       printf("\nLa clave del producto ingresada es incorrecta");
  else
   {
        printf("\tCantidad: ");
        scanf("%d", &CAN);
        A[I].existencia += CAN;
   }
  printf("\nIngrese otra clave del producto -0 para salir-: ");
  scanf("%d", &CLA);
}
}

void Nuevos_Productos(producto A[], int *T)
/* Esta función se utiliza para incorporar nuevos productos al inventario.
➥Dado que los productos se encuentran ordenados por clave, puede suceder que
➥al insertar un nuevo producto haya que mover los elementos del arreglo para
➥que continúen ordenados. */
{
int CLA, I, J;
printf("\nIngrese clave del producto -0 para salir-: ");
scanf("%d", &CLA);
while ((*T < 30) && (CLA))
{
  I=0;
  while ((I < *T) && (A[I].clave < CLA))
   /* Búsqueda de la posición que le corresponde a CLA en el arreglo. */
     I++;
  if (I == *T)      /* Se inserta el elemento en la última posición. */
  {
        A[I].clave = CLA;
        printf("\tNombre:");
        fflush(stdin);
        gets(A[I].nombre);
```

```
        printf("\tPrecio:");
        scanf("%f", &A[I].precio);
        printf("\tCantidad: ");
        scanf("%d", &A[I].existencia);
        *T = *T + 1;
}
else
   if (A[I].clave == CLA)
       printf("\nEl producto ya se encuentra en el inventario");
   else
   {
       for (J=*T;  J>I;  J--)
        /* Se inserta el nuevo producto en el arreglo. Se mueven una posición
➥a la derecha los elementos
        del arreglo que tengan una clave de producto mayor a la ingresada. */
           A[J] = A[J-1];
       A[I].clave = CLA;
       printf("\tNombre:");
        fflush(stdin);
        gets(A[I].nombre);
       printf("\tPrecio:");
       scanf("%f", &A[I].precio);
       printf("\tCantidad: ");
       scanf("%d", &A[I].existencia);
       *T = *T + 1;
   }
  printf("\nIngrese otra clave de producto -0 para salir-: ");
  scanf("%d", &CLA);
}
if (*T == 30)
  printf("\nYa no hay espacio para incorporar nuevos productos");
}

void Inventario(producto A[], int T)
/* Esta función se utiliza para escribir la información almacenada en —el
➥inventario— un arreglo unidimensional de tipo estructura producto de T
➥elementos. */
{
int I;
for (I=0; I<T; I++)
{
  printf("\nClave: %d", A[I].clave);
  printf("\tNombre: %s", A[I].nombre);
  printf("\tPrecio: %d", A[I].precio);
  printf("\tExistencia: %d \n", A[I].existencia);
}
}
```

Problema PR8.2

En una escuela guardan la información de sus alumnos utilizando arreglos unidimensionales. Se registra la siguiente información de cada alumno en una estructura:

- Matrícula del alumno (entero).
- Nombre y apellido (cadena de caracteres).
- Materias y promedios (arreglo unidimensional de estructura).
    - Materia (cadena de caracteres).
    - Promedio (real).

Dato: `ALU[N]`, donde `ALU` es un arreglo unidimensional de tipo `ALUMNO` (`1 ≤ N ≤ 50`).

Escribe un programa en **C** que obtenga lo siguiente:

a) La matrícula y el promedio general de cada alumno.
b) Las matrículas de los alumnos cuya calificación en la tercera materia sea mayor a 9.
c) El promedio general de la materia 4.

Nota: El problema es similar al del ejemplo 8.5. Varía en la forma de almacenar las calificaciones del alumno. Además de éstas, aparece el nombre de la materia.

En la siguiente figura se muestra la representación gráfica de la estructura de datos necesaria para resolver este problema.

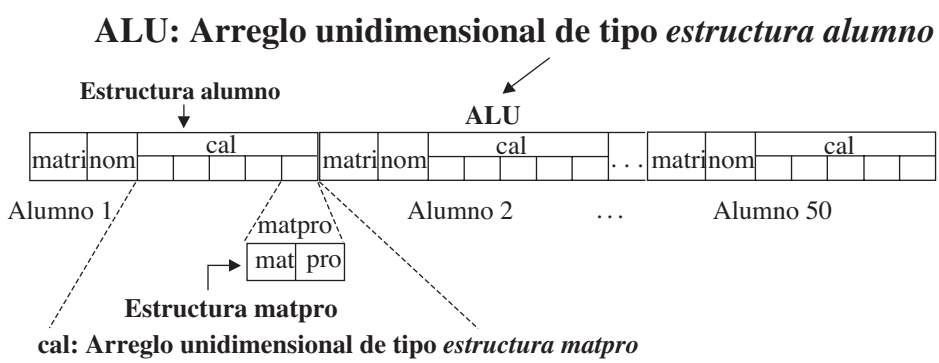


FIGURA 8.5
Representación gráfica de la estructura de datos necesaria para resolver el problema PR8.2

Programa 8.7

```
#include <stdio.h>
#include <string.h>

/* Escuela.
El programa genera información importante de los alumnos de una escuela. */

typedef struct                /* Declaración de la estructura matpro. */
{
  char mat[20];   /* Materia. */
  int pro;        /* Promedio. */
} matpro;

typedef struct                /* Declaración de la estructura alumno. */
{
  int matri;      /* Matrícula. */
  char nom[20];   /* Nombre del alumno. */
  matpro cal[5];  /* Observa que cal es un arreglo unidimensional de tipo
                  ➥estructura
                      matpro —la estructura definida en primer término. */
} alumno;

void Lectura(alumno * , int);
void F1(alumno *, int);
void F2(alumno *, int);                  /* Prototipos de funciones. */
void F3(alumno *, int);

void main(void)
{
alumno ALU[50];        /* ALU es un arreglo unidimensional de tipo alumno. */
int TAM;
do
{
  printf("Ingrese el tamaño del arreglo: ");
  scanf("%d", &TAM);
}
while (TAM > 50 || TAM < 1);  /* Se verifica que el tamaño del arreglo sea
                              ➥correcto. */
Lectura(ALU, TAM);
F1(ALU, TAM);
F2(ALU, TAM);
F3(ALU, TAM);
}

void Lectura(alumno A[], int T)
/* Esta función se utiliza para leer la información de un arreglo unidimensional
➥de tipo estructura alumno de T elementos. */
{
```

```
int I, J;
for(I=0; I<T; I++)
{
   printf("\nIngrese los datos del alumno %d", I+1);
   printf("\nIngrese la matrícula del alumno: ");
   scanf("%d", &A[I].matri);
   fflush(stdin);
   printf("Ingrese el nombre del alumno:");
   gets(A[I].nom);
   for (J=0; J<5; J++)
   {
        printf("\tMateria %d: ", J+1);
        fflush(stdin);
        gets(A[I].cal[J].mat);
        printf("\tPromedio %d: ", J+1);
        scanf("%d", &A[I].cal[J].pro);
   }
}
}

void F1(alumno A[], int T)
/* Esta función se utiliza para obtener la matrícula y el promedio general de
➥cada alumno. */
{
int I, J;
float SUM;
for (I=0; I<T; I++)
{
   printf("\nMatrícula del alumno : %d", A[I].matri);
   SUM = 0.0;
   for (J=0; J<5; J++)
        SUM = SUM + A[I].cal[J].pro;
   SUM = SUM / 5;
   printf("\tPromedio: %.2f", SUM);
}
}

void F2(alumno A[], int T)
/* Esta función se utiliza para obtener las matrículas de los alumnos cuya
➥calificación en la tercera materia es mayor a 9. */
{
int I;
printf("\nAlumnos con calificación mayor a 9 en la tercera materia");
for (I=0; I<T; I++)
   if (A[I].cal[2].pro > 9)
       printf("\nMatrícula del alumno : %d", A[I].matri);
}

void F3(alumno A[], int T)
/* Esta función se utiliza para obtener el promedio general de la cuarta materia. */
{
```

```
int I;
float SUM = 0.0;
for (I=0; I<T; I++)
   SUM = SUM + A[I].cal[3].pro;
SUM = SUM / T;
printf("\n\nPromedio de la cuarta materia: %.2f", SUM);
}
```

Problema PR8.3

En un hospital almacenan la siguiente información de sus pacientes:

- Nombre y apellido (cadena de caracteres).
- Edad (entero).
- Sexo (caracter).
- Condición (entero).
- Domicilio (estructura).
    - Calle (cadena de caracteres).
    - Número (entero).
    - Colonia (cadena de caracteres).
    - Código Postal (cadena de caracteres).
    - Ciudad (cadena de caracteres).
- Teléfono (cadena de caracteres).

Dato: `HOSPITAL[N]` (donde `HOSPITAL` es un arreglo unidimensional de tipo estructura `PACIENTE`, $1 \leq N \leq 100$).

Nota: Condición se refiere al estado de salud en que ingresa el paciente. Los valores que toma condición van de 1 a 5, y 5 representa el máximo grado de gravedad.

Escribe un programa en **C** que genere lo siguiente:

a) El porcentaje tanto de hombres como de mujeres registrados en el hospital.
b) El número de pacientes de cada una de las categorías de condición.
c) El nombre y teléfono de todos los pacientes que tuvieron una condición de ingreso de máxima gravedad (5).

En la siguiente figura se muestra la representación gráfica de la estructura de datos necesaria para resolver este problema.

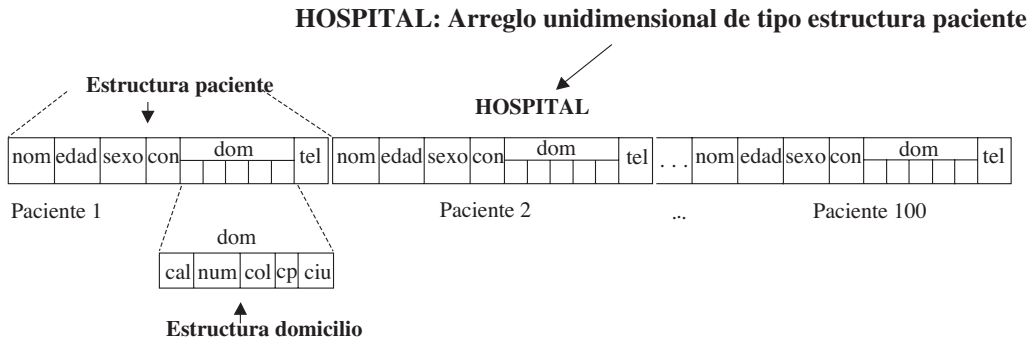


FIGURA 8.6

Representación gráfica de la estructura de datos necesaria para resolver el problema PR8.3

Programa 8.8

```
#include <stdio.h>
#include <string.h>

/* Hospital.
El programa genera información acerca de los pacientes de un hospital. */

typedef struct                  /* Declaración de la estructura domicilio. */
{
   char cal[20];          /* Calle. */
   int num;               /* Número. */
   char col[20];          /* Colonia. */
   char cp[5];            /* Código Postal. */
   char ciu[20];          /* Ciudad. */
} domicilio;

typedef struct                  /* Declaración de la estructura paciente. */
{
   char nom[20];          /* Nombre y apellido. */
   int edad;
   char sexo;
   int con;               /* Condición. */
   domicilio dom;           /* Observa que el campo dom es de tipo estructura
                            ➥domicilio. */
   char tel[10];          /* Teléfono. */
} paciente;

void Lectura(paciente *, int);
void F1(paciente *, int);
void F2(paciente *, int);                /* Prototipos de funciones. */
void F3(paciente *, int);

void main(void)
{
```

```
paciente HOSPITAL[100];   /* Arreglo unidimensional de tipo estructura
➥paciente. */
int TAM;
do
{
   printf("Ingrese el número de pacientes: ");
   scanf("%d", &TAM);
}
while (TAM > 50 || TAM < 1);   /* Se verifica que el tamaño del arreglo sea
                               ➥correcto. */
Lectura(HOSPITAL, TAM);
F1(HOSPITAL, TAM);
F2(HOSPITAL, TAM);
F3(HOSPITAL, TAM);
}

void Lectura(paciente A[], int T)
/* Esta función se utiliza para leer un arreglo unidimensional de tipo
➥estructura paciente de T elementos. */
{
int I;
for (I=0; I<T; I++)
{
   printf("\n\t\tPaciente %d", I+1);
   fflush(stdin);
   printf("\nNombre: ");
   gets(A[I].nom);
   printf("Edad: ");
   scanf("%d", &A[I].edad);
   printf("Sexo (F-M): ");
   scanf("%c", &A[I].sexo);
   printf("Condición (1..5): ");
   scanf("%d", &A[I].con);
   fflush(stdin);
   printf("\tCalle: ");
   gets(A[I].dom.cal);
   printf("\tNúmero: ");
   scanf("%d", &A[I].dom.num);
   fflush(stdin);
   printf("\tColonia: ");
   gets(A[I].dom.col);
   fflush(stdin);
   printf("\tCódigo Postal: ");
   gets(A[I].dom.cp);
   fflush(stdin);
   printf("\tCiudad: ");
   gets(A[I].dom.ciu);
   fflush(stdin);
   printf("Teléfono: ");
   gets(A[I].tel);
}
}

void F1(paciente A[], int T)
```

```
/* Esta función se utiliza para obtener el porcentaje tanto de hombres como de
➥mujeres registrados en el hospital. */
{
int I,FEM, MAS, TOT;
for (I=0; I<T; I++)
   switch (A[I].sexo)
   {
       case 'F': FEM++;
          break;
       case 'M': MAS++;
          break;
   }
TOT = FEM + MAS;
printf("\nPorcentaje de Hombres: %.2f%", (float)MAS / TOT * 100);
printf("\nPorcentaje de Mujeres: %.2f%", (float)FEM / TOT * 100);
}

void F2(paciente A[], int T)
/* Esta función se utiliza para obtener el número de pacientes que ingresaron al
➥hospital en cada una de las categorías de condición. */
{
int I, C1 = 0, C2 = 0, C3 = 0, C4 = 0, C5 = 0;
for (I=0; I<T; I++)
   switch (A[I].con)
   {
       case 1: C1++;
          break;
       case 2: C2++;
          break;
       case 3: C3++;
          break;
       case 4: C4++;
          break;
       case 5: C5++;
          break;
   }
printf("\nNúmero pacientes en condición 1: %d", C1);
printf("\nNúmero pacientes en condición 2: %d", C2);
printf("\nNúmero pacientes en condición 3: %d", C3);
printf("\nNúmero pacientes en condición 4: %d", C4);
printf("\nNúmero pacientes en condición 5: %d", C5);
}

void F3(paciente A[], int T)
/* La función F3 se utiliza para generar el nombre y teléfono de todos los
➥pacientes que tuvieron una condición de ingreso de máxima gravedad (5). */
{
int I;
printf("\nPacientes ingresados en estado de gravedad");
for (I=0; I<T; I++)
   if (A[I].con == 5)
        printf("\nNombre: %s\tTeléfono: %s", A[I].nom, A[I].tel);
}
```

Problema PR8.4

Una empresa de bienes raíces de Lima, Perú, lleva información sobre las propiedades que tiene disponibles tanto para venta como para renta.

- Clave de la propiedad (cadena de caracteres).
- Superficie cubierta (real).
- Superficie terreno (real).
- Características (cadena de caracteres).
- Ubicación geográfica.
    - Zona (cadena de caracteres).
    - Calle (cadena de caracteres).
    - Colonia (cadena de caracteres).
- Precio (real).
- Disponibilidad (caracter).

Dato: `PROPIE[N]` (donde `PROPIE` es un arreglo unidimensional de tipo estructura `PROPIEDADES`, $1 \leq N \leq 100$).

Escribe un programa en **C** que realice lo siguiente:

a) Un listado de las propiedades disponibles para venta en la zona de **Miraflores** cuyo valor oscile entre 450,000 y 650,000 nuevos soles.

b) Al recibir una zona geográfica y un cierto rango respecto al monto, obtenga un listado de todas las propiedades disponibles para renta.

Nota: El listado debe mostrar lo siguiente: clave de la propiedad, superficie cubierta, superficie total, características, calle, colonia y precio.

En la siguiente figura se muestra la representación gráfica de la estructura de datos necesaria para resolver este problema.

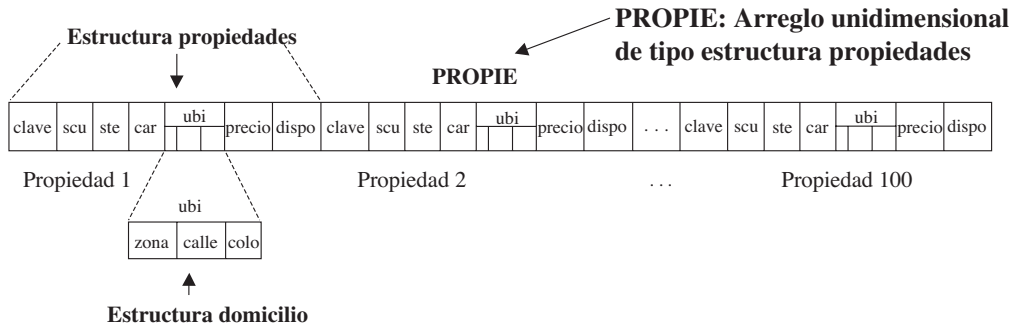


FIGURA 8.7

Representación gráfica de la estructura de datos necesaria para resolver el problema PR8.4

Programa 8.9

```
#include <stdio.h>
#include <string.h>

/* Bienes raíces.
El programa maneja información sobre las propiedades que tiene una empresa
➥de bienes raíces de la ciudad de Lima, Perú, tanto para venta como para
➥renta. */

typedef struct                    /* Declaración de la estructura ubicación.*/
{
   char zona[20];
   char calle[20];
   char colo[20];        /* Colonia. */
} ubicacion;

typedef struct                    /* Declaración de la estructura propiedades.*/
{
   char clave[5];
   float scu;              /* Superficie cubierta. */
   float ste;              /* Superficie terreno. */
   char car[50];           /* Características. */
   ubicacion ubi;             /* Observa que este campo es de tipo estructura
                              ubicación. */
   float precio;
   char dispo;                /* Disponibilidad. */
} propiedades;

void Lectura(propiedades , int);
void F1(propiedades *, int);          /* Prototipos de funciones. */
void F2(propiedades *, int);
```

```
void main(void)
{
propiedades PROPIE[100];
/* Se declara un arreglo unidimensional de tipo estructura propiedades. */
int TAM;
do
{
   printf("Ingrese el número de propiedades: ");
   scanf("%d", &TAM);
}
while (TAM > 100  || TAM < 1);
/* Se verifica que el tamaño del arreglo sea correcto. */
Lectura(PROPIE, TAM);
F1(PROPIE, TAM);
F2(PROPIE, TAM);
}

void Lectura(propiedades A[], int T)
/* Esta función se utiliza para leer un arreglo unidimensional de tipo estructura
➥propiedades de T elementos. */
{
int I;
for (I=0; I<T; I++)
{
   printf("\n\tIngrese datos de la propiedad %d", I + 1);
   printf("\nClave: ");
   fflush(stdin);
   gets(A[I].clave);
   printf("Superficie cubierta: ");
   scanf("%f", &A[I].scu);
   printf("Superficie terreno: ");
   scanf("%f", &A[I].ste);
   printf("Características: ");
   fflush(stdin);
   gets(A[I].car);
   printf("\tZona: ");
   fflush(stdin);
   gets(A[I].ubi.zona);
   printf("\tCalle: ");
   fflush(stdin);
   gets(A[I].ubi.calle);
   printf("\tColonia: ");
   fflush(stdin);
   gets(A[I].ubi.colo);
   printf("Precio: ");
   scanf("%f", &A[I].precio);
   printf("Disponibilidad (Venta-V  Renta-R): ");
   scanf("%c", &A[I].dispo);
}
}
```

```
void F1(propiedades A[], int T)
/* Esta función se utiliza para generar un listado de las propiedades
➥disponibles para venta en la zona de Miraflores, cuyo valor oscila entre
➥450,000 y 650,000 nuevos soles. */
{
int I;
printf("\n\t\tListado de Propiedades para Venta en Miraflores");
for (I=0; I<T; I++)
   if ((A[I].dispo == 'V') && (strcmp (A[I].ubi.zona, "Miraflores") == 0))
        if ((A[I].precio >= 450000) && (A[I].precio <= 650000))
        {
              printf("\nClave de la propiedad: ");
              puts(A[I].clave);
             printf("\nSuperficie cubierta: %f", A[I].scu);
             printf("\nSuperficie terreno: %f", A[I].ste);
             printf("\nCaracterísticas: ");
             puts(A[I].car);
             printf("Calle: ");
             puts(A[I].ubi.calle);
             printf("Colonia: ");
             puts(A[I].ubi.colo);
             printf("Precio: %.2f\n", A[I].precio);
        }
}

void F2(propiedades A[], int T)
/* Al recibir como datos una zona geográfica de Lima, Perú, y un cierto rango
➥respecto al monto, esta función genera un listado de todas las propiedades
➥disponibles para renta. */
{
int I;
float li, ls;
char zon[20];
printf("\n\t\tListado de Propiedades para Renta");
printf("\nIngrese zona geográfica: ");
fflush(stdin);
gets(zon);
printf("Ingrese el límite inferior del precio:");
scanf("%f", &li);
printf("Ingrese el límite superior del precio:");
scanf("%f", &ls);
for (I=0; I<T; I++)
   if ((A[I].dispo == 'R') && (strcmp (A[I].ubi.zona, zon) == 0))
        if ((A[I].precio >= li) && (A[I].precio <= ls))
   {
             printf("\nClave de la propiedad: ");
             puts(A[I].clave);
            printf("\nSuperficie cubierta: %d", A[I].scu);
            printf("\nSuperficie terreno: %d", A[I].ste);
```

```
            printf("\nCaracterísticas: ");
            puts(A[I].car);
            printf("Calle: ");
            puts(A[I].ubi.calle);
            printf("Colonia: ");
            puts(A[I].ubi.colo);
            printf("Precio: %.2f", A[I].precio);
    }
}
```

Problema PR8.5

En una empresa de artículos domésticos almacenan la siguiente información de cada uno de sus vendedores:

- Número vendedor (entero).
- Nombre y apellido (cadena de caracteres).
- Ventas del año (arreglo unidimensional de reales).
- Domicilio (estructura).
    - Calle y número (cadena de caracteres).
    - Colonia (cadena de caracteres).
    - Código Postal (cadena de caracteres).
    - Ciudad (cadena de caracteres).
- Salario mensual (real).
- Clave forma de pago (entero).
- Forma de pago (unión).
    - Banco (estructura).
- Nombre del banco (cadena de caracteres).
- Número de cuenta (cadena de caracteres).
    - Ventanilla (caracter).

Dato: `VENDEDORES[N]` (donde `VENDEDORES` es un arreglo unidimensional de tipo estructura `VENDEDOR, 1 ≤ N ≤ 100`).

Notas: `Ventas del año` es un arreglo unidimensional de 12 elementos de reales, en el que se almacenan las ventas de los empleados en cada uno de los meses. `Forma de pago` es una unión en la que se almacena la forma de pago al empleado: cuenta de cheques, nómina o ventanilla. En los dos primeros casos se utiliza una estructura para almacenar el nombre del banco y el número de cuenta del empleado.

Escribe un programa en **C** que realice lo siguiente:

a) Obtenga las ventas totales anuales de cada uno de los empleados.

b) Incremente 5% el salario a todos aquellos empleados cuyas ventas anuales superaron $1,500,000.

c) Liste el número de empleado, el nombre y total de ventas, de todos aquellos vendedores que en el año vendieron menos de $300,000.

d) Liste el número de empleado, el nombre del banco y el número de cuenta de todos aquellos empleados a quienes se les deposita en cuenta de cheques.

En la siguiente figura se muestra la representación gráfica de la estructura de datos necesaria para resolver este problema.

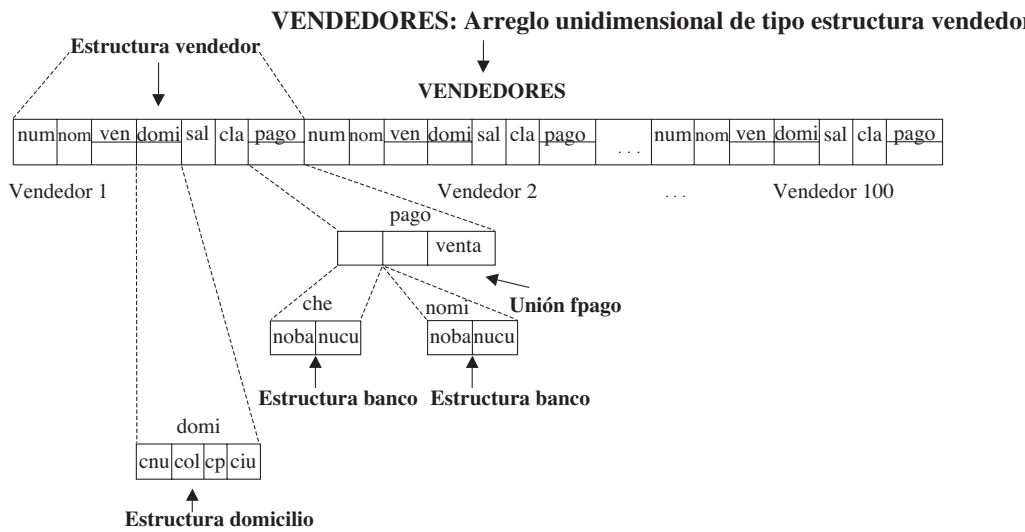


FIGURA 8.8

Representación gráfica de la estructura de datos necesaria para resolver el problema PR8.5

Programa 8.10

```
#include <stdio.h>
#include <string.h>

/* Vendedores.
El programa maneja información sobre las ventas que realizan los vendedores de
➥artículos domésticos de una importante empresa de la Ciudad de México. */
typedef struct                  /* Declaración de la estructura banco. */
{
```

```
   char noba[10];        /* Nombre del banco. */
   char nucu[10];        /* Número de cuenta. */
} banco;

typedef union                    /* Declaración de la union fpago. */
{
   banco che;            /* Cheque. Campo de tipo estructura banco. */
   banco nomi;           /* Cómina. Campo de tipo estructura banco. */
   char venta;           /* Ventanilla. */
} fpago;

typedef struct                   /* Declaración de la estructura domicilio. */
{
   char cnu[20];         /* Calle y número. */
   char col[20];         /* Colonia. */
   char cp[5];           /* Código Postal. */
   char ciu[15];         /* Ciudad. */
} domicilio;

typedef struct                   /* Declaración de la estructura vendedor. */
{
   int num;             /* Número de vendedor. */
   char nom[20];        /* Nombre del vendedor. */
   float ven[12];       /* Ventas del año. Arreglo unidimensional de tipo real. */
   domicilio domi;      /* domi es de tipo estructura domicilio. */
   float sal;             /* Salario mensual. */
   fpago pago;            /* pago es de tipo unión fpago. */
   int cla;               /* Clave forma de pago. */
} vendedor;

void Lectura(vendedor *, int);
void F1(vendedor *, int);
void F2(vendedor *, int);        /* Prototipos de funciones. */
void F3(vendedor *, int);
void F4(vendedor *, int);

void main(void)
{
vendedor VENDEDORES[100];
/* Declaración del arreglo unidimensional de tipo estructura vendedor. */
int TAM;
do
{
   printf("Ingrese el número de vendedores: ");
   scanf("%d", &TAM);
}
while (TAM > 100  || TAM < 1);
/* Se verifica que el número de elementos del arreglo sea correcto. */
Lectura (VENDEDORES, TAM);
F1 (VENDEDORES, TAM);
F2 (VENDEDORES, TAM);
```

```
F3 (VENDEDORES, TAM);
F4 (VENDEDORES, TAM);
printf("\n\tFIN DEL PROGRAMA");
}

void Lectura(vendedor A[], int T)
/* Esta función se utiliza para leer un arreglo unidimensional de tipo
➥estructura vendedor de T elementos. */
{
int I, J;
for (I=0; I<T; I++)
{
   printf("\n\tIngrese datos del vendedor %d", I+1);
   printf("\nNúmero de vendedor: ");
   scanf("%d", &A[I].num);
   printf("Nombre del vendedor: ");
   fflush(stdin);
   gets(A[I].nom);
   printf("Ventas del año: \n");
   for (J=0; J<12; J++)
   {
       printf("\tMes %d: ", J+1);
       scanf("%f", &A[I].ven[J]);
   }
   printf("Domicilio del vendedor: \n");
   printf("\tCalle y número: ");
   fflush(stdin);
   gets(A[I].domi.cnu);
   printf("\tColonia: ");
   fflush(stdin);
   gets(A[I].domi.col);
   printf("\tCódigo Postal: ");
   fflush(stdin);
   gets(A[I].domi.cp);
   printf("\tCiudad: ");
   fflush(stdin);
   gets(A[I].domi.ciu);
   printf("Salario del vendedor: ");
   scanf("%f", &A[I].sal);
   printf("Forma de Pago (Banco-1 Nómina-2 Ventanilla-3): ");
   scanf("%d", &A[I].cla);
   switch (A[I].cla)
   {
       case 1:{
                 printf("\tNombre del banco: ");
                   fflush(stdin);
                 gets(A[I].pago.che.noba);
                 printf("\tNúmero de cuenta: ");
                   fflush(stdin);
                 gets(A[I].pago.che.nucu);
             }
```

```
                break;
          case 2:{
                     printf("\tNombre del banco: ");
                       fflush(stdin);
                     gets(A[I].pago.nomi.noba);
                     printf("\tNúmero de cuenta: ");
                       fflush(stdin);
                     gets(A[I].pago.nomi.nucu);
                }
                break;
          case 3: A[I].pago.venta = 'S';
                break;
     }
}
}

void F1(vendedor A[], int T)
/* Esta función se utiliza para generar las ventas totales anuales de cada uno
➥de los vendedores de la empresa. */
{
int I, J;
float SUM;
printf("\n\t\tVentas Totales de los Vendedores");
for (I=0; I<T; I++)
{
   printf("\nVendedor: %d", A[I].num);
   SUM = 0.0;
   for (J=0; J<12; J++)
        SUM += A[I].ven[J];
   printf("\nVentas: %.2f\n", SUM);
}
}

void F2(vendedor A[], int T)
/* Esta función se utiliza para incrementar 5% el salario de todos aquellos
➥vendedores cuyas ventas anuales superaron $1,500,000. */
{
int I, J;
float SUM;
printf ("\n\t\tIncremento a los Vendedores con Ventas > 1,500,000$");
for (I=0; I<T; I++)
{
   SUM = 0.0;
   for (J=0; J<12; J++)
        SUM += A[I].ven[J];
   if (SUM > 1500000.00)
   {
        A[I].sal = A[I].sal * 1.05;
        printf("\nNúmero de empleado: %d\nVentas: %.2f\nNuevo salario: %.2f",
                A[I].num, SUM, A[I].sal);
   }
```

```
}
}

void F3(vendedor A[], int T)
/* Esta función se utiliza para generar un listado de todos aquellos
➥vendedores que en el año vendieron menos de $300,000. */
{
int I, J;
float SUM;
printf("\n\t\tVendedores con Ventas < 300,000");
for (I=0; I<T; I++)
{
   SUM = 0.0;
   for (J=0; J<12; J++)
        SUM += A[I].ven[J];
   if (SUM < 300000.00)
        printf("\nNúmero de empleado: %d\nNombre: %s\nVentas: %.2f", A[I].num,
               A[I].nom, SUM);
}
}

void F4(vendedor A[], int T)
/* Esta función se usa para imprimir el número de empleado, el nombre del
➥banco y el número de cuenta de todos aquellos empleados a quienes se les
➥deposita su sueldo en cuenta de cheques. */
{
int I;
float SUM;
printf("\n\t\tVendedores con Cuenta en el Banco");
for (I=0; I<T; I++)
   if (A[I].cla == 1)
        printf("\nNúmero de vendedor: %d\n Banco: %s\nCuenta: %s",
        ➥A[I].num,
               A[I].pago.che.noba, A[I].pago.che.nucu);
}
```

Problemas suplementarios

Problema PS8.1

Un importante banco, cuya casa central se encuentra ubicada en Quito, Ecuador, lleva la información de sus clientes en un arreglo unidimensional. Éste se encuentra ordenado en función del número de cuenta. El banco almacena la siguiente información de cada cliente:

- Número de cuenta (entero extendido).
- Nombre del cliente (cadena de caracteres).
- Domicilio (estructura).
    - Calle y número (cadena de caracteres).
    - Código Postal (cadena de caracteres).
    - Colonia (cadena de caracteres).
    - Ciudad (cadena de caracteres).
    - Teléfono (cadena de caracteres).
- Saldo (real).

Dato: `CLI[N]` (donde `CLI` es un arreglo unidimensional de tipo estructura `CLIENTE` de `N` elementos, `1 ≤ N ≤ 100`).

Escribe un programa en **C** que realice las siguientes operaciones:

a) Depósitos. Al recibir el número de cuenta de un cliente y un monto determinado, debe actualizar el saldo.

b) Retiros. Al recibir el número de cuenta de un cliente y un monto determinado por medio de un cheque o un retiro de un cajero, el programa debe actualizar el saldo. El cajero no puede pagar el cheque o autorizar el retiro si el saldo es insuficiente.

Nota: El programa debe realizar y validar diferentes transacciones. El fin de datos se expresa al ingresar el número 0.

Problema PS8.2

La Federación Mexicana de Fútbol (FEMEXFUT) almacena en un arreglo unidimensional la información de la tabla de posiciones de sus torneos apertura y clausura. Las estadísticas se ordenan lógicamente en función de los puntos. Se almacena la siguiente información de cada equipo:

- Nombre del equipo (cadena de caracteres).
- Partidos jugados (entero).
- Partidos ganados (entero).
- Partidos empatados (entero).
- Partidos perdidos (entero).
- Goles a favor (entero).
- Goles en contra (entero).
- Diferencia de goles (entero).
- Puntos (entero).

Dato: `FUTBOL[20]` (donde `FUTBOL` es un arreglo unidimensional de tipo estructura `EQUIPO`).

Escribe un programa en **C** que actualice la información después de cada fecha. El programa recibe la información de la siguiente manera:

```
América 0 – Puebla 2
Cruz Azul 3 – Veracruz 2
Necaxa 2 – Monterrey 3
. . .
```

Después de actualizar la información, el programa debe escribir la nueva tabla de posiciones, ordenada en función de los puntos de cada equipo.

Problema PS8.3

En una universidad de Barranquilla, en Colombia, almacenan la información de sus profesores utilizando arreglos unidimensionales. La siguiente información de cada profesor se almacena en una estructura:

- Número de empleado (entero).
- Nombre y apellido (cadena de caracteres).
- Departamento al que pertenece (cadena de caracteres).
- Puesto que ocupa (cadena de caracteres).
- Grado académico (cadena de caracteres).
- Nacionalidad (cadena de caracteres).
- Salario (arreglo unidimensional de reales).

Dato: `EMPLE[N]` (donde `EMPLE` es un arreglo unidimensional de tipo estructura `PROFESOR, 1 ≤ N ≤ 200`).

Nota: Salario es un arreglo unidimensional de tipo real de 12 posiciones que almacena los ingresos mensuales de los profesores. Considera además que en la universidad existen cuatro departamentos: Economía, Derecho, Computación y Administración.

Escribe un programa en **C** que obtenga lo siguiente:

a) El nombre, departamento al que pertenece y nacionalidad del profesor que más ganó el año anterior. También debe escribir el ingreso total del profesor.

b) El monto total pagado a los profesores extranjeros (nacionalidad diferente a Colombia) y el porcentaje respecto al monto total erogado por la universidad.

c) El departamento que más egresos —pago de salarios— tuvo el año anterior.

Problema PS8.4

En una empresa ubicada en Santiago de Chile almacenan la siguiente información de cada uno de sus empleados:

- Número de empleado (entero).
- Nombre y apellido (cadena de caracteres).
- Departamento (cadena de caracteres).
- Domicilio (estructura).
    - Calle y número (cadena de caracteres).
    - Colonia (cadena de caracteres).
    - Código Postal (cadena de caracteres).
    - Ciudad (cadena de caracteres).
    - Teléfono (cadena de caracteres).
- Salario mensual (real).

Dato: `EMPLE[N]` (donde `EMPLE` es un arreglo unidimensional, ordenado en función del número de empleado, de tipo estructura `EMPLEADO`, $1 \leq N \leq 100$).

Escribe un programa en **C** que contemple los siguientes módulos:

a) Altas. Al recibir el número de un empleado, debe darlo de alta incorporando lógicamente todos los datos del empleado.

b) Bajas. Al recibir el número de un empleado, debe darlo de baja.

c) Listado. Al recibir el nombre de un departamento, debe escribir el número de cada uno de sus empleados, sus nombres y salarios correspondientes.

Problema PS8.5

Una tienda especializada en artículos electrónicos vende como máximo 100 productos diferentes. La información de cada producto se almacena en una estructura:

- Clave del producto (entero).
- Nombre del producto (cadena de caracteres).
- Existencia (entero).

Dato: TIENDA[N] (donde TIENDA es un arreglo unidimensional de tipo estructura Producto de N elementos, 1 ≤ N ≤ 100).

Escribe un programa que actualice la información de acuerdo con las siguientes transacciones:

OPE_1 CLA_1 CAN_1
OPE_2 CLA_2 CAN_2
. . .
'0' 0 0

Donde:

OPE_i es una variable de tipo caracter que representa el tipo de operación que se realiza: 'C' compras, 'V' ventas.
CLA_i es una variable de tipo entero que representa la clave del producto.
CAN_i es una variable de tipo entero que significa la cantidad del producto.

Problema PS8.6

En una escuela privada de la Ciudad de México almacenan la información de cada uno de sus alumnos en un arreglo unidimensional de tipo estructura. Se almacena la siguiente información de cada alumno:

- Matrícula del alumno (entero).
- Nombre y apellido (cadena de caracteres).
- Domicilio (estructura).
    - Calle y número (cadena de caracteres).
    - Código Postal (entero).
    - Colonia (cadena de caracteres).
    - Ciudad (cadena de caracteres).
    - Teléfono (cadena de caracteres).
- Nivel de Estudios (estructura).
    - Nivel (cadena de caracteres).
    - Grado (entero).
    - Salón (cadena de caracteres).
    - Calificaciones (arreglo unidimensional de estructuras).
        - Materia (cadena de caracteres).
        - Promedio (real).

Dato: ESCUELA[N] (donde ESCUELA es un arreglo unidimensional de tipo estructura Alumno, 1 ≤ N ≤ 1000).

Nota: Cada alumno tiene siete materias en sus cursos.

Escribe un programa en **C** que realice lo siguiente:

a) Al recibir como dato la matrícula de un alumno, calcule e imprima el promedio general del mismo.

b) Al recibir como datos el nivel de estudios (primaria, secundaria o preparatoria), el grado y el salón, liste la matrícula de todos los alumnos, el nombre y su promedio.

c) Al recibir como datos el nivel de estudios (primaria, secundaria o preparatoria), el grado y el salón, obtenga el alumno que tiene el mayor promedio. Debe escribir la matrícula, su nombre y el promedio correspondiente.

CAPÍTULO 9

Archivos de datos

9.1. Introducción

En la actualidad es común procesar volúmenes de información tan grandes que es prácticamente imposible almacenar los datos en la memoria interna rápida —memoria principal— de la computadora. Estos datos se guardan generalmente en dispositivos de almacenamiento secundario como cintas y discos, en forma de **archivos de datos**, los cuales nos permiten almacenar la información de manera permanente y acceder a ella o modificarla cada vez que sea necesario.

Los archivos de datos se utilizan cuando el volumen de datos es significativo, o bien, cuando la aplicación requiere de la permanencia de los datos aun después de terminar de ejecutarse. En la actualidad, prácticamente todas las aplicaciones requieren almacenar datos en un archivo; por ejemplo, las aplicaciones de los bancos, casas de

bolsa, líneas aéreas para la reservación de vuelos y asientos, hospitales, hoteles, escuelas, etc.

Los archivos de datos se almacenan en dispositivos periféricos, como las cintas y los discos. En la mayoría de los casos, estos dispositivos no se encuentran físicamente en el lugar en el que trabajamos con la computadora. Por lo tanto, las operaciones de búsqueda, inserción, modificación y eliminación que se realizan sobre archivos tienen un alto costo en cuanto al tiempo.

Por ejemplo, imagina que te encuentras de vacaciones en Europa y deseas consultar el saldo de tu tarjeta de crédito en un cajero automático. Seguramente para ti es una operación inmediata, de unos pocos segundos, pero la información debe viajar a través del océano Atlántico para consultar el saldo que tienes en tu cuenta en una máquina que se encuentra en México y luego debe regresar a Europa para informarte cuál es tu saldo. El tiempo indudablemente juega un papel fundamental. No olvides que además del tiempo de ida y vuelta entre Europa y América, debes considerar el tiempo que se necesita para localizar tu cuenta en la computadora, considerando que los grandes bancos mexicanos tienen más de ocho millones de cuentahabientes.

Una forma de optimizar estas operaciones es utilizar medios de comunicación como la fibra óptica y/o el satélite entre la terminal y el servidor en el que se encuentra la información, y estructuras de datos poderosas, como árboles-B, para localizar la información dentro del archivo o la base de datos correspondiente.

El formato de los archivos generalmente es de **texto** o **binario**, y la forma de acceso a los mismos es **secuencial** o de **acceso directo**. En los primeros lenguajes de alto nivel como Pascal existía prácticamente una relación directa entre el formato del archivo y el método de acceso utilizado. Sin embargo, las cosas han cambiado con el tiempo, en el lenguaje **C** existen funciones que permiten trabajar con métodos de acceso directo aun cuando el archivo tenga un formato tipo texto. La relación que existe actualmente entre el formato del archivo y el método de acceso no es muy clara.

9.2. Archivos de texto y método de acceso secuencial

En los **archivos de texto** los datos se almacenan en formato texto y ocupan posiciones consecutivas en el dispositivo de almacenamiento secundario. La única

forma de acceder a los componentes de un archivo de texto es hacerlo en forma **secuencial**. Es decir, accediendo al primer componente, luego al segundo, y así sucesivamente hasta llegar al último, y por consiguiente al fin del archivo. Un elemento importante cuando se trabaja con archivos de texto es el *área del búfer*, que es el lugar donde los datos se almacenan temporalmente mientras se transfieren de la memoria al dispositivo secundario en que se encuentran o viceversa.

El lenguaje de programación **C** no impone restricciones ni formatos específicos para almacenar elementos en un archivo. Además, proporciona un conjunto extenso de funciones de biblioteca para el manejo de archivos. Es importante señalar que antes de trabajar con un archivo debemos abrirlo y cuando terminamos de trabajar con él debemos cerrarlo por seguridad de la información que se haya almacenado. En el lenguaje **C** un **archivo** básicamente se abre y cierra de la siguiente forma:

```
    /* El conjunto de instrucciones muestra la sintaxis para abrir y ce-
rrar un Âarchivo en
       el lenguaje de programación C. */

    . . .
    FILE *apuntador_archivo;
    apuntador_archivo = fopen (nombre_archivo, "tipo_archivo");
    if (apuntador_archivo != NULL)
    {
          proceso;                    /* trabajo con el archivo.  */
          fclose(apuntador_archivo);
    }
    else
          printf("No se puede abrir el archivo");
    . . .
```

La primera instrucción:

```
FILE *apuntador_archivo;
```

indica que `apuntador_archivo` es un apuntador al inicio de la estructura `FILE`, área del búfer que siempre se escribe con mayúsculas. La segunda instrucción:

```
apuntador_archivo = fopen (nombre_archivo, "tipo-archivo");
```

permite abrir un archivo llamado `nombre_archivo` que puede ser una variable de tipo cadena de caracteres, o bien, una constante sin extensión o con extensión `txt` para realizar actividades de `tipo_archivo`. Observa que la función `fopen` tiene dos argumentos: el nombre del archivo y el tipo de archivo, que puede ser de lectura, escritura, etc. En la siguiente tabla se muestran los diferentes tipos de archivos.

TABLA 9.1. Tipos de archivos

Tipo de archivo	*Explicación*
`"r"`	Se abre un archivo sólo para lectura.
`"w"`	Se abre un archivo sólo para escritura. Si el archivo ya existe, el apuntador se coloca al inicio y sobrescribe, destruyendo al archivo anterior.
`"a"`	Se abre un archivo para agregar nuevos datos al final. Si el archivo no existe, crea uno nuevo.
`"r+"`	Se abre un archivo para realizar modificaciones. Permite leer y escribir. El archivo tiene que existir.
`"w+"`	Se abre un archivo para leer y escribir. Si el archivo existe, el apuntador se coloca al inicio, sobrescribe y destruye el archivo anterior.
`"a+"`	Se abre un archivo para lectura y para incorporar nuevos datos al final. Si el archivo no existe, se crea uno nuevo.

La siguiente instrucción:

```
if (apuntador_archivo != NULL)
```

permite evaluar el contenido del apuntador. Si éste es igual a `NULL`, implica que el archivo no se pudo abrir, en cuyo caso es conveniente escribir un mensaje para notificar esta situación. Por otra parte, si el contenido del apuntador es distinto de `NULL` entonces se comienza a trabajar sobre el archivo. Por último, la instrucción:

```
fclose (apuntador_archivo);
```

se utiliza para cerrar el archivo.

Analicemos a continuación diferentes ejemplos que nos permitirán comprender perfectamente tanto el manejo de los archivos de texto, como el método de acceso secuencial.

EJEMPLO 9.1

En el siguiente programa podemos observar la forma como se abre y cierra un archivo de texto, y la manera como se almacenan caracteres. Se recomienda el uso de las instrucciones `getc` y `fgetc` para lectura de caracteres, así como `putc` y `fputc` para escritura de caracteres.

Programa 9.1

```
#include <stdio.h>

/* Archivos y caracteres.
 El programa escribe caracteres en un archivo. */

void main(void)
{
char p1;
FILE *ar;
ar = fopen("arc.txt", "w");    /* Se abre el archivo arc.txt para escritura. */
if (ar != NULL)
{
    while ((p1=getchar()) !=  '\n')
    /* Se escriben caracteres en el archivo mientras no se detecte el caracter
    ➥que indica el fin de la línea. */
         fputc(p1, ar);
   fclose(ar);                          /* Se cierra el archivo. */
}
else
   printf("No se puede abrir el archivo");
}
```

EJEMPLO 9.2

En el siguiente programa podemos observar la forma como se leen caracteres de un archivo. Se introduce la instrucción `feof()`, que se utiliza para verificar el fin del archivo.

Programa 9.2

```
#include <stdio.h>

/* Archivos y caracteres.
 El programa lee caracteres de un archivo. */

void main(void)
{
char p1;
FILE *ar;
if ((ar = fopen("arc.txt", "r")) != NULL)  /* Se abre el archivo para lectura. */
➥/* Observa que las dos instrucciones del programa 9.1 necesarias para abrir un
➥archivo y verificar que éste en realidad se haya abierto, se pueden agrupar
➥en una sola instrucción. */
{
```

```
      while (!feof(ar))
      /* Se leen caracteres del archivo mientras no se detecte el fin del
➥archivo. */
      {
         p1 = fgetc(ar);      /* Lee el caracter del archivo. */
         putchar(p1);         /* Despliega el caracter en la pantalla. */
      }
      fclose(ar);
}
else
    printf("No se puede abrir el archivo");
}
```

EJEMPLO 9.3

En el siguiente programa podemos observar la forma como se manejan las cadenas de caracteres en un archivo. Se introducen las instrucciones `fgets` y `fputs` para lectura y escritura de cadenas de caracteres, respectivamente.

Programa 9.3

```
#include <stdio.h>

/* Archivos y cadenas de caracteres.
El programa escribe cadenas de caracteres en un archivo. */

void main(void)
{
char cad[50];
int res;
FILE *ar;
if ((ar = fopen("arc.txt", "w")) != NULL)
/* Se abre el archivo para escritura. En la misma instrucción se verifica si se
➥pudo abrir. */
{
    printf("\n¿Desea ingresar una cadena de caracteres?  Sí-1 No-0:");
    scanf("%d", &res);
    while (res)
    {
       fflush(stdin);
       printf("Ingrese la cadena: ");
       gets(cad);
       fputs(cad, ar);     /* Observa la forma como se escribe la cadena en el
                           ➥archivo.*/
       printf("\n¿Desea ingresar otra cadena de caracteres?  Sí-1  No-0:");
       scanf("%d", &res);
       if (res)
```

```
            fputs("\n", ar);
            /* Se indica un salto de línea, excepto en la última cadena. Si no
            ➥se hiciera esta indicación, la función fputs pegaría las cadenas y
            ➥luego tendríamos dificultades en el momento de leerlas. Por otra
            ➥parte, si realizáramos este salto de línea al final de la última
            ➥cadena, en la escritura se repetiría la última cadena. */
    }
    fclose(ar);
}
else
    printf("No se puede abrir el archivo");
}
```


EJEMPLO 9.4

En el siguiente programa podemos observar la forma como se leen las cadenas de caracteres de un archivo.

Programa 9.4

```
include <stdio.h>

/* Archivos y cadenas de caracteres.
El programa lee cadenas de caracteres de un archivo. */

void main(void)
{
char cad[50];
FILE *ap;
if ((ap=fopen ("arc.txt", "r")) != NULL)
/* Se abre el archivo para lectura y se verifica si se abrió correctamente. */
{
    while (!feof(ap))
    /* Mientras no se detecte el fin de archivo se siguen leyendo cadenas de
    ➥caracteres. */
    {
         fgets(cad, 50, ap);
         /* Observa que la instrucción para leer cadenas requiere de tres
         ➥argumentos. */
        puts(cad);       /* Despliega la cadena en la pantalla. */
    }
    fclose(ap);
}
else
    printf("No se puede abrir el archivo");
}
```

EJEMPLO 9.5

Escribe un programa en **C** que, al recibir como datos la matrícula y cinco calificaciones de N alumnos de un curso universitario, almacene la información en el archivo tal como se presenta a continuación.

Datos: N

MAT_1, $CAL_{1,1}$, $CAL_{1,2}$, $CAL_{1,3}$, $CAL_{1,4}$, $CAL_{1,5}$
MAT_2, $CAL_{2,1}$, $CAL_{2,2}$, $CAL_{2,3}$, $CAL_{2,4}$, $CAL_{2,5}$
. . .
MAT_N, $CAL_{N,1}$, $CAL_{N,2}$, $CAL_{N,3}$, $CAL_{N,4}$, $CAL_{N,5}$

Donde: N es una variable de tipo entero que representa el número de alumnos, $1 \leq N \leq 35$.

MAT_i es una variable de tipo entero que representa la matrícula del alumno i.

$CAL_{i,j}$ es una variable de tipo real que indica la calificación j del alumno i.

Nota: En el programa se puede observar la forma como se manejan las variables enteras y reales en un archivo. Se introducen las instrucciones `fscanf` y `fprintf` para lectura y escritura, respectivamente.

Programa 9.5

```
#include <stdio.h>

/* Archivos con variables enteras y reales.
El programa almacena datos de un grupo de alumnos en un archivo. */

void main(void)
{
int i, j, n, mat;
float cal;
FILE *ar;
printf("\nIngrese el número de alumnos: ");
scanf("%d", &n);
/* Se asume que el valor que ingresa el usuario está comprendido entre 1 y 35. */
if ((ar = fopen("arc8.txt", "w")) != NULL)
{
      fprintf(ar,"%d ", n);            /* Se escribe el número de alumnos en el
                                       ➥archivo. */
    for (i=0; i<n; i++)
    {
         printf("\nIngrese la matrícula del alumno %d: ", i+1);
         scanf("%d", &mat);
         fprintf(ar,"\n%d ", mat);   /* Se escribe la matrícula en el
                                     ➥archivo. */
```

```
          for (j=0; j<5; j++)
          {
               printf("\nCalificación %d: ", j+1);
               scanf("%f", &cal);
               fprintf(ar,"%.2f ", cal);   /* Se escriben las calificaciones en
                                           ➥el archivo. */
            }
    }
    fclose(ar);
}
else
    printf("No se puede abrir el archivo");
}
```


EJEMPLO 9.6

Escribe un programa en **C** que lea de un archivo el número de alumnos (N), la matrícula y las cinco calificaciones de cada uno de ellos, y que imprima en pantalla la matrícula y el promedio de cada alumno.

Programa 9.6

```
#include <stdio.h>

/* Archivos con variables enteras y reales.
El programa lee datos de alumnos almacenados en un archivo y escribe la
➥matrícula y el promedio de cada alumno. */

void main(void)
{
int i, j, n, mat;
float cal, pro;
FILE *ar;
if ((ar = fopen("arc9.txt", "r")) != NULL)
{
     fscanf(ar, "%d", &n);             /* Se lee el valor de n. */
     for (i = 0; i < n; i++)
     {
          fscanf(ar, "%d", &mat);      /* Se lee la matrícula de cada alumno. */
          printf("%d\t", mat);
          pro = 0;
          for (j=0; j<5; j++)
          {
               fscanf(ar, "%f", &cal);      /* Se leen las cinco calificaciones
                                            ➥del alumno. */
                pro += cal;
          }
          printf("\t %.2f  ", pro / 5);   /* Se escribe el promedio de cada
                                           ➥alumno. */
          printf("\n");
```

```
    }
    fclose(ar);
}
else
   printf("No se puede abrir el archivo");
}
```

EJEMPLO 9.7

Escribe un programa en **C** similar al anterior, pero con la diferencia de que debe utilizar una función para realizar la lectura del archivo. Éste, por otra parte, se debe abrir y cerrar en el programa principal.

Programa 9.7

```
#include <stdio.h>

/* Archivos con variables enteras y reales.
El programa lee información de los alumnos de una escuela, almacenada en un
➥archivo. Utiliza una función para realizar la lectura, pero el archivo se abre
➥y cierra desde el programa principal. */

void promedio(FILE *);
/* Prototipo de función. Se pasa un archivo como parámetro. */

void main(void)
{
FILE *ar;
if ((ar = fopen("arc9.txt", "r")) !=  NULL)
{
    promedio(ar);        /* Se llama a la función promedio. Observe la forma
                         ➥como se pasa el archivo como parámetro. */
    fclose(ar);
}
else
    printf("No se puede abrir el archivo");
}

void promedio(FILE *ar1)   /* Observa la forma como se recibe el archivo. */
/* Esta función lee los datos de los alumnos desde un archivo, e imprime tanto
➥la matrícula como el promedio de cada alumno. */
{
int i, j, n, mat;
float pro, cal;
fscanf(ar1, "%d", &n);
for (i=0; i<n; i++)
{
```

```
            fscanf(ar1, "%d", &mat);
            printf("%d\t", mat);
             pro = 0;
            for (j = 0; j < 5; j++)
            {
              fscanf(ar1, "%f", &cal);
              pro += cal;
         }
         printf("\t %.2f  ", pro / 5);
         printf("\n");
    }
    }
```


9.3. Archivos de acceso directo

Los **archivos de acceso directo** almacenan los datos en bloques de longitud fija. Esta característica es muy importante porque nos permite tener acceso directamente a un bloque del archivo —siempre que conozcamos la posición en la que se encuentra— sin tener que recorrer el archivo en forma secuencial hasta localizar el bloque. Un bloque tiene siempre la misma longitud en términos de bytes y generalmente representa una estructura de datos tipo registro, conocido en **C** simplemente como estructura, aunque también puede almacenar un arreglo completo. Otra característica importante de los archivos de acceso directo es que podemos modificar con facilidad el archivo, ya que el programa cuenta con diversas funciones para ello. Recordemos que en los archivos de texto teníamos que generar un nuevo archivo cada vez que necesitábamos actualizarlo —modificarlo.

En el lenguaje **C** un **archivo de acceso directo** se abre y cierra de la siguiente forma:

```
    /* El conjunto de instrucciones muestra la sintaxis para abrir y cerrar un
➥archivo de acceso directo en el lenguaje de programación C. */

    . . .
    FILE *apuntador_archivo;
    apuntador_archivo = fopen (nombre_archivo, "tipo_archivo");
    if (apuntador_archivo != NULL)
    {
            proceso;                    /* Trabajo con el archivo.  */
            fclose(apuntador_archivo);
    }
    else
            printf("No se puede abrir el archivo");
    . . .
```

Observa que este conjunto de instrucciones es idéntico al que hemos presentado para abrir y cerrar archivos de texto. Los tipos de archivos, por otra parte, son idénticos a los que se mostraron en la tabla 9.1.

Analicemos a continuación diferentes ejemplos que nos permitirán comprender el manejo de archivos de acceso directo.

EJEMPLO 9.8

En el siguiente programa podemos observar la forma en que se abren, cierran y almacenan bloques —estructuras— en un archivo de acceso directo. Observa el uso de la función **fwrite** para escribir un bloque en el archivo.

Cada bloque en este programa representa una *estructura* que se utiliza para almacenar información sobre los alumnos de una escuela. Los campos de la estructura son los siguientes:

- Matrícula del alumno (entero).
- Nombre del alumno (cadena de caracteres).
- Carrera en la que está inscrito (entero).
- Promedio del alumno (real).

Observa que para indicar la carrera en la que está inscrito el alumno se utiliza: 1 para Economía, 2 para Contabilidad, 3 para Derecho, 4 para Ingeniería en Computación y 5 para Ingeniería Industrial.

Programa 9.8

```
#include <stdio.h>

/* Alumnos.
El programa almacena variables de tipo estructura alumno en un archivo. */

typedef struct                /* Declaración de la estructura alumno. */
{
   int matricula;
   char nombre[20];
   int carrera;
   float promedio;
}alumno;

void escribe(FILE *);         /* Prototipo de función. */

void main(void)
{
```

```
FILE *ar;
if ((ar = fopen ("ad1.dat", "w")) != NULL)
     escribe(ar);
else
     printf("\nEl archivo no se puede abrir");
fclose(ar);
}

void escribe(FILE *ap)
/* Esta función sirve para leer los datos de los alumnos utilizando una
➥estructura tipo alumno, que se almacenará posteriormente en un archivo. */
{
alumno alu;
int i = 0, r;
printf("\n¿Desea ingresar información sobre alumnos? (Sí-1  No-0):  ");
scanf("%d", &r);
while (r)
{
   i++;
   printf("Matrícula del alumno %d: ", i);
   scanf("%d", &alu.matricula);
   printf("Nombre del alumno %d: ", i);
   fflush(stdin);
   gets(alu.nombre);
   printf("Carrera del alumno %d: ", i);
   scanf("%d", &alu.carrera);
   printf("Promedio del alumno %d: ", i);
   scanf("%f", &alu.promedio);

   fwrite(&alu, sizeof(alumno), 1, ap);
   /* Observa que la función fwrite tiene cuatro argumentos. El primero se
   ➥utiliza para indicar la variable tipo estructura que se desea almacenar; el
   ➥segundo muestra el tamaño de esa variable en términos de bytes; el tercero
   ➥señala el número de variables que se leerán o almacenarán en el dispositivo
   ➥de almacenamiento secundario, y el último representa el apuntador al inicio de
   ➥la estructura FILE. */

   printf("\n¿Desea ingresar información sobre más alumnos? (Sí-1 No-0):  ");
   scanf("%d", &r);
}
}
```

EJEMPLO 9.9

Escribe un programa en **C** que, al recibir como dato el archivo de datos que se generó en el programa anterior, lea los registros del archivo y escriba la información de cada estructura en la pantalla de la computadora.

Dato: `ad1.dat`

Nota: En este programa podemos observar la forma en que se utiliza la instrucción **fread** para leer bloques de un archivo.

Programa 9.9

```
#include <stdio.h>

/* Alumnos.
El programa lee bloques —variables de tipo estructura alumno— de un archivo
➥de acceso directo. */

typedef struct {           /* Declaración de la estructura alumno. */
   int matricula;
   char nombre[20];
   int carrera;
   float promedio;
} alumno;

void lee(FILE *);    /* Prototipo de función. */

void main(void)
{
FILE *ar;
if ((ar = fopen ("ad1.dat", "r")) != NULL)
   escribe(ar);
else
   printf("\nEl archivo no se puede abrir");
fclose(ar);
}

void lee(FILE *ap)
/* Esta función se utiliza para leer bloques de un archivo de acceso directo. */
{
alumno alu;

fread(&alu, sizeof(alumno), 1, ap);
/* Observa que la función fread tiene los mismos argumentos que la función
➥fwrite del programa anterior. También es importante tomar nota de que cuando
➥tenemos que leer los registros de un archivo utilizando una estructura
➥repetitiva como el while, debemos realizar una primera lectura antes de
➥ingresar al ciclo y luego las siguientes dentro del ciclo, pero como última
➥instrucción del mismo. Esto se debe a que la lógica que siguen las
➥instrucciones fwrite y fread es moverse y leer o escribir, según sea el caso.
➥Si no lo hiciéramos de esta forma, terminaríamos escribiendo la información
➥del último registro dos veces. Vamos a realizar un ejemplo sencillo para
➥comprobar esto. */

while (!feof(ap))
{
```

```
        printf("\nMatrícula: %d", alu.matricula);
        printf("\tCarrera: %d", alu.carrera);
        printf("\tPromedio: %f\t ", alu.promedio);
        puts(alu.nombre);
        fread(&alu, sizeof(alumno), 1, ap);
    }
    }
```


EJEMPLO 9.10

Dado como dato el archivo de acceso directo generado en el programa 9.9, construya un programa en ***C*** que le pregunte al usuario el número de registro del alumno en el cuál se debe modificar el promedio, obtenga este valor, y modifique tanto el registro como el archivo correspondiente.

Dato: `ad1.dat`

Nota: En el siguiente programa podemos observar la forma en que se utiliza la instrucción `fseek`.

```
    #include <stdio.h>

    /* Alumnos.
    El programa pregunta al usuario el número de registro que desea
    ➥modificar, obtiene el nuevo promedio del alumno y modifica tanto el
    ➥registro como el archivo correspondiente. */

    typedef struct            /* Declaración de la estructura alumno. */
    {
      int matricula;
      char nombre[20];
      int carrera;
      float promedio;
    } alumno;

    void modifica(FILE *);    /* Prototipo de función. */

    void main(void)
    {
    FILE *ar;
    if ((ar = fopen ("ad1.dat", "r+")) != NULL)
       modifica(ar);
    else
       printf("\nEl archivo no se puede abrir");
    fclose(ar);
    }
```

```
void modifica(FILE *ap)
/* Esta función se utiliza para modificar el promedio de un alumno. */
{
int d;
alumno alu;
printf("\nIngrese el número de registro que desea modificar: ");
/* Observa que el lenguaje C almacena el primer registro en la
➥posición cero. Por lo tanto, si desea modificar el registro n,
➥debe buscarlo en la posición n-1. */
➥scanf("%d", &d);

fseek(ap, (d-1)*sizeof(alumno), 0);
/* Observa que la instrucción fseek tiene tres argumentos. El primero
➥indica que el apuntador se debe posicionar al inicio del FILE.
➥El segundo señala el número de bloques que debe moverse, en términos
➥de bytes, para llegar al registro correspondiente. Nota que el
➥primer registro ocupa la posición 0. Finalmente, el tercer argumento
➥muestra a partir de qué posición se debe mover el bloque de bytes:
➥se utiliza el 0 para indicar el inicio del archivo, 1 para expresar
➥que se debe mover a partir de la posición en la que actualmente se
➥encuentra y 2 para indicar que el movimiento es a partir del fin del
➥archivo. */

fread(&alu, sizeof(alumno), 1, ap);
/* Luego de posicionarnos en el registro que nos interesa, lo
➥leemos. */

printf("\nIngrese el promedio correcto del alumno: ");
scanf("%f", &alu.promedio);   /* Modificamos el registro con el
                              ➥nuevo promedio. */

fseek(ap, (d-1)*sizeof(alumno), 0);
/* Nos tenemos que posicionar nuevamente en el lugar correcto para
➥escribir el registro modificado. Observa que si no hacemos este
➥reposicionamiento escribiríamos el registro actualizado en la
➥siguiente posición. */

fwrite(&alu, sizeof(alumno), 1, ap);
}
```

EJEMPLO 9.11

Escribe un programa en **C** que, al recibir como dato el archivo de acceso directo ad5.dat, incremente 10% el salario de cada empleado que haya tenido ventas mayores a $1,000,000 durante el año. Los campos de la estructura que se utilizan para almacenar la información de los empleados son los siguientes:

- Clave del empleado (entero).
- Departamento en que trabaja (entero).

- Salario (real).
- Ventas (arreglo unidimensional de reales).

Dato: ad5.dat

Nota: En el siguiente programa podemos observar la forma en que se utilizan las instrucciones **`ftell`**, **`sizeof`** y **`rewind`.**

Programa 9.11

```
#include <stdio.h>

/* Incrementa salarios.
El programa incrementa el salario de los empleados de una empresa
—actualiza el archivo correspondiente— si sus ventas son superiores
al millón de pesos anuales. */

typedef struct              /* Declaración de la estructura empleado. */
{
   int clave;
   int departamento;
   float salario;
   float ventas[12];
}empleado;

void incrementa(FILE *);    /* Prototipo de función. */

void main(void)
{
FILE *ar;
if ((ar = fopen("ad5.dat", "r+")) != NULL)
/* El archivo se abre en la modalidad para leer y escribir. */
   incrementa(ar);
else
   printf("\nEl archivo no se puede abrir");

rewind(ar);
/* La función rewind se utiliza para posicionarnos en el inicio del
➥archivo cada vez que sea necesario. En este programa no tiene ninguna
➥utilidad, sólo se escribió para explicar su uso. */

fclose(ar);
}

void incrementa(FILE *ap)
/* Esta función se utiliza para incrementar el salario de todos aquellos
➥empleados que hayan tenido ventas anuales por más de $1,000,000.
➥Actualiza además el archivo correspondiente. */
```

```
{
int i, j, t;
float sum;
empleado emple;

t = sizeof(empleado);
/* La función sizeof se utiliza para conocer el tamaño de la estructura
➥empleado. */

fread(&emple, sizeof(empleado), 1, ap);    /* Se lee el primer registro
                                           ➥del archivo. */
while(!feof(ap))
{
   i = ftell(ap) / t;
   /* La función ftell se utiliza para conocer la posición de nuestro
   ➥apuntador en el archivo. La variable i nos proporciona en este caso
   ➥el tamaño de todos los bloques que existen debajo de nuestra
   ➥posición. Si conocemos el tamaño de cada bloque, entonces podemos
   ➥obtener el número de bloques que hay exactamente debajo de nuestra
   ➥posición. */

   sum = 0;
   for (j=0; j<12; j++)
        sum += emple.ventas[j];  /* Se calculan las ventas de cada
                                 ➥vendedor. */
   if (sum > 1000000)
   {
        emple.salario = emple.salario * 1.10;  /* Se incrementa el
                                               ➥salario. */
        fseek(ap, (i-1)*sizeof(empleado), 0);
        /* Nos posicionamos para escribir el registro actualizado. */
        fwrite(&emple, sizeof(empleado), 1, ap);
        fseek(ap, i*sizeof(empleado), 0);
        /* Nos posicionamos nuevamente para leer el siguiente registro.
        ➥Esta instrucción no debería ser necesaria, pero la función
        ➥fwrite se comporta a veces de manera inestable en algunos
        ➥compiladores de C. Para asegurarnos que siempre funcione
        ➥correctamente, realizamos este nuevo reposicionamiento. */

   }
   fread(&emple, sizeof(empleado), 1, ap);
}
}
```

Problemas resueltos

Problema PR9.1

Escribe un programa en **C** que, al recibir como dato el archivo de texto `libro.txt` que contiene el texto del primer capítulo de un libro, incorpore los siguientes caracteres a dicho archivo:`'Fin del texto'`. Utiliza funciones para el manejo de caracteres.

Dato: `libro.txt`

Programa 9.12

```
#include <stdio.h>

/* Incorpora caracteres.
  El programa agrega caracteres al archivo libro.txt. */

void main(void)
{
char p1;
FILE *ar;
ar = fopen("libro.txt", "a");
/* Se abre el archivo con la opción para incorporar caracteres. */
if (ar != NULL)
{
    while ((p1 = getchar()) != '\n')
       fputc(p1, ar);
    fclose(ar);
}
else
  printf("No se puede abrir el archivo");
}
```

Problema PR9.2

Construye un programa en **C** que, al recibir como datos una cadena de caracteres almacenada en el archivo de texto `arch.txt` y un caracter, determine cuántas veces se encuentra el caracter en el archivo.

Datos: `arch.txt`, `car` (donde `car` representa el caracter que se ingresa).

Programa 9.13

```
#include <stdio.h>

/* Cuenta caracteres.
El programa, al recibir como dato un archivo de texto y un caracter, cuenta
➥el número de veces que se encuentra el caracter en el archivo. */

int cuenta(char);                /* Prototipo de función. */

void main(void)
{
int res;
char car;
printf("\nIngrese el caracter que se va a buscar en el archivo: ");
car = getchar();
res = cuenta(car);
if (res != -1)
   printf("\n\nEl caracter %c se encuentra en el archivo %d veces", car, res);
else
   printf("No se pudo abrir el archivo");
}

int cuenta(char car)
/* Esta función determina cuántas veces se encuentra el caracter en el
➥archivo. */
{
int  res, con = 0;
char p;
FILE *ar;
if ((ar = fopen ("arc.txt", "r")) != NULL)    /* Se abre el archivo para
                                              ➥lectura. */
{
   while (!feof(ar))    /* Se trabaja con el archivo mientras no se llegue
                        ➥al fin de éste. */
   {
         p = getc(ar);
         if (p == car)   /* Se realiza la comparación de los caracteres. */
         con++;
   }
   fclose(ar);
   res = con;
}
else
   res = -1;
return (res);
}
```

Problema PR 9.3

Escribe un programa en **C** que, al recibir como dato el archivo de texto `arc5.txt` formado por cadenas de caracteres, determine el número de letras minúsculas y mayúsculas que existen en el archivo. Utiliza solamente funciones que lean caracteres, no cadenas de caracteres.

Dato: `arc.txt`

Programa 9.14

```
#include <stdio.h>
#include <ctype.h>

/* Letras minúsculas y mayúsculas.
El programa, al recibir como dato un archivo formado por cadenas de caracteres,
➥determina el número de letras minúsculas y mayúsculas que hay en el archivo. */

void minymay(FILE *);                  /* Prototipo de función. */
/* Observa que esta función va a recibir un archivo como parámetro. */

void main(void)
{
char p;
FILE *ar;
if ((ar = fopen("arc5.txt", "r")) != NULL)
{
   minymay(ar);
   /* Se llama a la función minymay. Se pasa el archivo ar como parámetro. */
   fclose(ar);
}
else
   printf("No se pudo abrir el archivo");
}

void minymay(FILE *arc)
/* Esta función cuenta el número de minúsculas y mayúsculas que hay en el
➥archivo arc. */
{
int min = 0, may = 0;
char p;
while (!feof(arc))
{
   p = fgetc(arc);   /* Se utiliza la función fgetc() para leer caracteres
                     ➥del archivo. */
   if (islower(p))
      min++;
```

```
   else
      if (isupper(p))
          may++;
}
printf("\nNúmero de minúsculas: %d", min);
printf("\nNúmero de mayúsculas: %d", may);
}
```

Problema PR 9.4

Escribe un programa en **C** que resuelva el problema anterior, pero ahora utilizando funciones que lean cadenas de caracteres.

Dato: `arc.txt`

Programa 9.15

```
#include <stdio.h>
#include <ctype.h>

/* Letras minúsculas y mayúsculas.
El programa, al recibir como dato un archivo formado por cadenas de
➥caracteres, determina el número de letras minúsculas y mayúsculas que hay
➥en el archivo. */

void minymay(FILE *);             /* Prototipo de función. */

void main(void)
{
FILE *ap;
if ((ap = fopen ("arc.txt", "r")) != NULL)
{
      minymay(ap);
    fclose(ap);
}
else
    printf("No se puede abrir el archivo");
}

void minymay(FILE *ap1)
/* Esta función se utiliza para leer cadenas de caracteres de un archivo
➥y contar el número de letras minúsculas y mayúsculas que existen en el
➥archivo. */
{
char cad[30];
int i, mi = 0, ma = 0;
```

```
while (!feof(ap1))
{
    fgets(cad,30,ap1);
    /* Se utiliza la función fgets() para leer cadenas de caracteres del
    ➥archivo. */
    i=0;
    while (cad[i] != '\0')
    {
       if (islower(cad[i]))
           mi++;
       else
           if (isupper(cad[i]))
               ma++;
       i++;
    }
}
printf("\n\nNúmero de letras minúsculas: %d", mi);
printf("\nNúmero de letras mayúsculas: %d", ma);
}
```


Problema PR 9.5

Escribe un programa en **C** que, al recibir como dato el archivo de texto `arc2.txt` compuesto por cadenas de caracteres, que pueden contener números reales, obtenga la suma y el promedio de dichos números. Por ejemplo, el archivo se podría presentar de la siguiente forma:

```
109.209as.309.sasa409.50
abc208.108.208.308.40
307.107.207.307.40
```

Dato: `arc2.txt`

Programa 9.16

```
#include <stdio.h>
#include <stdlib.h>

/* Suma reales.
El programa lee cadenas de caracteres de un archivo, detecta aquellas que
➥comienzan con números, los suma y calcula el promedio de los mismos. */

void sumypro(FILE *);                /* Prototipo de función. */
```

```
void main(void)
{
FILE *ap;
if ((ap=fopen("arc2.txt", "r")) != NULL)
{
   sumypro(ap);
   /* Se llama a la función sumypro. Se pasa el archivo ap como parámetro. */
   fclose(ap);
}
else
   printf("No se puede abrir el archivo");
}

void sumypro(FILE *ap1)
/* Esta función lee cadenas de caracteres de un archivo, detecta aquellas
➥que comienzan con números, y obtiene la suma y el promedio de dichos
➥números. */
{
char cad[30];
int i = 0;
float sum = 0.0, r;
while (!feof (ap1))
{
   fgets(cad,30,ap1);            /* Se lee la cadena del archivo. */
   r = atof(cad);
   /* Recuerda que la función atof convierte una cadena de caracteres que
   ➥contiene números reales a un valor de tipo double. Si la cadena comienza
   ➥con otro caracter o no contiene números, regresa 0 o el valor queda
   ➥indefinido. */
   if (r)
   {
       i++;
       sum += r;
   }
}
printf("\nSuma: %.2f", sum);
if (i)  /* Si el valor de i es distinto de cero, calcula el promedio. */
   printf("\nPromedio: %.2f", sum/i);
}
```

Problema PR 9.6

Desarrolla un programa en **C** que, al recibir como dato el archivo de texto `arc.txt` compuesto por cadenas de caracteres en las que puede aparecer la palabra méxico escrita con minúsculas, cada vez que localice la palabra `méxico` en una cadena, la reemplace por su forma correcta —la primera letra con mayúsculas— y escriba la nueva cadena en el archivo `arc1.txt`. Observa que la palabra `méxico` se puede

encontrar varias veces en una misma cadena. Por ejemplo, el archivo se podría presentar de la siguiente forma:

```
es méxico lindo méxico
méxico es maravilloso méxico
me gusta la gente de méxico
méxico méxico méxico
```

y debe quedar en el archivo `arc2.txt` de la siguiente forma:

```
es México lindo México
México es maravilloso México
me gusta la gente de México
México México México
```

Dato: `arc.txt`

Programa 9.17

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

/* Reemplaza palabras.
El programa lee cadenas de caracteres de un archivo y cada que vez que
➥encuentra la palabra México escrita en forma incorrecta —la primera con
➥minúscula— la reemplaza por su forma correcta y escribe la cadena en otro
➥archivo. */

void cambia(FILE *, FILE *);
/* Prototipo de función. Se pasan dos archivos como parámetros. */

void main(void)
{
FILE *ar;
FILE *ap;
ar = fopen("arc.txt", "r");   /* Se abre el archivo arc.txt para lectura. */
ap = fopen("arc1.txt", "w");  /* Se abre el archivo arc1.txt para escritura. */
if ((ar != NULL) && (ap != NULL))
{
    cambia(ar, ap);
    fclose(ar);
    fclose(ap);
}
else
    printf("No se pueden abrir los archivos");
}
```

```
void cambia(FILE *ap1, FILE *ap2)
{
/* Esta función reemplaza en la cadena de caracteres la palabra méxico escrita
➥con minúsculas —la primera letra— por su forma correcta y escribe la cadena
➥de caracteres en un nuevo archivo. */
int i, j, k;
char cad[30], *cad1="", *cad2="", aux[30];
while (!feof(ap1))
{
   fgets(cad, 30, ap1);
   strcpy(cad1, cad);
      cad2 = strstr(cad1, "méxico");     /* Localiza la subcadena méxico
                                          ➥en cad1. */
   while (cad2!=NULL)
   {
   cad2[0]='M';           /* Reemplaza la letra minúscula por la mayúscula. */
   i = strlen(cad1);
   j = strlen(cad2);
   k = i - j;    /* En k se almacena la diferencia de las longitudes de
                 ➥las cadenas
                    cad1 y cad2. */
      if (k)
      {
          strncpy(aux, cad1, k);
          /* Se copia la subcadena de k caracteres de cad1 a aux —desde el
          ➥inicio de
          cad1 hasta el caracter anterior a méxico. */
          aux[k] = '\0';
          strcat(aux, cad2);
          strcpy(cad1, aux);
      }
      else
           strcpy(cad1, cad2);
      cad2 = strstr(cad1, "méxico");
   }
   fputs(cad1, ap2);     /* Se escribe la cadena correcta en el archivo ap2. */
}
}
```

Problema PR 9.7

Construye un programa en **C** que, al recibir como datos los archivos ordenados `arc9.dat` y `arc10.dat` que contienen información sobre la matrícula y tres calificaciones de los alumnos de una escuela, mezcle los dos archivos anteriores considerando el orden ascendente de las matrículas y forme un tercer archivo, `arc11.dat`, ordenado también lógicamente en función de las matrículas. Cabe

destacar que la matrícula de cada alumno es un valor entero y las tres calificaciones son valores reales. Por ejemplo, los archivos se podrían presentar de la siguiente forma:

arc9.dat

```
55   6.7   7.8   7.8
67   7.2   8.8   7.8
85   7.7   8.7   8.9
93   8.7   9.9   9.6
```

arc10.dat

```
31   8.7   6.7    8.9
45   7.8   7.6    5.8
61   7.8   9.0    9.9
82   8.8   9.9    9.7
96   8.9   9.1    9.9
99   9.3   9.6    9.8
```

La mezcla de los mismos debe quedar como se muestra a continuación:

arc11.dat

```
31   8.7   6.7   8.9
45   7.8   7.6   5.8
55   6.7   7.8   7.8
61   7.8   9.0   9.9
67   7.2   8.8   7.8
82   8.8   9.9   9.7
85   7.7   8.7   8.9
93   8.7   9.9   9.6
96   8.9   9.1   9.9
99   9.3   9.6   9.8
```

Datos: `arc9.dat, arc10.dat`

Programa 9.18

```
include <stdio.h>

/* Mezcla.
El programa mezcla, respetando el orden, dos archivos que se encuentran
➥ordenados en forma ascendente considerando la matrícula de los alumnos. */

void mezcla(FILE *, FILE *, FILE *);          /* Prototipo de función. */
```

```
void main(void)
{
FILE *ar, *ar1, *ar2;
ar = fopen("arc9.dat", "r");
ar1 = fopen("arc10.dat", "r");
ar2 = fopen("arc11.dat", "w");
if (((ar != NULL) && (ar1 != NULL)) && (ar2 != NULL))
{
   mezcla(ar, ar1, ar2);
   fclose(ar);
   fclose(ar1);
   fclose(ar2);
}
else
   printf("No se pueden abrir los archivos");
}

void mezcla(FILE *ar, FILE *ar1, FILE *ar2)
/* Esta función mezcla, respetando el orden, dos archivos que se encuentran
➥ordenados en función de la matrícula. */
{
int i, mat, mat1, b=1, b1=1;
float ca[3], ca1[3], cal;
while ((((!feof(ar)) ¦¦ !b) && ((!feof(ar1)) ¦¦ !b1))
{
   if (b)   /* Si la bandera b está encendida, se lee del archivo ar la
            ➥matrícula y las tres calificaciones del alumno. */
   {
          fscanf(ar, "%d", &mat);
          for (i=0; i<3; i++)
              fscanf(ar, "%f", &ca[i]);
          b = 0;
   }
   if (b1)  /* Si la bandera b1 está encendida, se lee del archivo ar
            ➥la matrícula y las tres calificaciones del alumno. */
   {
          fscanf(ar1, "%d", &mat1);
          for (i=0; i<3; i++)
              fscanf(ar1, "%f", &ca1[i]);
          b1 = 0;
   }
   if (mat < mat1)
   {
          fprintf(ar2, "%d\t", mat);
          for (i=0; i<3; i++)
              fprintf(ar2, "%f\t", ca[i]);
          fputs("\n", ar2);
          b = 1;
```

```
    }
    else
    {
            fprintf(ar2, "%d\t", mat1);
            for (i=0; i<3; i++)
                fprintf(ar2, "%f\t", ca1[i]);
            fputs("\n", ar2);
            b1 = 1;
    }
}
if (!b)
{
            fprintf(ar2, "%d\t", mat);
            for (i=0; i<3; i++)
                fprintf(ar2, "%f\t", ca[i]);
            fputs("\n", ar2);
            while (!feof(ar))
            {
                fscanf(ar, "%d", &mat);
                fprintf(ar2, "%d\t", mat);
                for (i=0; i<3; i++)
                {
                   fscanf(ar, "%f", &cal);
                   fprintf(ar2, "%f\t", cal);
                }
                fputs("\n", ar2);
            }
}
if(!b1)
{
            fprintf(ar2, "%d\t", mat1);
            for (i=0; i<3; i++)
                fprintf(ar2, "%f\t", ca1[i]);
            fputs("\n", ar2);
            while (!feof(ar1))
            {
               fscanf(ar1, "%d", &mat1);
                fprintf(ar2, "%d\t", mat1);
                for (i=0; i<3; i++)
                {
                  fscanf(ar1, "%f", &cal);
                  fprintf(ar2, "%f\t", cal);
                }
                fputs("\n", ar2);
            }
}
}
```

Problema PR 9.8

Escribe un programa en **C** que, al recibir como dato el archivo de acceso directo `ad5.dat` que contiene registros de los alumnos de una escuela, ordenados de mayor a menor en función de su matrícula, genere un nuevo archivo pero ahora ordenado de menor a mayor, también en función de la matrícula. Los campos de las estructuras almacenadas en el archivo son los siguientes:

- Matrícula del alumno (entero).
- Nombre del alumno (cadena de caracteres).
- Carrera en la que está inscrito (entero).
- Promedio del alumno (real).

Dato: `ad5.dat`

Programa 9.19

```
#include <stdio.h>

/* Ordena de menor a mayor.
El programa ordena de menor a mayor en función de la matrícula, creando un
➥nuevo archivo, un archivo de acceso directo compuesto por estructuras y
➥ordenado de mayor a menor. */

typedef struct                   /* Declaración de la estructura alumno. */
{
   int matricula;
   char nombre[20];
   int carrera;
   float promedio;
} alumno;

void ordena(FILE *, FILE *);    /* Prototipo de función. */.

void main(void)
{
FILE *ar1, *ar2;
ar1 = fopen("ad5.dat", "r");
ar2 = fopen("ad6.dat", "w");
if ((ar1 != NULL) && (ar2 != NULL))
    ordena(ar1, ar2);
else
    printf("\nEl o los archivos no se pudieron abrir");
fclose(ar1);
fclose(ar2);
}
```

```
void ordena(FILE *ap1, FILE *ap2)
/* Esta función ordena de menor a mayor un archivo compuesto por estructuras,
➥en función de su matrícula, y genera un nuevo archivo. */
{
alumno alu;
int t, n, i;
t = sizeof(alumno);
fseek (ap1, sizeof(alumno), 2);
n = (ftell(ap1) / t) - 1;
/* Se obtiene el número de registros que componen el archivo. El valor de n,
➥a su vez, se utilizará para posicionarnos en el archivo. */
rewind(ap1);
for (i = (n-1); i >= 0; i--)           /* Se utiliza un ciclo descendente. */
{
    fseek(ap1, i * sizeof(alumno), 0);
    fread(&alu, sizeof(alumno), 1, ap1);
    fwrite(&alu, sizeof(alumno), 1, ap2);
}
}
```


Problema PR 9.9

En el archivo de acceso directo `esc.dat` se almacena la información de los alumnos de una escuela utilizando estructuras. Se registra la siguiente información de cada alumno:

- Matrícula del alumno (entero).
- Nombre y apellido (cadena de caracteres).
- Materias y promedios (arreglo unidimensional de estructura).
    - Materia (cadena de caracteres).
    - Promedio (real).

Escribe un programa en **C** que obtenga lo siguiente:

a) La matrícula y el promedio general de cada alumno.

b) Las matrículas de los alumnos cuyas calificaciones en la tercera materia sean mayores a 9.

c) El promedio general de la materia 4.

Dato: `esc.dat`

Programa 9.20

```
#include <stdio.h>

/* Escuela.
El programa, al recibir como dato un archivo de acceso directo que contiene
➥información de los alumnos de una escuela, genera información estadística
➥importante. */

typedef struct                  /* Declaración de la estructura matcal. */
{
    char materia[20];
    int calificacion;
} matcal;

typedef struct                  /* Declaración de la estructura alumno. */
{
    int matricula;
    char nombre[20];
    matcal cal[5];
    /* Observa que un campo de esta estructura es a su vez estructura. */
} alumno;

void F1(FILE *);
void F2(FILE *);                /* Prototipos de funciones. */
float F3FILE *);

void main(void)
{
float pro;
FILE *ap;
if ((ap = fopen("esc.dat", "r")) != NULL)
{
    F1(ap);
    F2(ap);
    pro = F3(ap);
    printf("\n\nPROMEDIO GENERAL MATERIA 4: %f", pro);
}
else
    printf("\nEl archivo no se puede abrir");
fclose(ap);
}

void F1(FILE *ap)
/* La función escribe la matrícula y el promedio general de cada alumno. */
{
alumno alu;
int j;
float sum, pro;
printf("\nMATRÍCULA y PROMEDIOS");
```

```
    fread (&alu, sizeof(alumno), 1, ap);
    while (!feof(ap))
    {
        printf("\nMatrícula: %d", alu.matricula);
        sum = 0.0;
        for (j=0; j<5; j++)
            sum += alu.cal[j].calificacion;
        pro = sum / 5;
        printf("\tPromedio: %f", pro);
        fread(&alu, sizeof(alumno), 1, ap);
    }
    }

    void F2(FILE *ap)
    /* La función escribe la matrícula de los alumnos cuya calificación en la
    ➥tercera materia es mayor a 9. */
    {
    alumno alu;
    int j;
    rewind(ap);
    printf("\n\nALUMNOS CON CALIFICACIÓN > 9 EN MATERIA 3");
    fread(&alu, sizeof(alumno), 1, ap);
    while (!feof(ap))
    {
        if (alu.cal[2].calificacion > 9)
            printf("\nMatrícula del alumno: %d", alu.matricula);
        fread(&alu, sizeof(alumno), 1, ap);
    }
    }

    float F3(FILE *ap)
    /* Esta función obtiene el promedio general de la materia 4. */
    {
    alumno alu;
    int i = 0;
    float sum = 0, pro;
    rewind(ap);
    fread(&alu, sizeof(alumno), 1, ap);
    while (!feof(ap))
    {
        i++;
        sum += alu.cal[3].calificacion;
        fread(&alu, sizeof(alumno), 1, ap);
    }
    pro = (float)sum / i;
    return (pro);
    }
```

Problema PR 9.10

En un archivo de acceso directo se almacena la información de los alumnos que presentaron el examen de admisión a una universidad privada de la Ciudad de México. Se almacena la siguiente información de cada alumno en una estructura:

- Clave del alumno (entero).
- Nombre del alumno (cadena de caracteres).
- Carrera universitaria (entero).
- Promedio de preparatoria (real).
- Calificación examen de admisión (real).
- Teléfono (cadena de caracteres).

Observa que para indicar la carrera en la que se quiere inscribir el alumno, se utiliza: 1 para Economía, 2 para Contabilidad, 3 para Derecho, 4 para Ingeniería en Computación y 5 para Ingeniería Industrial.

Escribe un programa en **C** que realice lo siguiente:

a) Obtenga el promedio general del examen de admisión.

b) Genere un archivo de alumnos admitidos por cada carrera. Se consideran admitidos aquellos alumnos que sacaron por lo menos 1300 puntos en el examen de admisión y su promedio de preparatoria sea mayor o igual a 8, o bien aquellos que tengan un promedio mayor o igual a 7 pero que en el examen hayan sacado un puntaje superior a 1399 puntos.

c) Obtenga el promedio del examen de admisión de los alumnos admitidos en cada carrera.

Dato: `alu.dat`

Programa 9.21

```
#include <stdio.h>

/* Examen de admisión.
El programa, al recibir como dato un archivo de acceso directo que contiene
➥información sobre los alumnos que presentaron el examen de admisión a una
➥universidad, genera información importante para el Departamento de Control
➥Escolar. */
```

```
typedef struct                  /* Declaración de la estructura alumno. */
{
    int clave;
    char nombre[20];
    int carrera;
    float promedio;
    float examen;
    char telefono[12];
} alumno;

float F1(FILE *);
void F2(FILE *,FILE *, FILE *, FILE *, FILE *, FILE *);
void F3(FILE *, FILE *, FILE *, FILE *, FILE *);  /* Prototipos de funciones. */

void main(void)
{
float pro;
FILE *ap, *c1, *c2, *c3, *c4, *c5;
ap = fopen("alu1.dat", "r");
/* Observa que los archivos car1.dat, car2.dat, car3.dat, car4.dat y car5.dat
➥se abren en la modalidad para escribir y leer. */
c1 = fopen("car1.dat", "w+");
c2 = fopen("car2.dat", "w+");
c3 = fopen("car3.dat", "w+");
c4 = fopen("car4.dat", "w+");
c5 = fopen("car5.dat", "w+");
if ((ap!=NULL) && (c1!=NULL) && (c2!=NULL) && (c3!=NULL) &&
    (c4!=NULL) && (c5!=NULL))
{
    pro = F1(ap);
    printf("\nPROMEDIO EXAMEN DE ADMISIÓN: %.2f", pro);
    F2(ap, c1, c2, c3, c4, c5);
    F3(c1, c2, c3, c4, c5);
}
else
    printf("\nEl o los archivos no se pudieron abrir");
fclose(ap);
fclose(c1);
fclose(c2);
fclose(c3);
fclose(c4);
fclose(c5);
}

float F1(FILE *ap)
/* Esta función obtiene el promedio del examen de admisión. */
{
alumno alu;
float sum = 0, pro;
int i = 0;
```

```
fread(&alu, sizeof(alumno), 1, ap);
while (!feof(ap))
{
    i++;
    sum += alu.examen;
    fread(&alu, sizeof(alumno), 1, ap);
}
pro = sum / i;
return (pro);
}

void F2(FILE *ap, FILE *c1, FILE *c2, FILE *c3, FILE *c4, FILE *c5)
/* Esta función genera un archivo de los alumnos admitidos en cada una de
➥las carreras de la universidad. */
{
alumno alu;
rewind(ap);
fread(&alu, sizeof(alumno), 1, ap);
while (!feof(ap))
{
    /* Se analiza si el candidato es admitido a la universidad. */
    if (((alu.examen >= 1300) && (alu.promedio >= 8)) || ((alu.examen >=
    ➥1400) &&
      (alu.promedio >= 7)))
    {
      switch (alu.carrera)
      {
           case 1:  fwrite(&alu, sizeof(alumno), 1, c1);
              break;
           case 2:  fwrite(&alu, sizeof(alumno), 1, c2);
              break;
           case 3:  fwrite(&alu, sizeof(alumno), 1, c3);
              break;
           case 4:  fwrite(&alu, sizeof(alumno), 1, c4);
              break;
           case 5:  fwrite(&alu, sizeof(alumno), 1, c5);
              break;
      }
   }
   fread(&alu, sizeof(alumno), 1, ap);
}
}

void F3 (FILE *c1, FILE *c2, FILE *c3, FILE *c4, FILE *c5)
/* Esta función se utiliza para obtener el promedio que consiguieron los
➥alumnos admitidos en cada una de las carreras. */
{
alumno alu;
float cal[5], sum;
int i, j;
```

```
    i = 0;
    sum = 0;
    rewind(c1);     /* Es importante posicionarse al inicio del archivo, pues
    ➥de lo contrario se generaría un error al ejecutar el programa. */
    fread(&alu, sizeof(alumno), 1, c1);
    while (!feof(c1))
    {
        i++;
        sum += alu.examen;
        fread(&alu, sizeof(alumno), 1, c1);
    }
    if (i)
        cal[0] = (sum / i);
    else
        cal[0] = 0;

    rewind(c2);
    sum = 0;
    i = 0;
    fread(&alu, sizeof(alumno), 1, c2);
    while (!feof(c2))
    {
        i++;
        sum += alu.examen;
        fread(&alu, sizeof(alumno), 1, c2);
    }
    if (i)
        cal[1] = (sum / i);
    else
        cal[1] = 0;

    rewind(c3);
    sum = 0;
    i = 0;
    fread(&alu, sizeof(alumno), 1, c3);
    while (!feof(c3))
    {
        i++;
        sum += alu.examen;
        fread(&alu, sizeof(alumno), 1, c3);
    }
    if (i)
        cal[2] = (sum / i);
    else
        cal[2] = 0;

    rewind(c4);
    sum = 0;
    i = 0;
```

```
fread(&alu, sizeof(alumno), 1, c4);
while (!feof(c4))
{
    i++;
    sum += alu.examen;
    fread(&alu, sizeof(alumno), 1, c4);
}
if (i)
    cal[3] = (sum / i);
else
    cal[3] = 0;

rewind(c5);
sum = 0;
i = 0;
fread(&alu, sizeof(alumno), 1, c5);
while (!feof(c5))
{
    i++;
    sum += alu.examen;
    fread(&alu, sizeof(alumno), 1, c5);
}
if (i)
    cal[4] = (sum / i);
else
    cal[4] = 0;
/* Se imprimen los promedios de los alumnos admitidos en cada carrera. */
for (i=0; i<5; i++)
  printf("\nPromedio carrera %d: %.2f", i+1, cal[i]);
}
```

Problemas suplementarios

Problema PS9.1

Escribe un programa en **C** que, al recibir como dato un archivo de texto formado por cadenas de caracteres, determine la longitud de la cadena más grande sin utilizar la función `strlen`.

Dato: `arc1.txt`

Problema PS9.2

Escribe un programa en **C** que, al recibir como dato el archivo `arc.txt` compuesto por cadenas de caracteres, calcule el número de cada una de las vocales que se encuentra en el archivo. Por ejemplo, si el archivo contiene las siguientes cadenas de caracteres:

```
México es la novena economía del mundo,
pero tiene más pobres que la mayoría de los países europeos
con macroeconomías peores que la de México.
```


El programa debe dar los siguientes resultados:

```
a: 11
e: 19
i:  7
o: 17
u:  4
```

Dato: `arc.txt`

Problema PS9.3

Escribe un programa en **C** que, al recibir como dato un archivo de texto compuesto por cadenas de caracteres, determine cuántas palabras hay en el archivo. Cada palabra se separa por medio de un espacio en blanco. Por ejemplo, si el archivo es el siguiente:

```
sa sa sa yacusá yacusá
le mando le mando le mando al maestro
```

El programa debe escribir que hay 13 palabras.

Dato: `arc.txt`

Problema PS9.4

Escribe un programa en **C** que, al recibir como datos una cadena de caracteres y el archivo de texto `arc2.txt`, compuesto también por cadenas de caracteres, determine

cuántas veces se encuentra la primera cadena de caracteres en el archivo. Por ejemplo, si el archivo es el siguiente:

arc2.txt

```
sasaasassassssassas
ssssaaabbbsassbsasbbbassssss
sssssaaaaaaassssssasssassbbbsbsb
sssssabsabsbbbbsbabsas
```

y la cadena de caracteres es: sas

el programa debe regresar: `10`

Datos: `cad[50] y arc2.dat` (donde `cad` representa la cadena de 50 caracteres como máximo)

Problema PS9.5

Escribe un programa en **C** que, al recibir como dato un archivo de texto compuesto por cadenas de caracteres, forme un nuevo archivo en el cual las cadenas aparezcan intercambiadas: *la última con la primera, la penúltima con la segunda, y así sucesivamente*.

Dato: `arc.txt`

Problema PS9.6

Escribe un programa en **C** que, al recibir como dato el archivo `doc.dat` compuesto por cadenas de caracteres, revise la ortografía del mismo y verifique si se cumplen las siguientes reglas ortográficas: *antes de b va m, no n*; *antes de p va m, no n*, y finalmente, *antes de v va n, no m*.

Dato: `doc.dat`

Problema PS 9.7

Escribe un programa en **C** que, al recibir como dato el archivo de acceso directo `ad5.dat` que contiene los registros de los alumnos de una escuela —algunos están

repetidos— ordenados de mayor a menor en función de su matrícula, genere un nuevo archivo pero sin registros repetidos. Los campos de las estructuras almacenadas en el archivo son los siguientes:

- Matrícula del alumno (entero).
- Nombre del alumno (cadena de caracteres).
- Carrera en la que está inscrito (entero).
- Promedio del alumno (real).

Dato: `ad5.dat`

Problema PS9.8

Construye un programa en **C** que, al recibir como dato el archivo `arc.dat` que contiene información sobre la matrícula y tres calificaciones de los alumnos de una escuela, ordene ese archivo en forma ascendente considerando la matrícula del alumno y genere un nuevo archivo `arc1.dat`. Cabe destacar que la matrícula del alumno es un valor entero y las tres calificaciones son valores reales. Por ejemplo, si el archivo se presenta de la siguiente forma:

```
arc.dat
51   8.7   6.7   8.9
15   7.8   7.6   5.8
11   7.8   9.0   9.9
32   8.8   9.9   9.7
96   8.9   9.1   9.9
29   9.3   9.6   9.8
```

después de la ordenación debe quedar de la siguiente forma:

```
arc1.dat
11   7.8   9.0   9.9
15   7.8   7.6   5.8
29   9.3   9.6   9.8
32   8.8   9.9   9.7
51   8.7   6.7   8.9
96   8.9   9.1   9.9
```

Dato: `arc.dat`

Problema PS9.9

Una comercializadora que distribuye pinturas e impermeabilizantes como principales productos, ubicada en la ciudad de Monterrey, en México, almacena en un archivo de acceso directo, *ordenado de menor a mayor en función de la clave*, toda la información relativa a sus productos:

- Clave del producto (entero).
- Nombre del producto (cadena de caracteres).
- Existencia (entero).
- Precio unitario (real).

Escribe un programa en **C** que construya los siguientes módulos:

a) Ventas. El módulo registra la venta de diferentes productos a un cliente —tienda. Obtiene el total de la venta y actualiza el inventario correspondiente. El fin de datos para la venta de un cliente es 0.

b) Reabastecimiento. Este módulo permite actualizar el inventario al incorporar productos —cantidades— al mismo. El fin de datos es 0.

c) Nuevos Productos. El módulo permite incorporar nuevos productos al inventario. El fin de datos es 0. Observa que éstos se deberán insertar en la posición que les corresponde de acuerdo con su clave. Es probable que deba generar un nuevo archivo para resolver este módulo.

d) Inventario. El módulo permite imprimir el inventario completo.

Dato: `com.dat`

Problema PS9.10

En un hospital de Quito, en Ecuador, almacenan la información de sus pacientes en un archivo de acceso directo, `pacientes.dat`, que se encuentra ordenado en forma ascendente en función de la clave del paciente. Los datos de cada hospitalizado se almacenan en una *estructura*, cuyos campos son los siguientes:

- Clave del paciente (entero).
- Nombre y apellido (cadena de caracteres).
- Edad (entero).

- Sexo (caracter).
- Condición (entero).
- Domicilio (estructura).
    - Calle (cadena de caracteres).
    - Número (entero).
    - Colonia (cadena de caracteres).
    - Código Postal (cadena de caracteres).
    - Ciudad (cadena de caracteres).
    - Teléfono (cadena de caracteres).

Escribe un programa en **C** que obtenga lo siguiente:

a) El porcentaje tanto de hombres como de mujeres registrados en el hospital.

b) El número de pacientes de cada una de las categorías de condición.

c) El número de pacientes que hay en cada categoría de edad: 0-9, 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99, >=100.

Nota: Observa que `Condición` se refiere al estado de salud en que ingresó el paciente. Los valores que toma `Condición` van de 1 a 5, y 5 representa el máximo grado de gravedad.

Dato: `pacientes.dat`

ÍNDICE

A

Algoritmo, 2
Alias, creación, 293
Apuntadores, 146
  y arreglos, 181
Archivos, 334
  de acceso directo, 343
  de datos, 333, 336
  de texto, 334
  tipos de, 336
Arreglo, 176
  finito, 176
  bidimensional, 214, 266
  búsqueda binaria, 198
  búsqueda secuencial de arreglos desordenados, 195
  búsqueda secuencial en arreglos en forma creciente, 196
  componentes, 176
  homogéneo, 176
  ordenado, 176
  unidimensional, 266
Arreglos, 175
  bidimensionales, 213
    declaración de, 215
  de más de dos dimensiones, 220
  multidimensionales, 213
  tridimensionales, declaración de, 221
  unidimensionales, 175, 176
    declaración, 177

B

Bloque de asignación, 11

C

Cadena de caracteres, 266
Campo, 288
Caracter, 254
Caracteres, 253
  cadenas, 253, 257
  de control, 23
  y cadenas de caracteres, 253
    estructurados, 253
    simples, 253
Ciclo, 89
Coma, operador, 16
Componentes, 214
Constantes, 9
Construcción de diagramas de flujo, 18
Creación de sinónimos o alias, 293
ctype.h, funciones de la biblioteca, 255

D

Datos, archivos de, 336
Determinismo, 4
Diagrama de flujo
  esquematización, 5
  símbolos utilizados, 5

E

Estructura
  repetitiva
    `do-while`, 103
    `for`, 90
    `while`, 97
  selectiva
    doble `if - else`, 54
    múltiple `switch`, 58
    simple `if`, 50
  `while`, 90
Estructuras, 288
  anidadas, 295
  con arreglos, 298
  declaración, 289
  finita, 288
  heterogénea, 288
  selectivas
    algorítmicas, 49
    en cascada, 64
    repetitivas, 89
  y uniones, 287
Expresiones lógicas o booleanas, 15

F

Finitud, 4
Formato
  de escritura de las variables, 25
  de variables, 25
Función, 137
Funciones de la biblioteca math.h, 34

H

Heterogéneos, 287

I

Identificador, 9
Índices, 214

L

Lenguaje C, palabras reservadas, 9
Lenguaje de programación C, 22

M

Modificaciones al símbolo %, 26

N

Nombre único, 288

O

Operador coma, 16
Operadores, 12
  aritméticos, 12
    simplificados, 13
  de incremento (++) y decremento (– –), 14
  lógicos, 16
  prioridades, 17
  relacionales, 15
Ordenación
  por inserción directa (forma creciente), 200
  por selección directa (forma creciente), 202

P

Palabras reservadas, 9
Parámetros
  por referencia, 138, 146
  por valor, 138, 146
Paso de funciones como parámetros, 152
Precisión, 4
Programa, 22

R

Reducción de problemas, 137
Registros, 288

S

stdlib.h, funciones de la biblioteca, 261
string.h, funciones de la biblioteca, 264

T

Texto, 334
  acceso directo, 334
  archivos de texto, 334
  binario, 334
  método de acceso secuencial, 334
  secuencial, 334, 335
Tipos
    de archivos, 336
    de datos, 8
    estructurados, 175
    simples, 8, 175

U

Unión, 301
Uniones, 301

V

Variables, 10
  estáticas, 138, 139
  formato, 25
  globales, 138, 139
  locales, 13