

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



4243
0054
E2



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 x 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

PEARSON
Educación
®

LITOGRÁFICA INGRAMEX, S.A.
CENTENO No. 162-1
COL. GRANJAS ESMERALDA
09810 MÉXICO, D.F.

2006

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 <i>if</i>	50
2.3. La estructura selectiva doble <i>if-else</i>	54
2.4. La estructura selectiva múltiple <i>switch</i>	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 <i>for</i>	90
3.3. La estructura repetitiva <i>while</i>	97
3.4. La estructura repetitiva <i>do-while</i>	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.
 $1\frac{1}{2}$ cucharadas de orégano seco.
185 g de piloncillo rallado.

Relleno:

1 cucharada de aceite.
 $\frac{1}{2}$ cebolla finamente picada.
2 dientes de ajo finamente picados.
 $\frac{1}{2}$ 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.
1/2 cucharada de pimienta recién molida.

1

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 enfrie 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. Quitales 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; sumérgeto en el huevo batido e inmediatamente ponlo en el aceite. Frié 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.

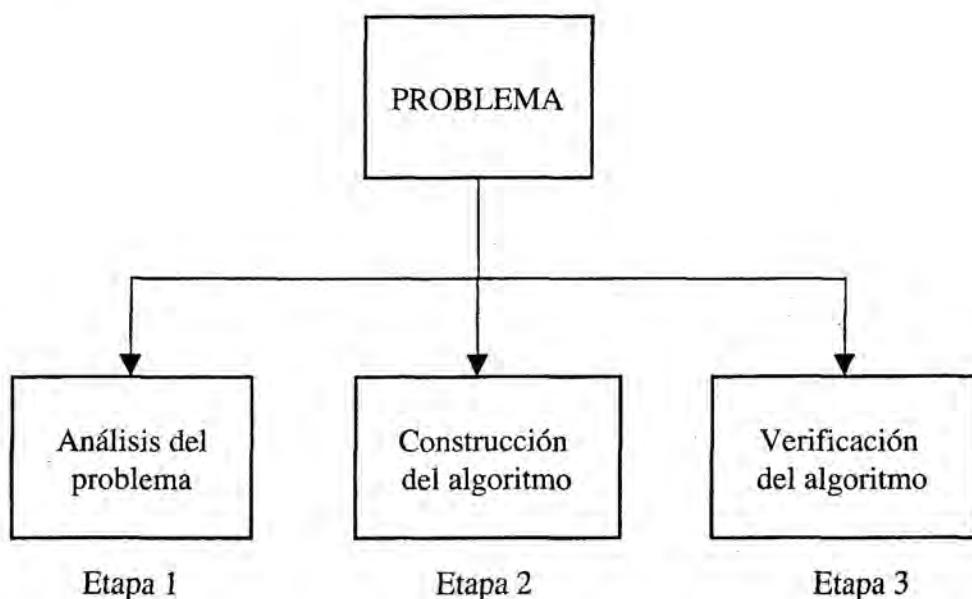


FIGURA 1.1
Etapas para solucionar un problema.

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).

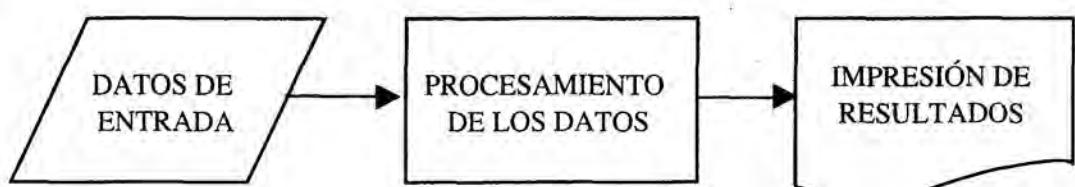


FIGURA 1.2
Módulos o secciones de un algoritmo.

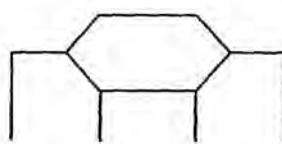
1.2 Diagramas de flujo

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 <i>inicio</i> y el <i>fin</i> del diagrama de flujo.
	Se utiliza para introducir los datos de entrada. Expresa <i>lectura</i> .
	Representa un <i>proceso</i> . En su interior se colocan asignaciones, operaciones aritméticas, cambios de valor de celdas en memoria, etc.
	Se utiliza para representar una <i>decisión</i> . 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 <i>if</i> e <i>if-else</i> que estudiaremos en el siguiente capítulo, así como en las estructuras repetitivas <i>for</i> , <i>while</i> y <i>do-while</i> , 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.

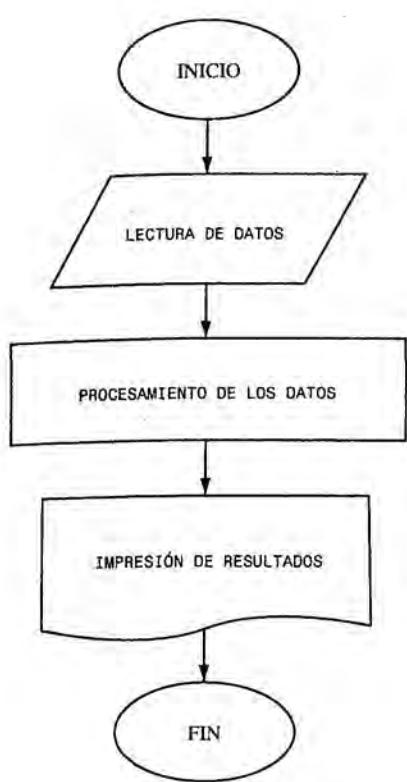


Representa conexión entre páginas diferentes.



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

<i>Tipo de datos en C</i>	<i>Descripción</i>	<i>Rango</i>
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 carácter 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, carácter, 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 carácter 't' a la variable ca1. */
    ca2 = '?';          /* Asignación del carácter '?' 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	<code>x = 4.5 + 3;</code> <code>v = 4.5 + 3;</code>	<code>x = 7</code> <code>v = 7.5</code>
-	Resta	<code>x = 4.5 - 3;</code> <code>v = 4.5 - 3;</code>	<code>x = 1</code> <code>v = 1.5</code>
*	Multiplicación	<code>x = 4.5 * 3;</code> <code>v = 4.5 * 3;</code> <code>v = 4 * 3;</code>	<code>x = 12</code> <code>v = 13.5</code> <code>v = 12.0</code>
/	División	<code>x = 4 / 3;</code> <code>x = 4.0 / 3.0;</code> <code>v = 4 / 3;</code> <code>v = 4.0 / 3;</code> <code>v = (float) 4 / 3;</code> <code>v = ((float) 5 + 3) / 6;</code>	<code>x = 1</code> <code>x = 1</code> <code>v = 1.0</code> <code>v = 1.33</code> <code>v = 1.33</code> <code>v = 1.33</code>
%	Módulo(residuo)	<code>x = 15 % 2;</code> <code>v = (15 % 2) / 2;</code> <code>v = ((float) (15 % 2)) / 2;</code>	<code>x = 1</code> <code>v = 0.0</code> <code>v = 0.5</code>

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
$\ast, /, \%$	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
$+$	$+ =$	$x = 6;$ $y = 4;$ $x += 5;$ $x += y;$	$x = 6;$ $y = 4;$ $x = x + 5;$ $x = x + y;$	$x = 6$ $y = 4$ $x = 11$ $x = 15$
	$- =$	$x = 10;$ $y = 5;$ $x -= 3;$ $x -= y;$	$x = 10;$ $y = 5;$ $x = x - 3;$ $x = x - y;$	$x = 10$ $y = 5$ $x = 7$ $x = 2$
	$* =$	$x = 5;$ $y = 3;$ $x *= 4;$ $x *= y;$	$x = 5;$ $y = 3;$ $x = x * 4;$ $x = x * y;$	$x = 5$ $y = 3$ $x = 20$ $x = 60$

continúa

TABLA 1.6. Continuación

<i>Operador aritmético</i>	<i>Forma simplificada de uso</i>	<i>Ejemplos</i>	<i>Equivalencia</i>	<i>Resultados</i>
/	/=	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

<i>Operador</i>	<i>Operación</i>	<i>Ejemplos</i>	<i>Resultados</i>
++	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

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

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

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

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

TABLA 1.10. Tabla de verdad de los operadores lógicos

<i>P</i>	<i>Q</i>	<i>(! P)</i>	<i>(! Q)</i>	<i>(P Q)</i>	<i>(P && Q)</i>
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
$x = (v = 3, v * 5);$	$v = 3$ $x = v * 5;$	$v = 3$ $x = 15$
$x = (v += 5, v \% 3);$	$v = v + 5;$ $x = v \% 3;$	$v = 8$ $x = 2$
$x = (y = (15 > 10), z = (2 >= y), y \&& z);$	$y = (15 > 10);$ $z = (2 >= y);$ $x = y \&& z;$	$y = 1$ $z = 1$ $x = 1$
$x = (y = (! (7 > 15)), z = (35 > 40) \&& y,$ $(!(y \&& z)));$	$y = (! (7 > 15));$ $z = (35 > 40) \&& y;$ $x = (!(y \&& z));$	$y = 1$ $z = 0$ $x = 1$

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

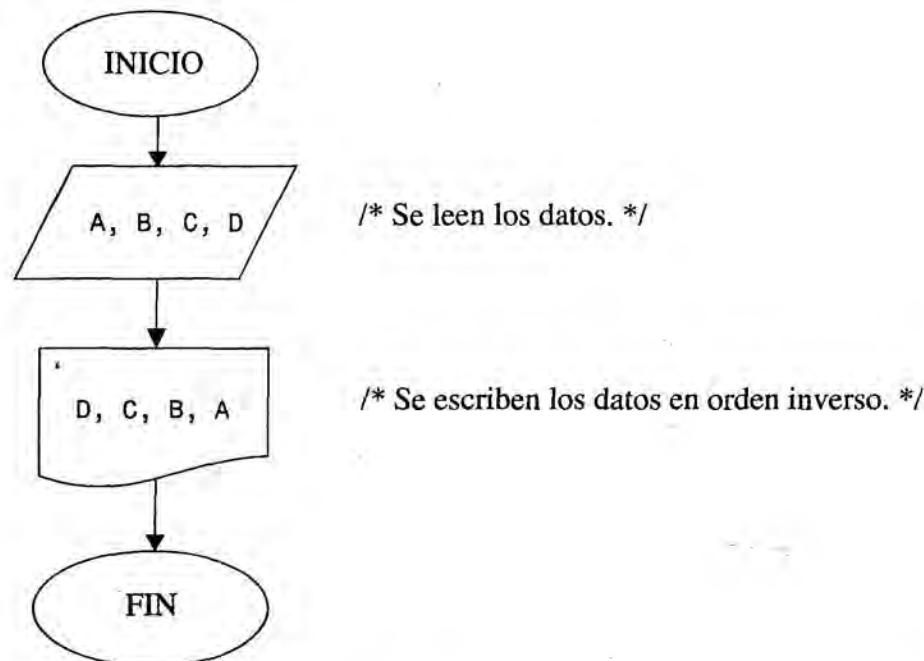
Un **diagrama de flujo** muestra, como señalamos anteriormente, la esquematización gráfica de un algoritmo. Su correcta construcción es importante, porque a partir del mismo se debe escribir el programa en un lenguaje de programación determinado. Es nuestro interés que comiences a desarrollar habilidad 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



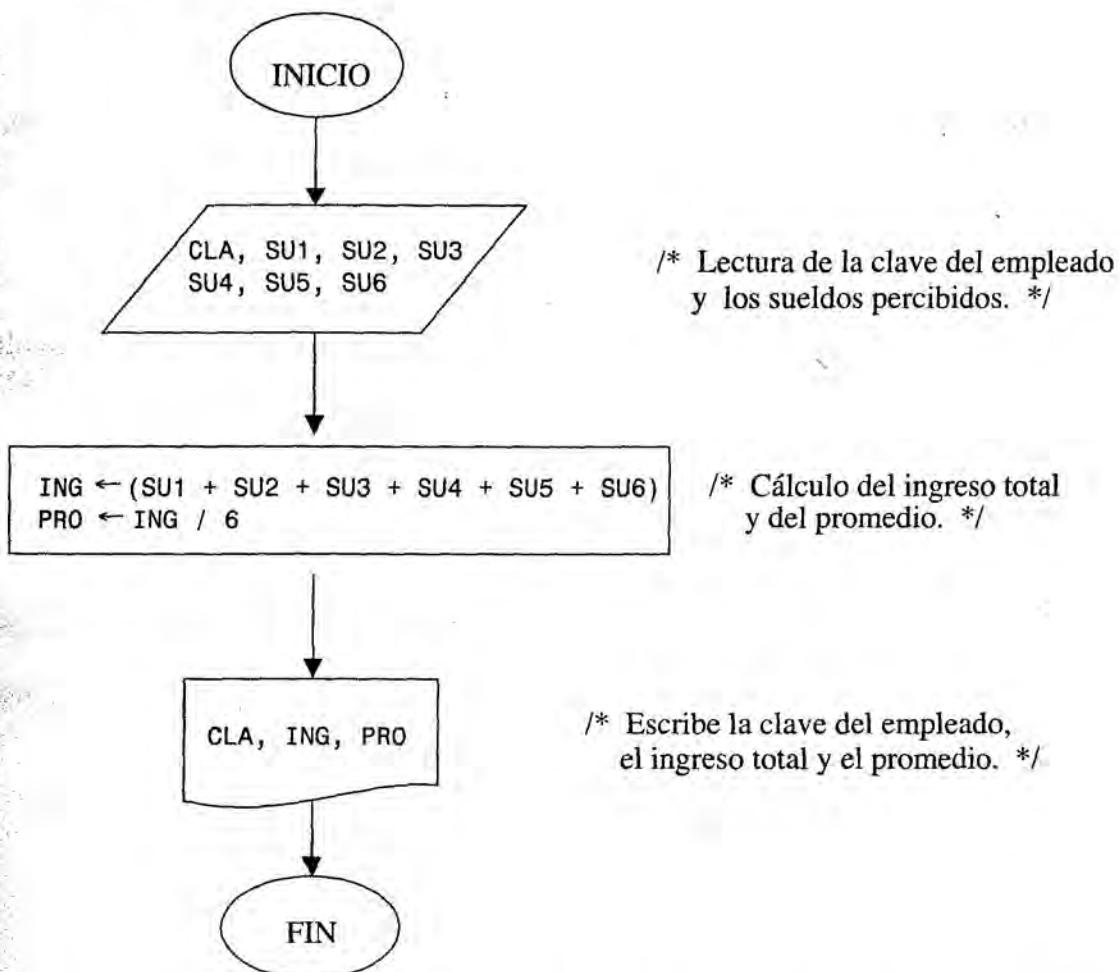
Observa que si se ingresan los datos: 10, 20, 30 y 40, la impresión produce lo siguiente: 40, 30, 20 y 10.

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.

<i>Corrida</i>	<i>Datos</i>	<i>Resultados</i>								
		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.

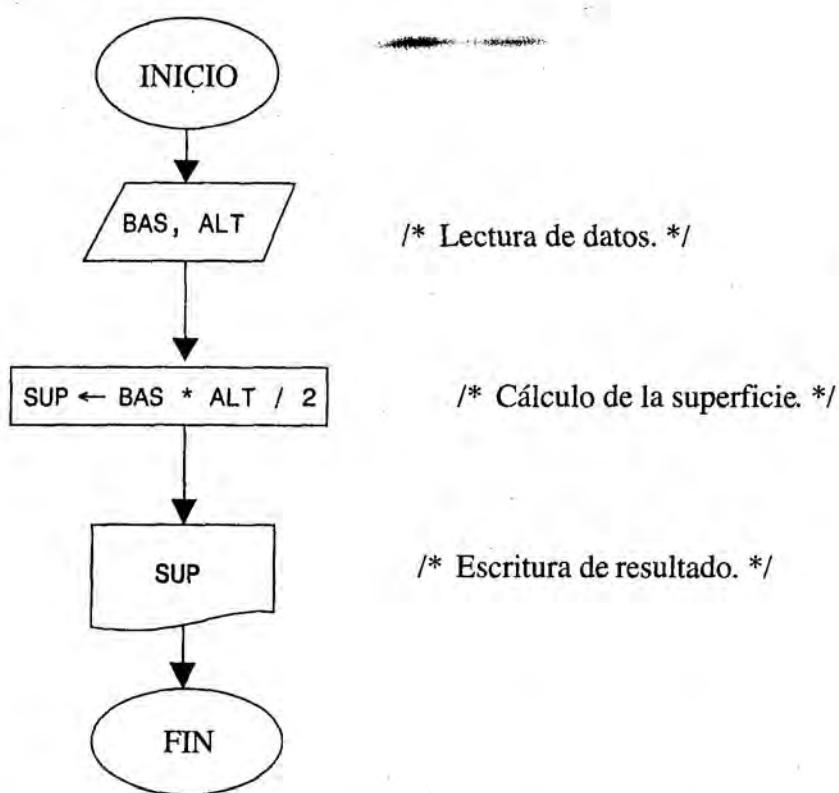
Datos: BAS, ALT

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:

$$\text{Superficie} = (\text{base} * \text{altura}) / 2$$

Fórmula 1.1

Diagrama de flujo 1.3

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	
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 deseas 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

<i>Carácter de control</i>	<i>Explicación</i>
\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

1

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 (<code>unsigned int</code>).
%d %i	Escribe enteros de 2 bytes (<code>int</code>).
/ld	Imprime enteros de largo alcance (<code>long</code>).
%f	Escribe reales de 4 bytes (<code>float</code>).
%lf	Escribe reales de doble precisión, 8 bytes (<code>double</code>).
%e	Imprime en forma exponencial.
%g	Imprime en %f o %e en función del tamaño del número.
%c	Escribe un carácter de un byte (<code>char</code>).
%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", &CLA, &SU1, &SU2, &SU3, &SU4, &SU5, &SU6);
    ING = (SU1 + SU2 + SU3 + SU4 + SU5 + SU6);
    PRO = ING / 6;
    printf("\n %d  %.2f  %.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: %.2f", SUP);
```

1

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.

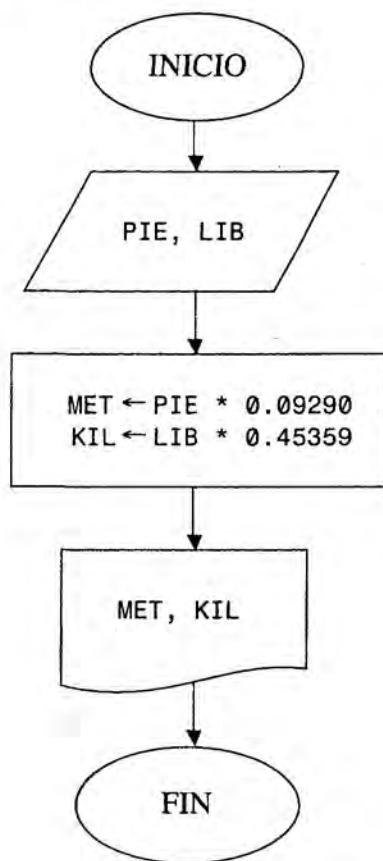
Datos: PIE, LIB

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: $\pi = 3.141592$.

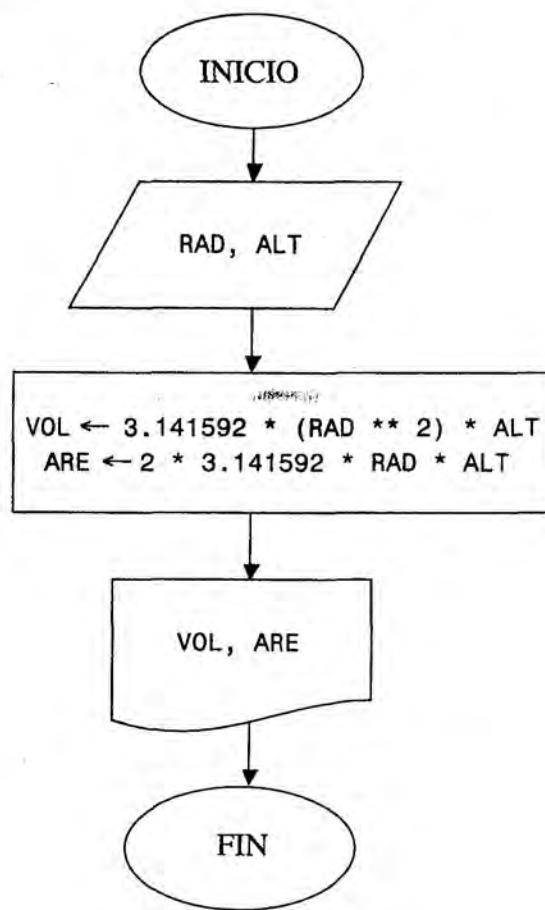
- La superficie del cilindro la calculamos como:

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

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

Problema PR1.5

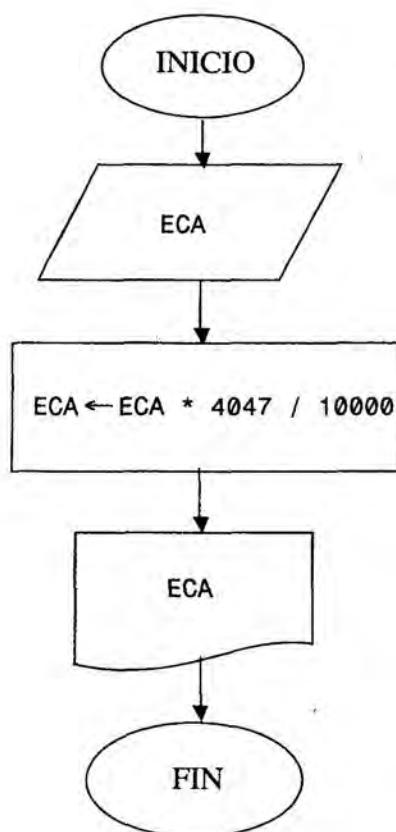
Una persona compró una estancia en un país sudamericano. La extensión de la estancia está especificada en acres. Construye un diagrama de flujo que, al recibir como dato la extensión de la estancia en *acres*, calcule e imprima la extensión de la misma en hectáreas.

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 $10,000 \text{ m}^2$.

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:

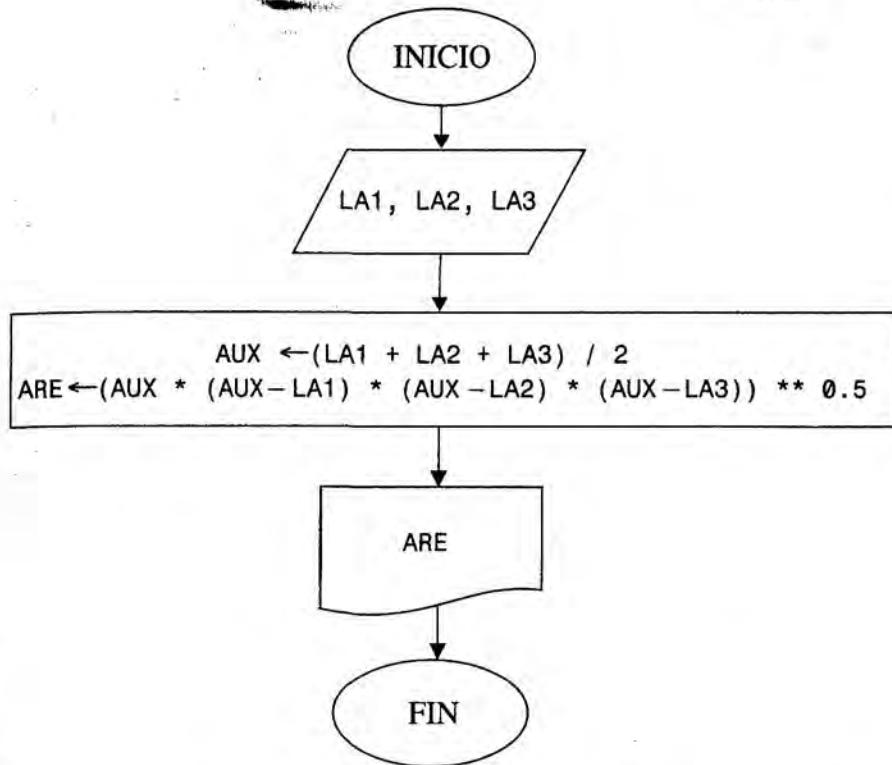
$$\text{AREA} = \sqrt{\text{AUX} * (\text{AUX} - \text{LAI}) * (\text{AUX} - \text{LA2}) * (\text{AUX} - \text{LA3})}$$

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

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: %.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).

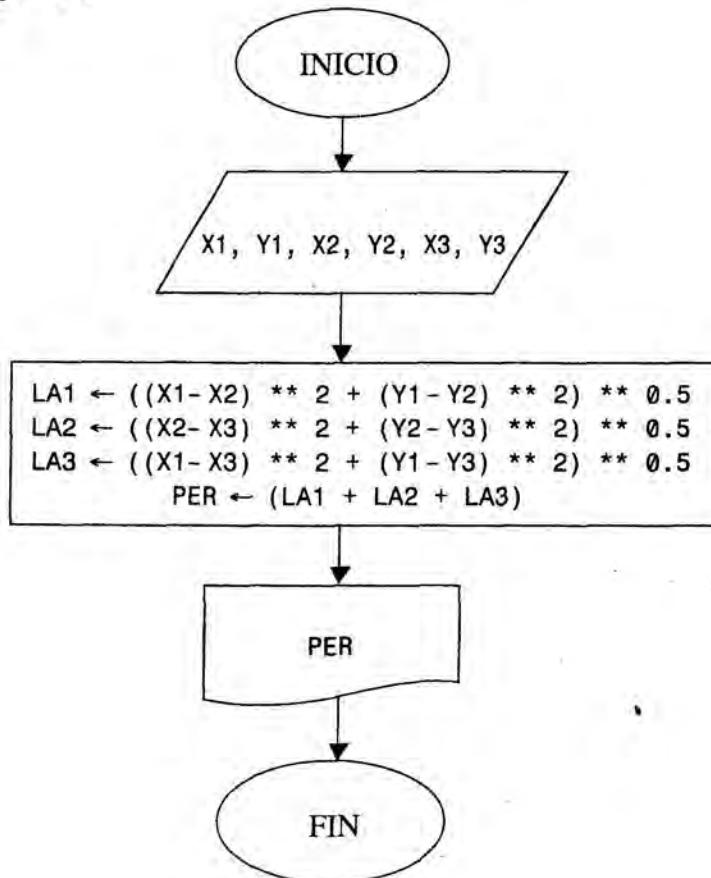
Consideraciones:

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

$$\text{DIS} = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

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.

1

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: %.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);  
})
```

1

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 → 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
XX
XXXX
```

Problema PS1.8

1

En las olimpiadas 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:

$$TSE = MIN * 60 + SEG + CEN / 100$$

Fórmula 1.8

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

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

Fórmula 1.9

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

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

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:

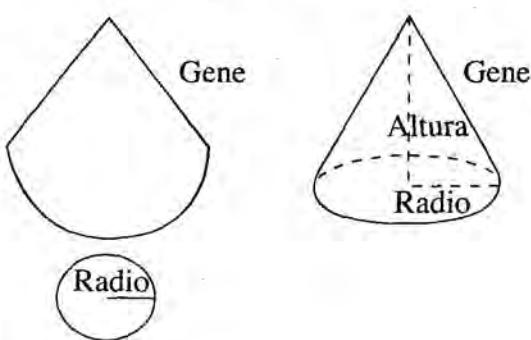


FIGURA 1.4

- 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:

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

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:

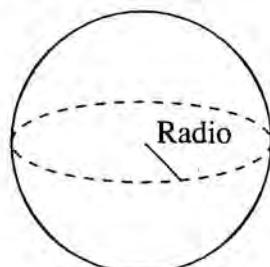


FIGURA 1.5.

- 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

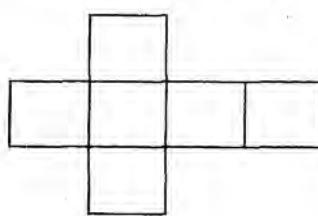
Problema PS1.13

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.

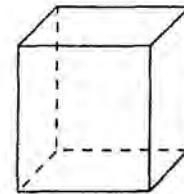
Dato: LAD (variable de tipo real que representa el lado del hexaedro).

Consideraciones:

- Un hexaedro o cubo tiene la siguiente forma:



L



L

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:

$$\text{Área total} = 6 * L^2$$

1

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:

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

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:

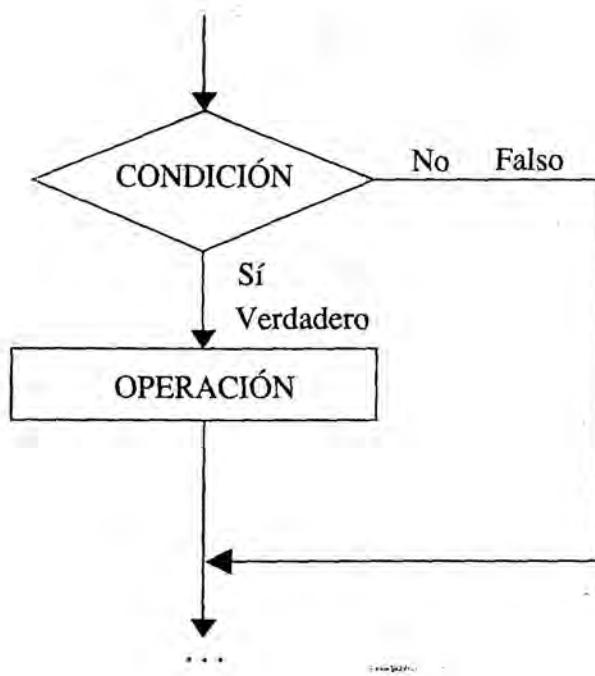


FIGURA 2.1

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>;
    . . .
```

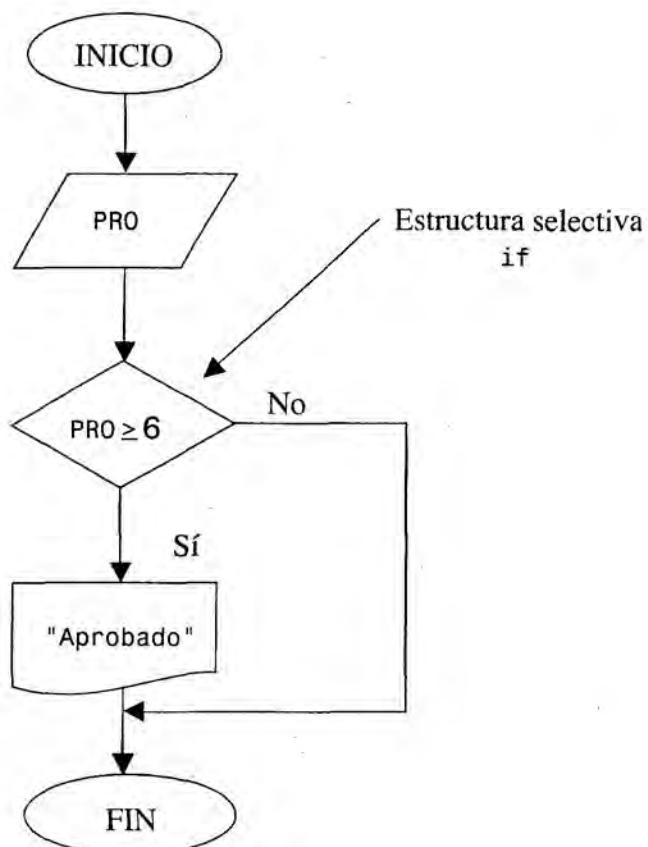
2

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. */

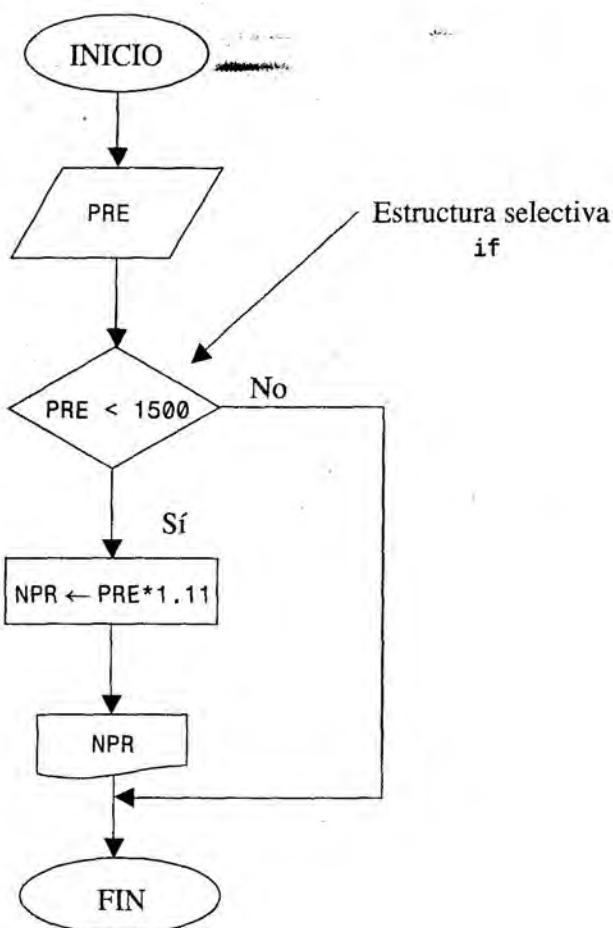
void main(void)
{
float PRO;
printf("ingrese el promedio del alumno: ");
scanf("%f", &PRO);
if (PRO >= 6)
    printf("\nAprobado");
}
```

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



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.

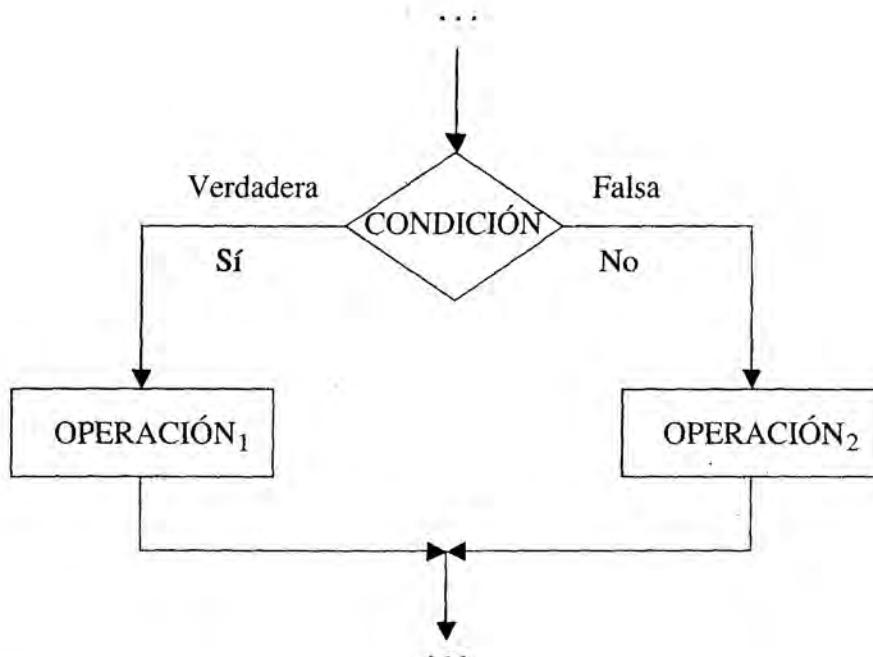


FIGURA 2.2
Estructura selectiva doble **if-else**

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. */
<condición>
    <operación1>;
    <operación2>;
```

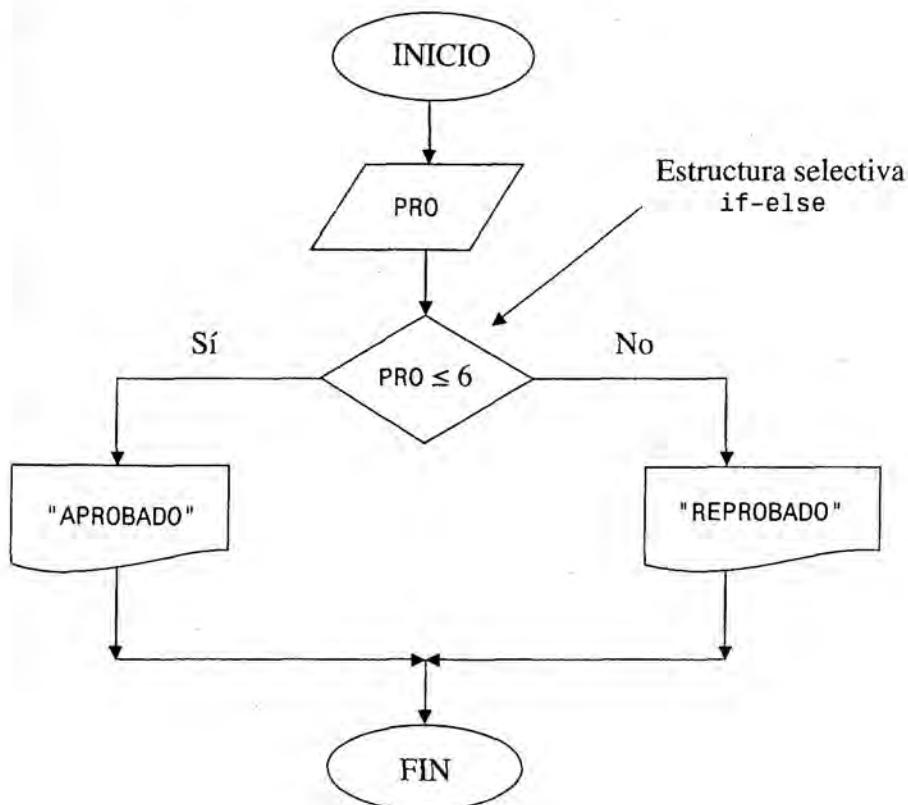
2

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. */

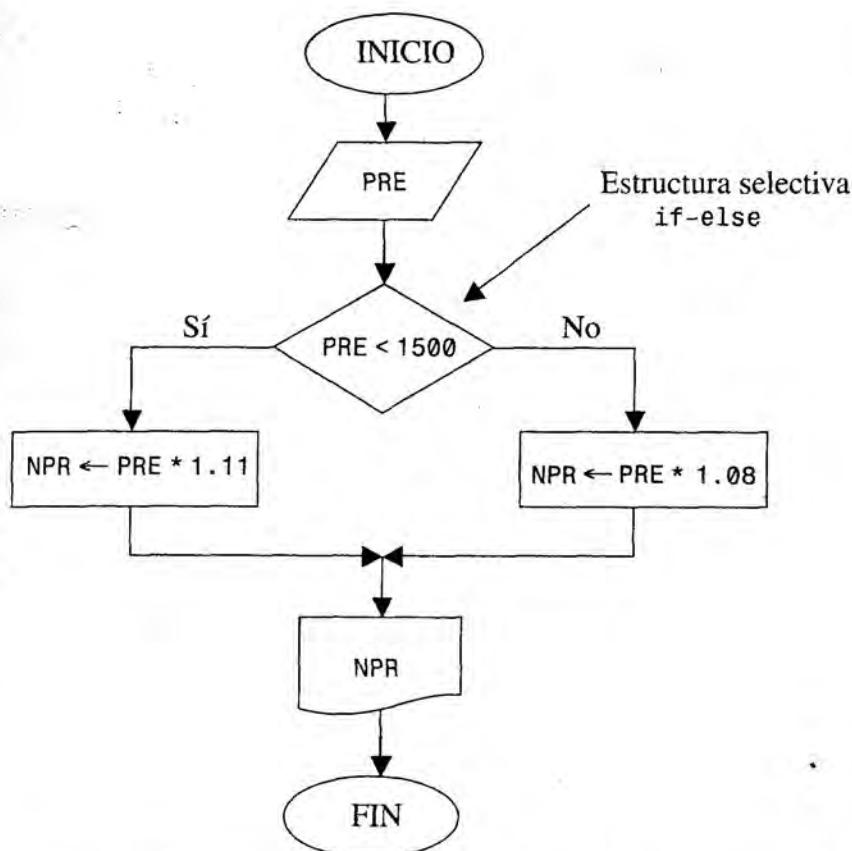
void main(void)
{
float PRO;
printf("Ingrese el promedio del alumno: ");
scanf("%f", &PRO);
if (PRO >= 6.0)
    printf("\nAprobado");
else
    printf("\nReprobado");
}
```

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



2

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.

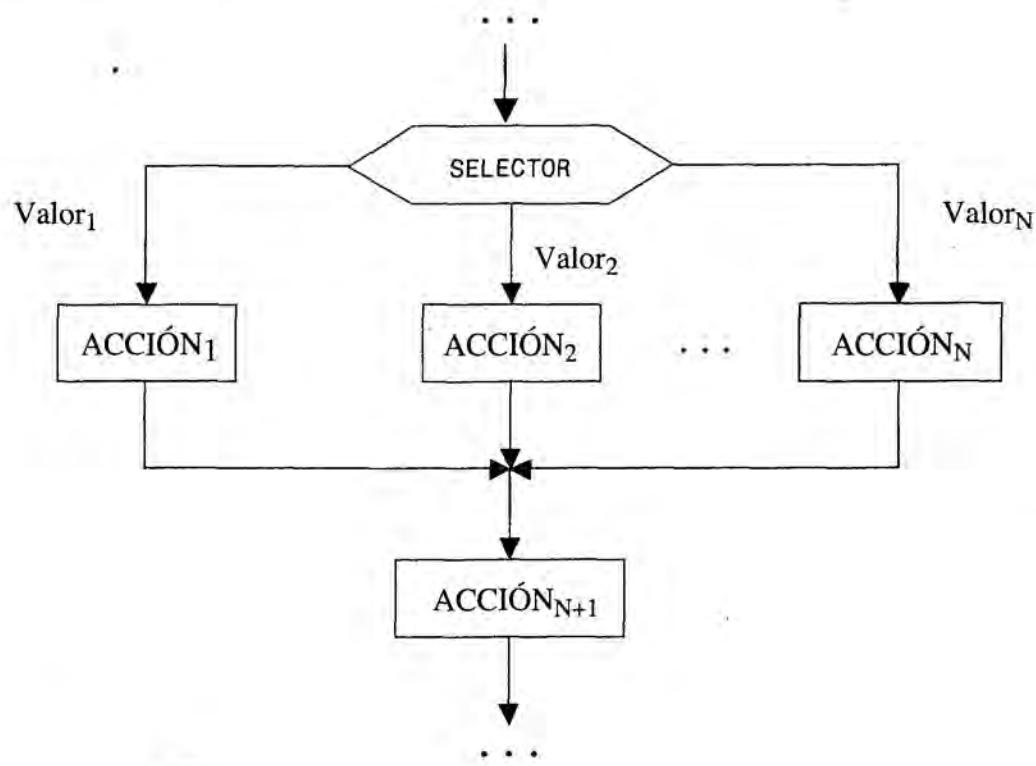


FIGURA 2.3
Estructura selectiva múltiple **switch**

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>;
    case <valorN> : <acciónN>;
        break;

    acciónN+1;
}
```

2

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*.

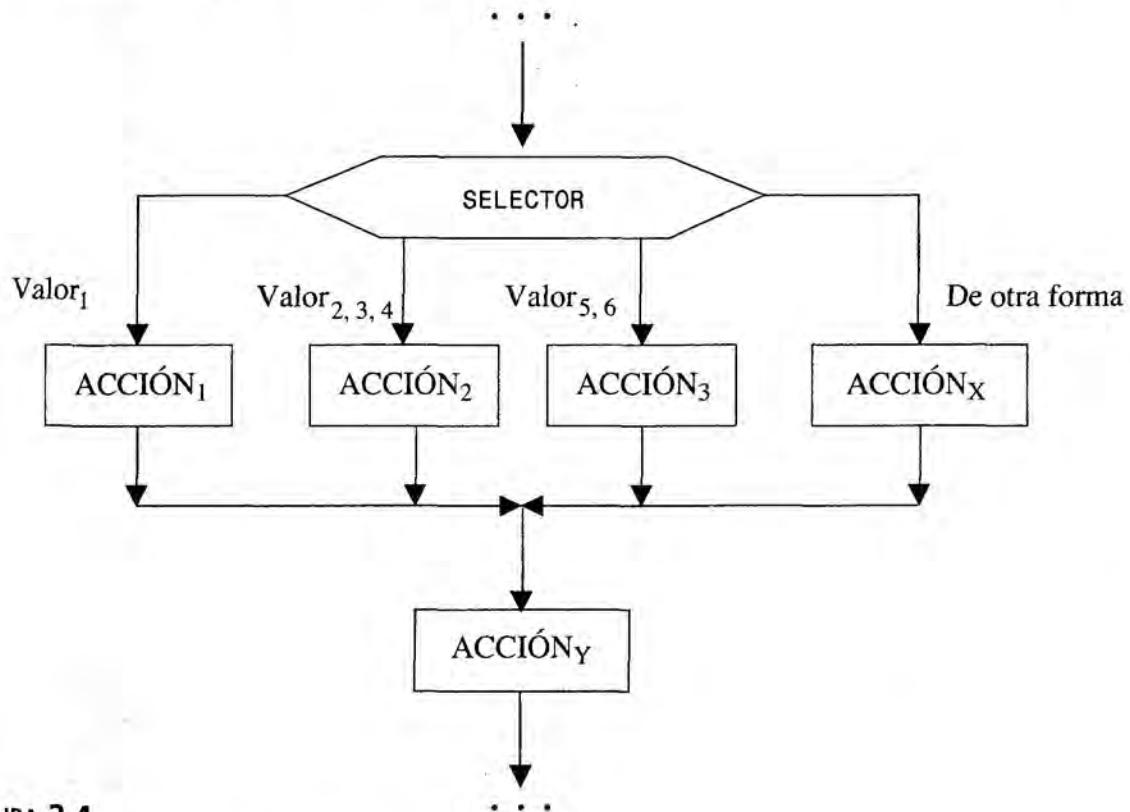


FIGURA 2.4
Estructura selectiva múltiple switch

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 <valor> : <acción>;
        break;
    case <valor2> :
    case <valor3> :
    case <valor4> : <acción2>;
        break;
    case <valor5> :
    case <valor6> : <acción3>;
        break;
    default:       : <acciónx>;
        break;
}
accióny;
. . .

```

EJEMPLO 2.5

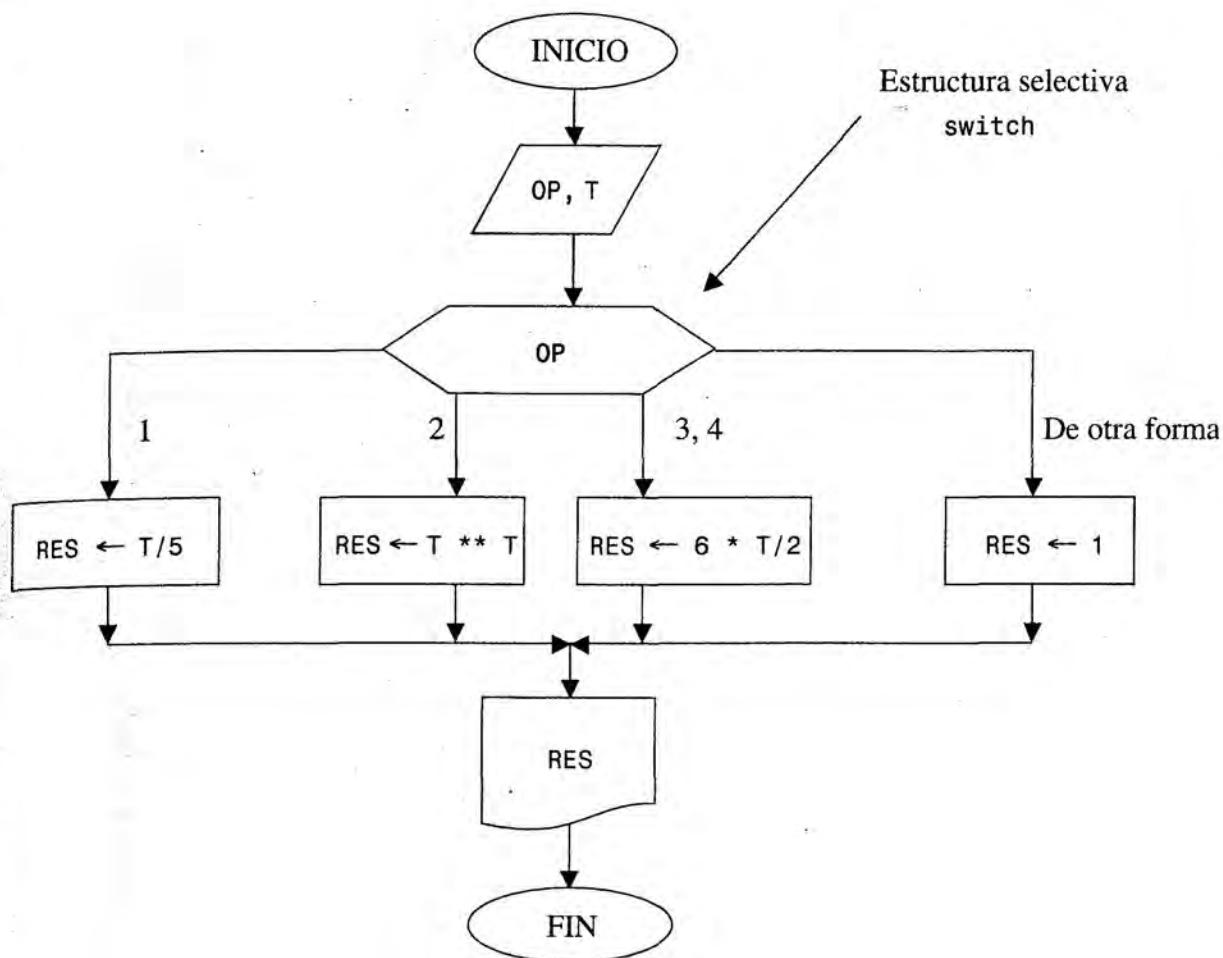
Construye un diagrama de flujo y el correspondiente programa en **C** que, al recibir como datos dos variables de tipo entero, obtenga el resultado de la siguiente función:

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

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.

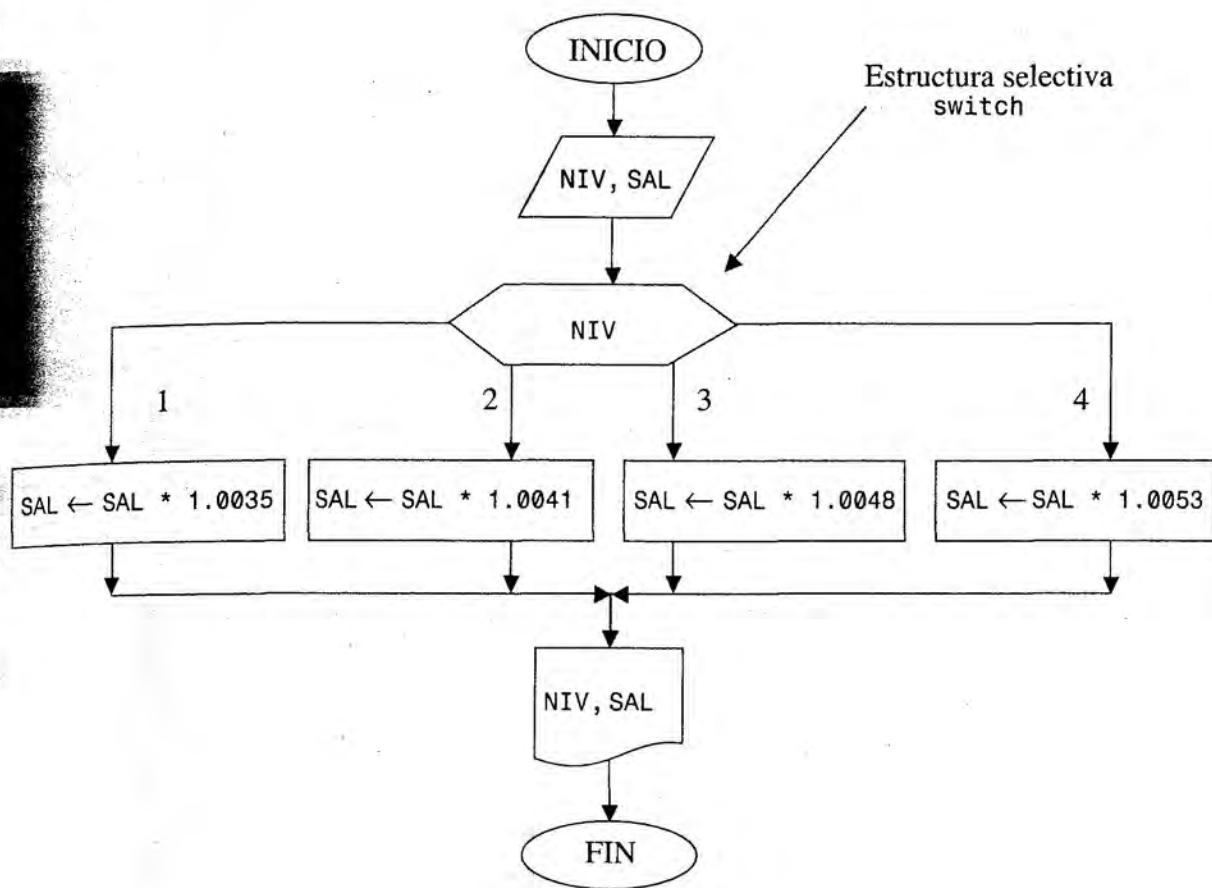
Datos: NIV y SAL

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: %.2f", NIV, SAL);
```

2.5. Estructuras selectivas en cascada

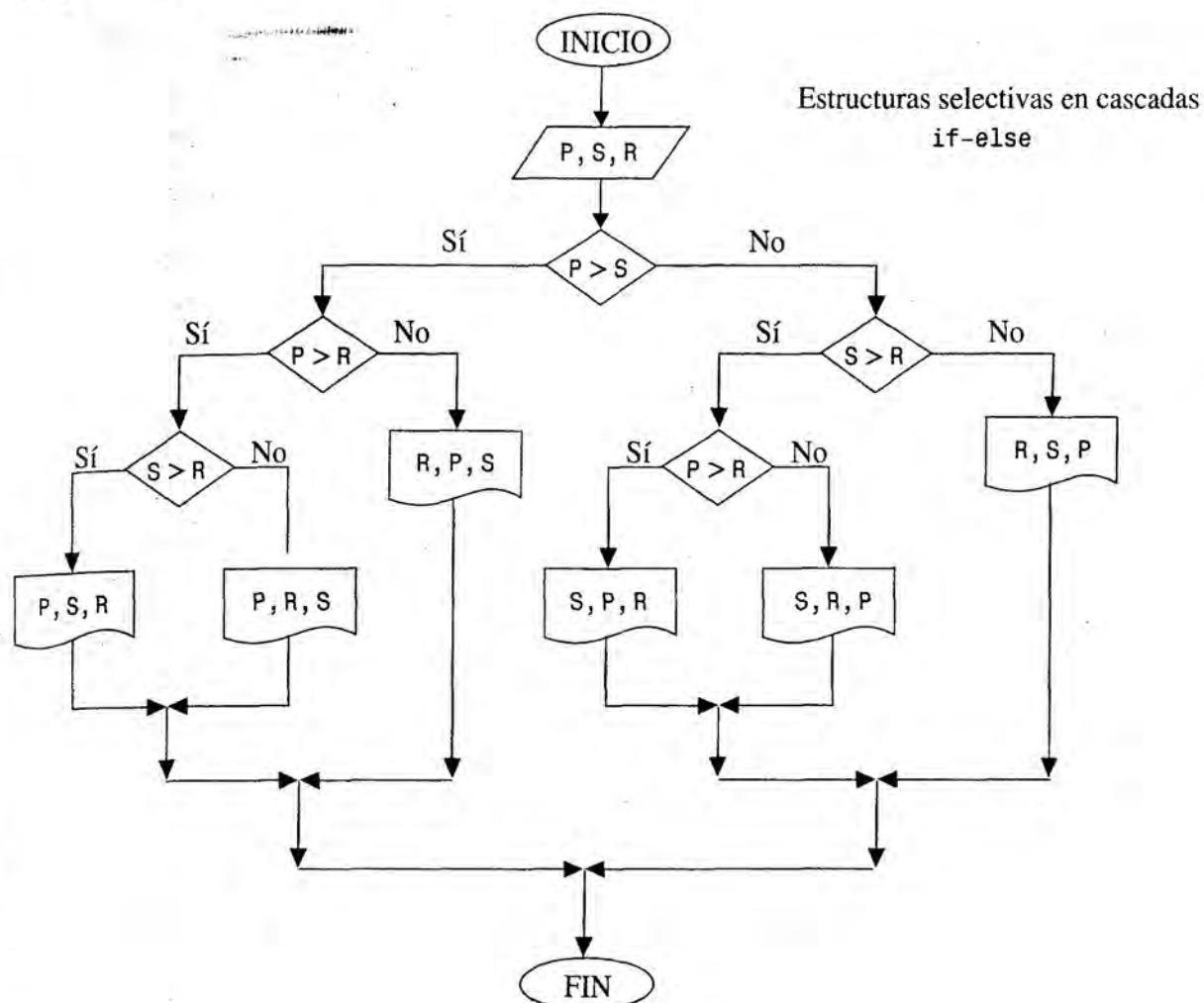
En el desarrollo de la solución de problemas se encuentran frecuentemente casos en los que, luego de tomar una decisión y señalar el correspondiente camino a seguir, es necesario tomar otra decisión. Este proceso se puede repetir numerosas veces. Una forma de solucionarlo es aplicando *estructuras selectivas 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: %.2f %.2f %.2f", P, S, R);
            else
                printf("\n\n El orden es P, R y S: %.2f %.2f %.2f", P, R, S);
        else
            printf("\n\n El orden es R, P y S: %.2f %.2f %.2f", R, P, S);
    else
        if (S > R)
            if (P > R)
                printf("\n\n El orden es S, P y R: %.2f %.2f %.2f", S, P, R);
            else
                printf("\n\n El orden es S, R y P: %.2f %.2f %.2f", S, R, P);
        else
            printf("\n\n El orden es R, S y P: %.2f %.2f %.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.

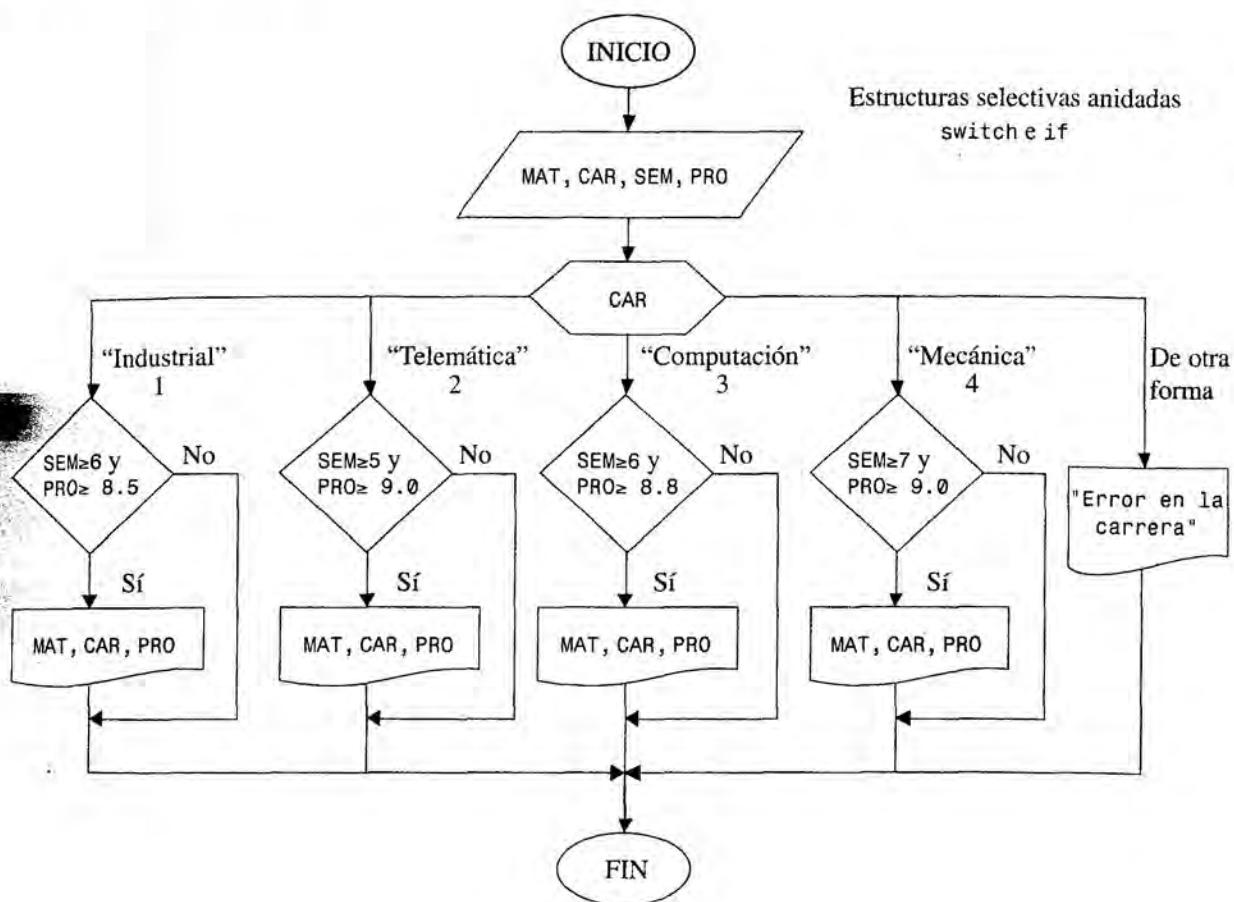
TABLA 2.4

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

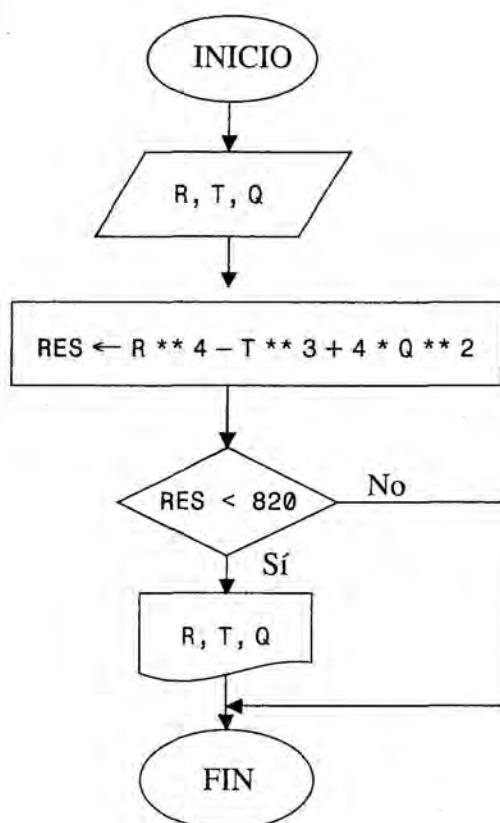
Escribe un diagrama de flujo y el correspondiente programa en C que, al recibir como datos tres valores enteros R, T y Q, determine si los mismos satisfacen la siguiente expresión, y que, en caso afirmativo, escriba los valores correspondientes de R, T y Q.

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

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

2

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. */

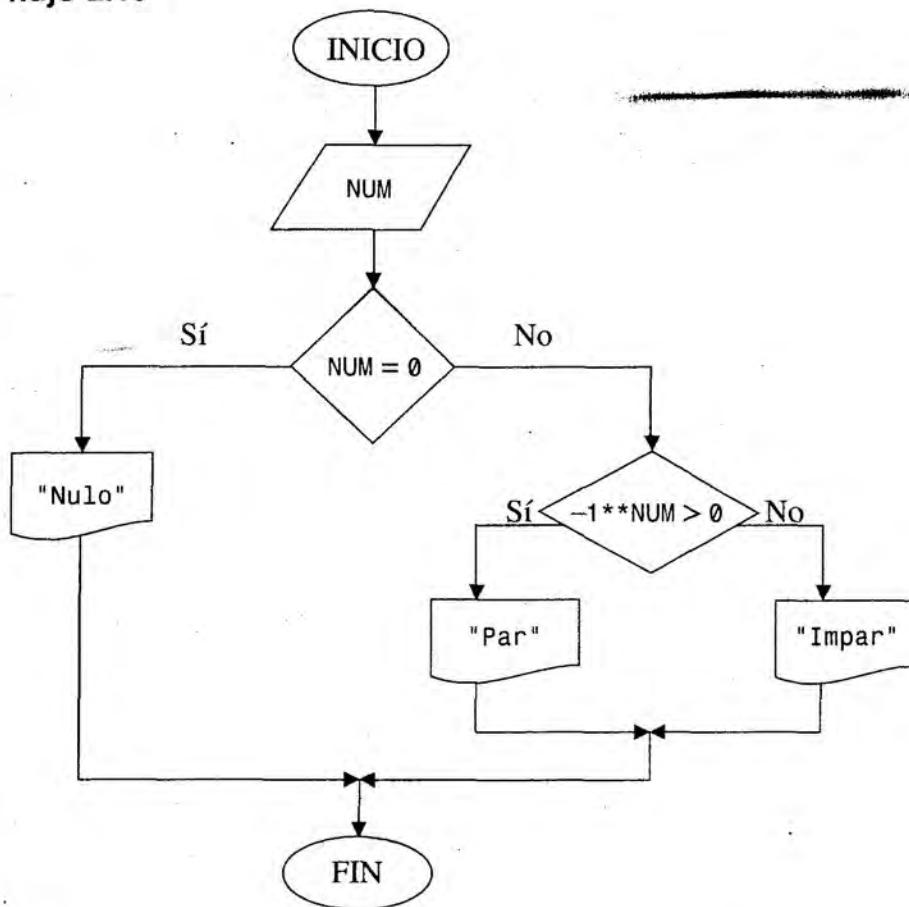
void main(void)
{
float RES;
int R, T, Q;
printf("Ingrese los valores de R, T y Q: ");
scanf("%d %d %d", &R, &T, &Q);
RES = pow(R, 4) - pow(T, 3) + 4 * pow(Q, 2);
if (RES < 820)
    printf("\nR = %d\tT = %d\tQ = %d", R, T, Q);
}
```

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



2

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

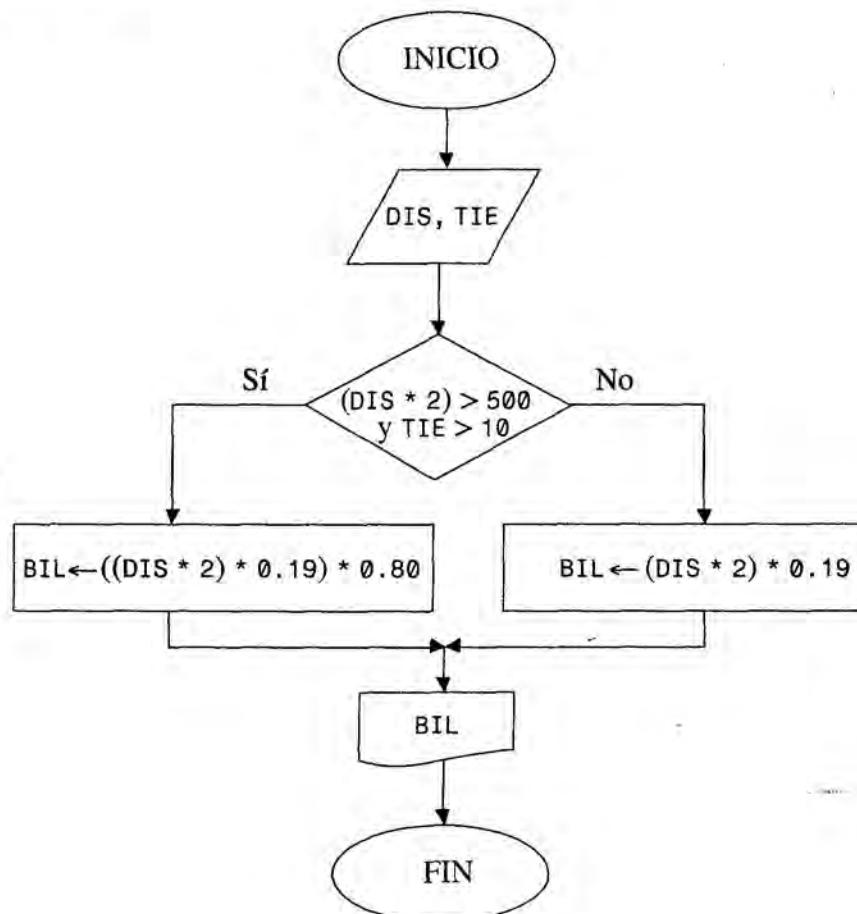
Construye un diagrama de flujo y el correspondiente programa en C que permita calcular el precio del billete ida y vuelta en ferrocarril, conociendo tanto la distancia entre las dos ciudades como el tiempo de estancia en la ciudad destino. Si el número de días de estancia es superior a 10 y la distancia 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.

IS y TIE: variables de tipo entero.
IL: 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: %.2f", BIL);
}

```

2

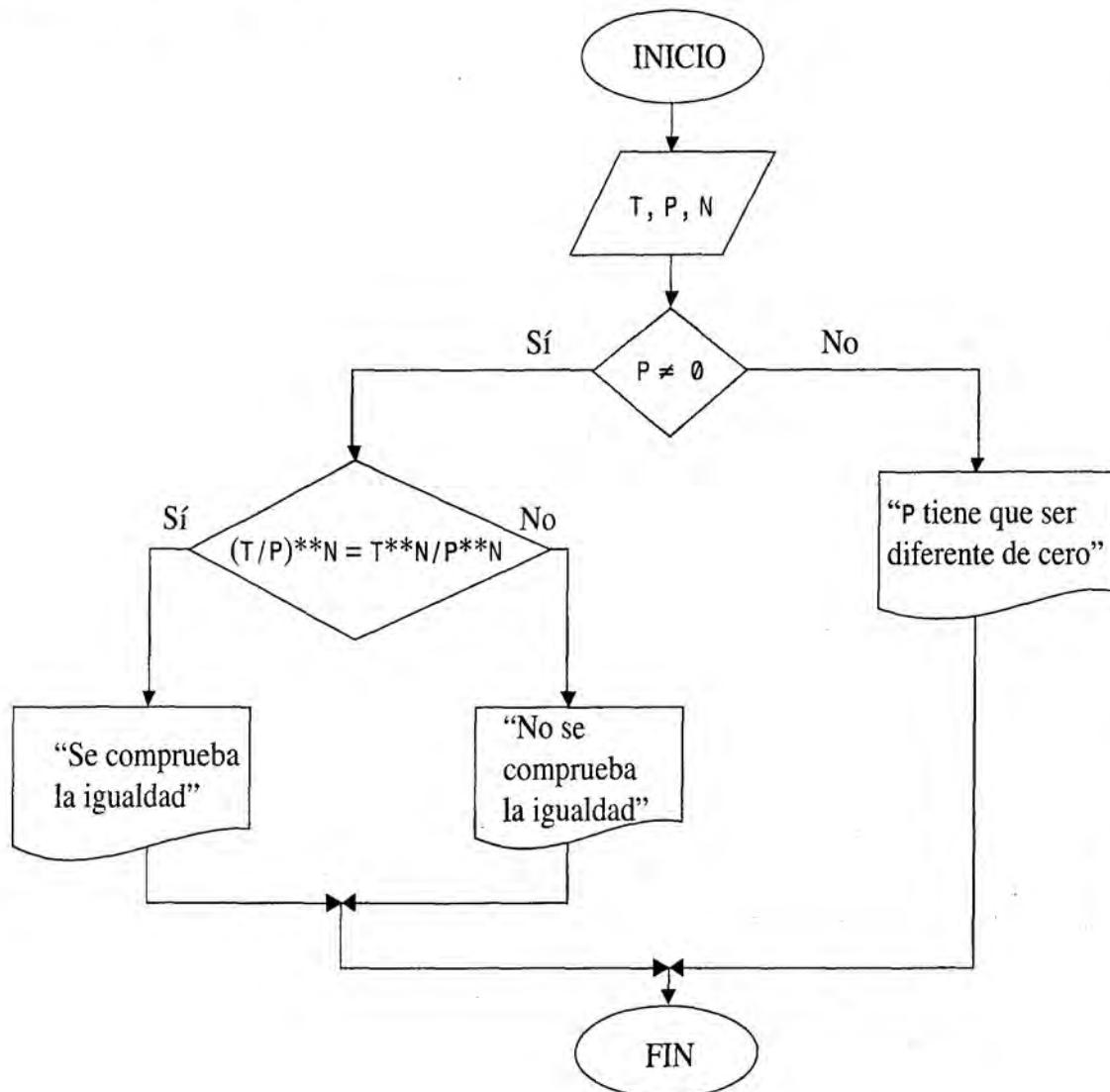
Problema PR2.4

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");

```

2

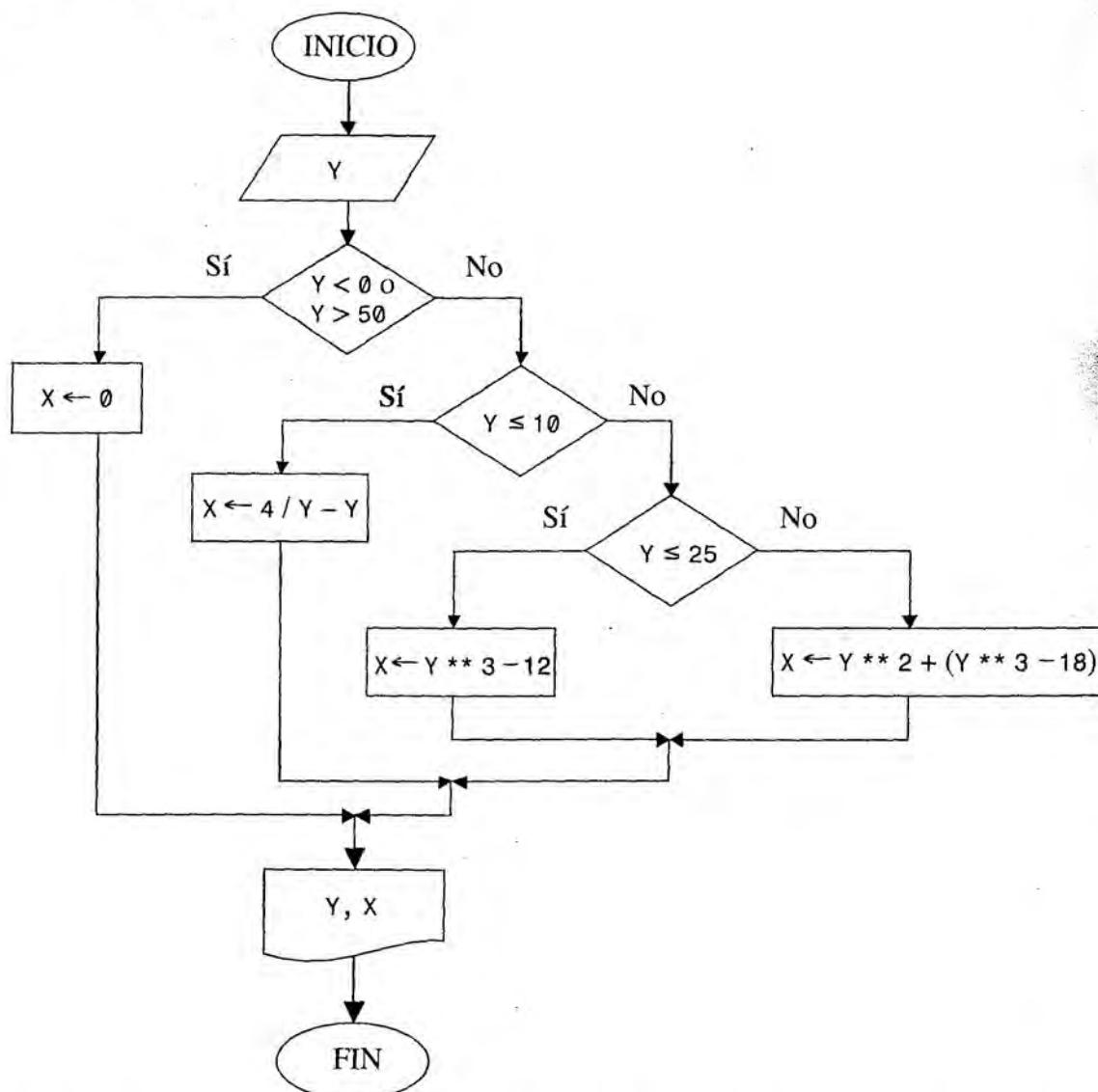
Problema PR2.5

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;
    intf("Ingrese el valor de Y: ");
    scanf("%d", &Y);
    if (Y < 0 || Y > 50)
        X = 0;
    se
        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);
}

```

2

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.

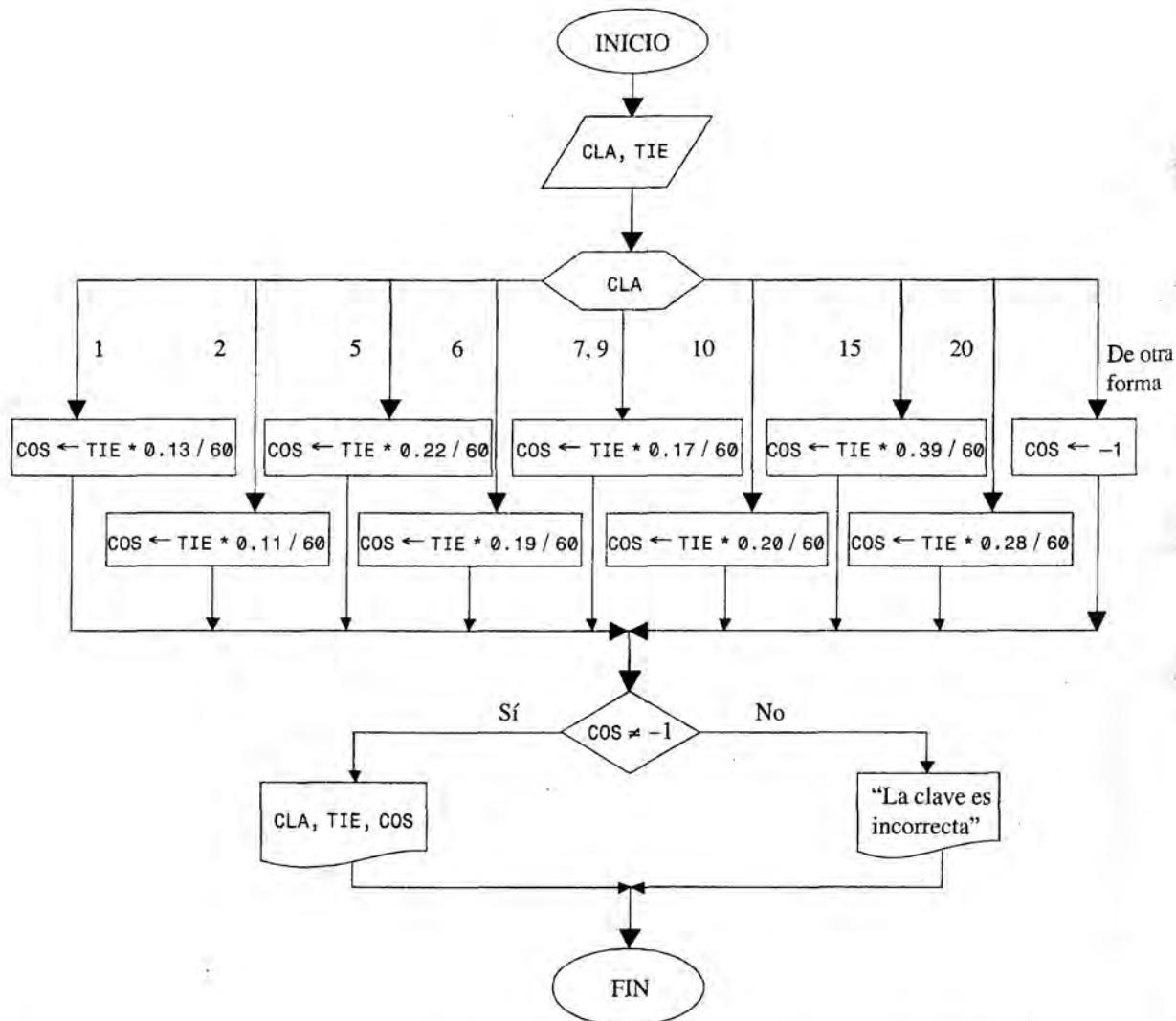
TABLA 2.5

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");
}

```

2

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%.

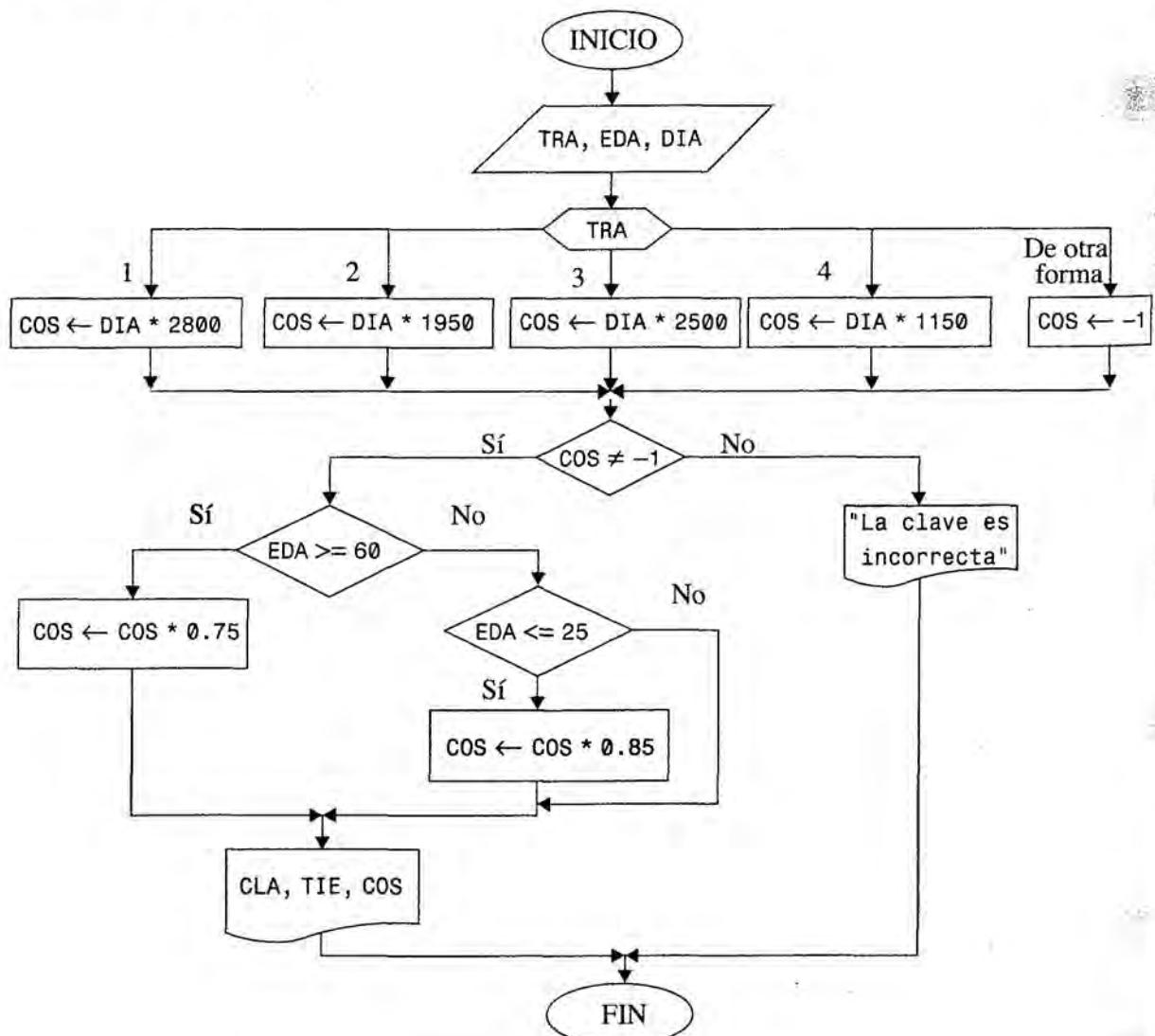
TABLA 2.6

<i>Tipo de tratamiento</i>	<i>Costo/cliente/día</i>
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: %.2f",
                TRA, DIA, COS);
        }
        else
            printf("\nLa clave del tratamiento es incorrecta");
    }
}
```

2

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.

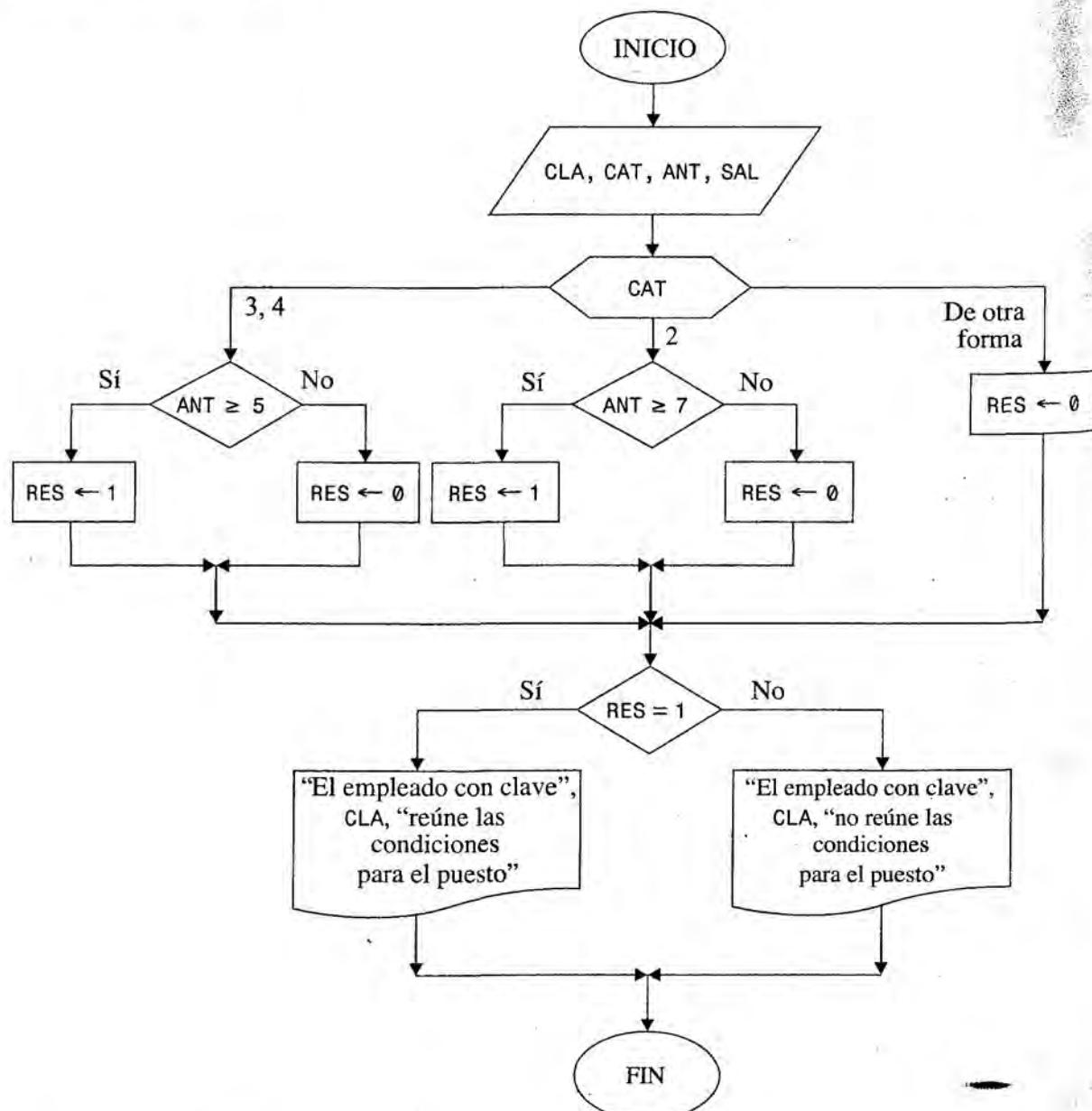
Datos: CLA, CAT, ANT y SAL

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.

$\text{Salario} < \$18,000 \Rightarrow \text{Incremento } 12\%.$

$\$18,000 \leq \text{Salario} \leq \$30,000 \Rightarrow \text{Incremento } 8\%.$

$\$30,000 < \text{Salario} \leq \$50,000 \Rightarrow \text{Incremento } 7\%.$

$\$50,000 < \text{Salario} \Rightarrow \text{Incremento } 6\%.$

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: n_1 y n_2 (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).

2

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 \Rightarrow Descuento 0%.

\$800 \leq Compra \leq \$1500 \Rightarrow Descuento 10%.

\$1500 < Compra \leq \$5000 \Rightarrow Descuento 15%.

\$5000 < Compra \Rightarrow 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

<i>Medidas de longitud</i>	<i>Medidas de volumen</i>	<i>Medidas de peso</i>
1 pulgada \equiv 25.40 milímetros	1 pie ³ \equiv 0.02832 metros ³	1 onza \equiv 28.35 gramos
1 yarda \equiv 0.9144 metros	1 yarda ³ \equiv 0.7646 metros ³	1 libra \equiv 0.45359 kilogramos
1 milla \equiv 1.6093 kilómetros	1 pinta \equiv 0.56826 litros	1 ton. inglesa \equiv 1.0160 toneladas

TABLA 2.7 Continuación

<i>Medidas de longitud</i>	<i>Medidas de volumen</i>	<i>Medidas de peso</i>
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		

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

<i>Categoría trabajador</i>	<i>Hora extra</i>
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.

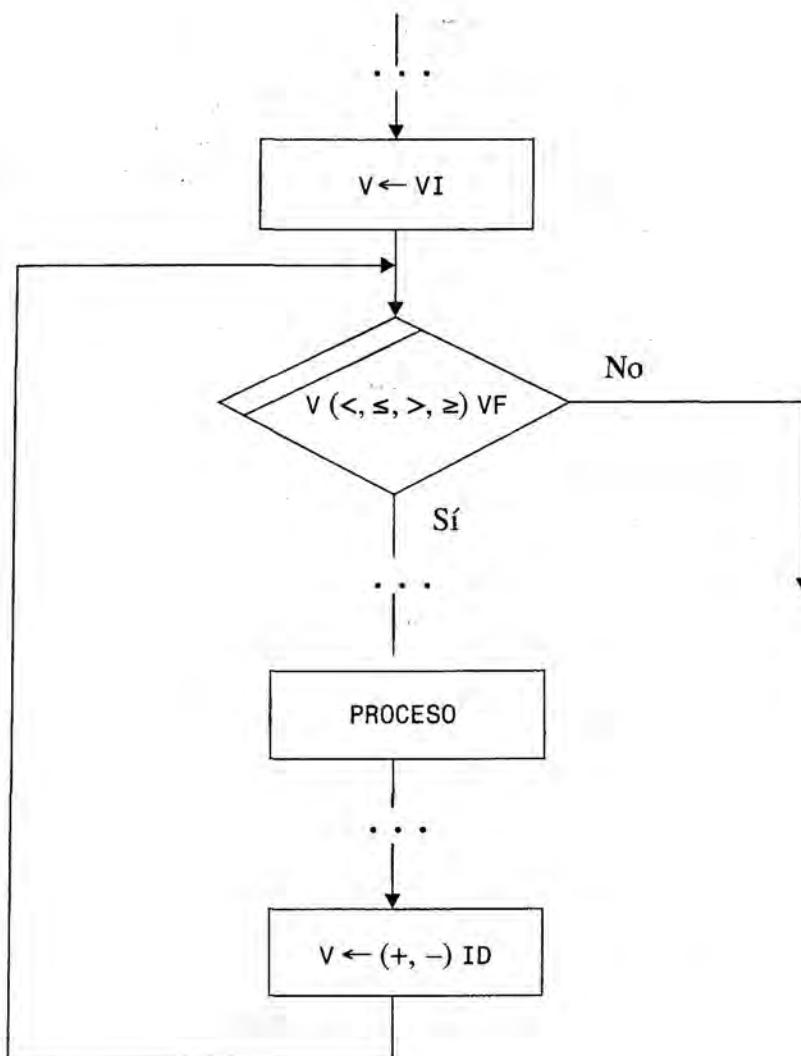
Otra estructura algorítmica repetitiva es `do-while`. A diferencia de las estructuras anteriores en las que las condiciones se evalúan al principio del ciclo, en ésta se evalúan al final. Esto implica que el conjunto de instrucciones se ejecuta al menos una vez. El `do-while` es una estructura de menor 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.

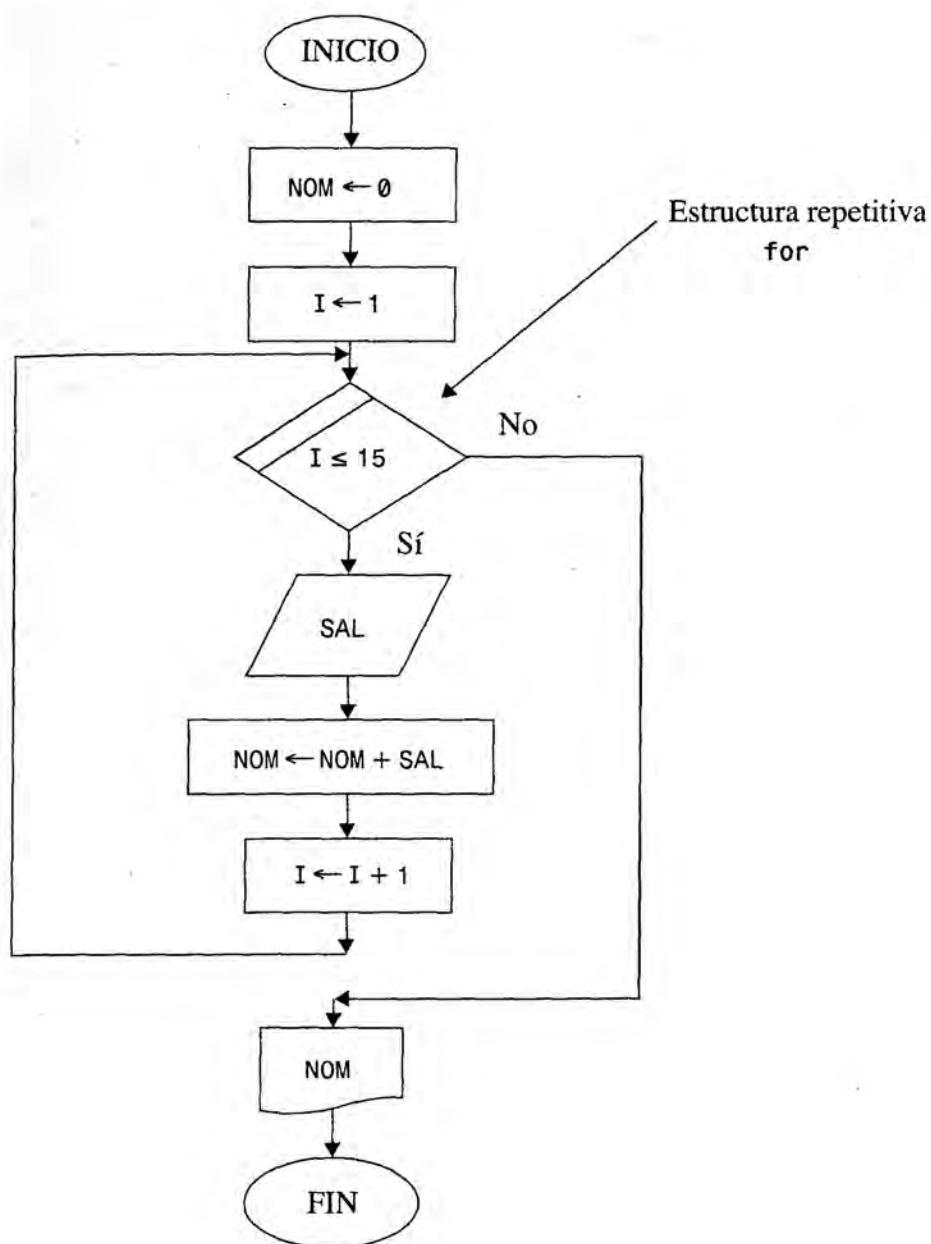
EJEMPLO 3.1

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: SAL₁, SAL₂,..., SAL₁₅

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



3

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>I</i>	<i>Datos</i>	<i>Resultados</i>
	<i>SAL</i>	<i>NOM</i>
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.

: variable de tipo entero.
AL 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

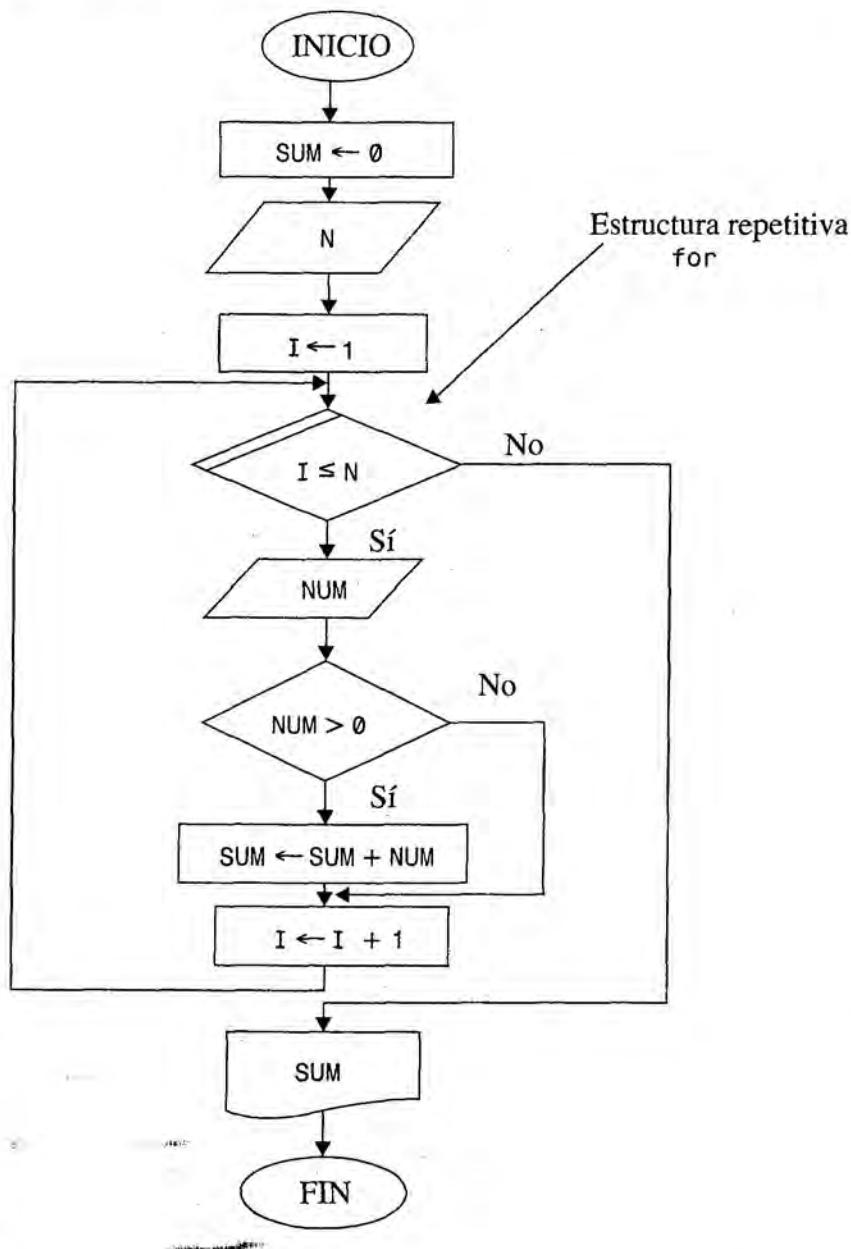
Escribe un diagrama de flujo y el correspondiente programa en C que, al recibir como datos N números enteros, obtenga solamente la suma de los números positivos.

Datos: N, NUM₁, NUM₂, ..., 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);
}
```

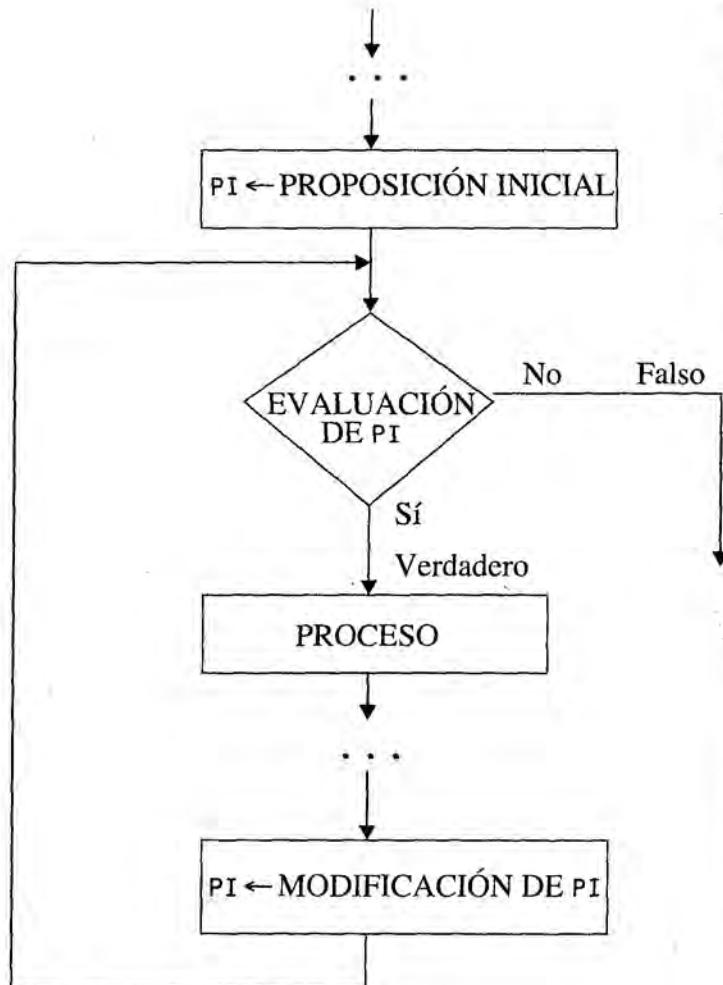
3

3.3. 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 encontramos 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

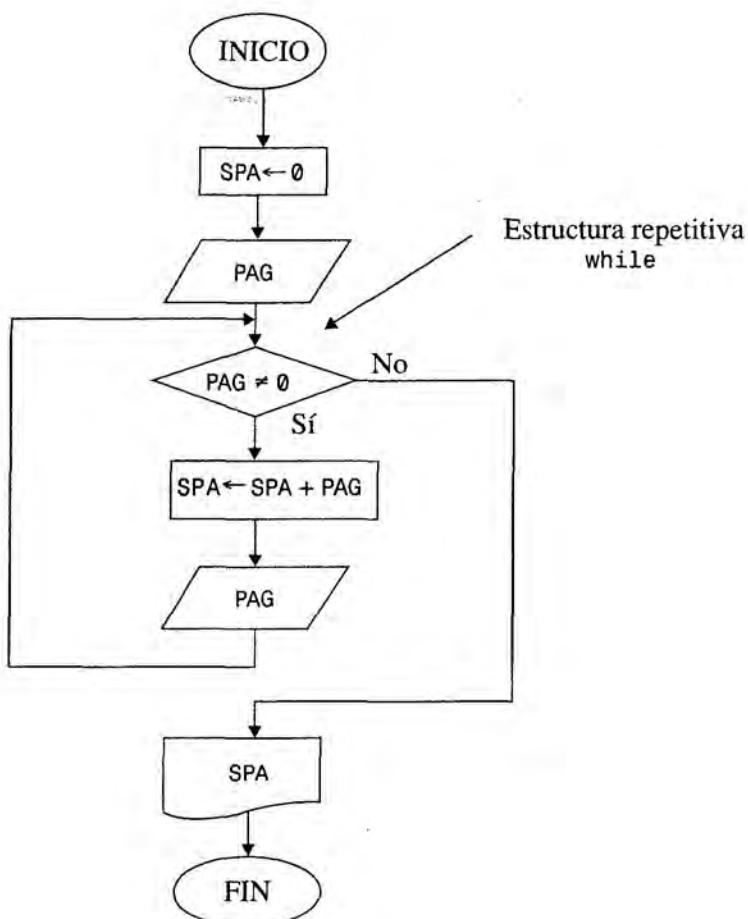
Construye un diagrama de flujo y el programa correspondiente en *C* que, al recibir como datos los pagos efectuados en el último mes, permita obtener la suma de los mismos.

Datos: $PAG_1, PAG_2, \dots, 0$

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.

3

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);
}
```

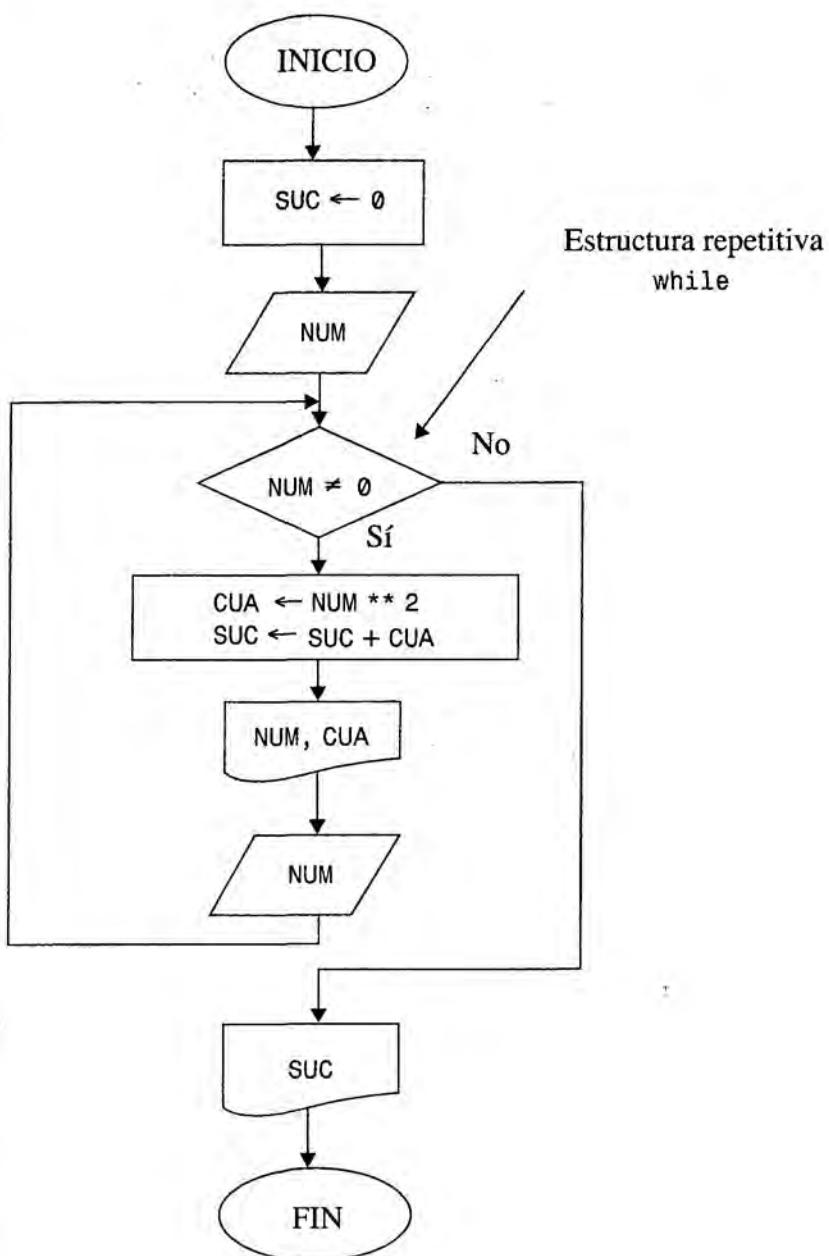
EJEMPLO 3.4

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: NUM₁, NUM₂, NUM₃, ..., 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.



3

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:

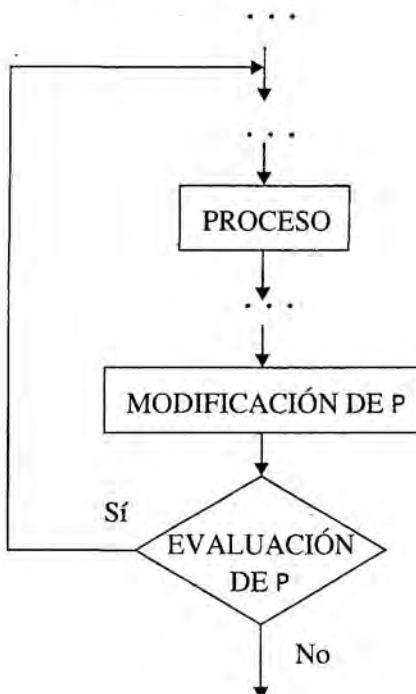


FIGURA 3.3

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 */
.
.

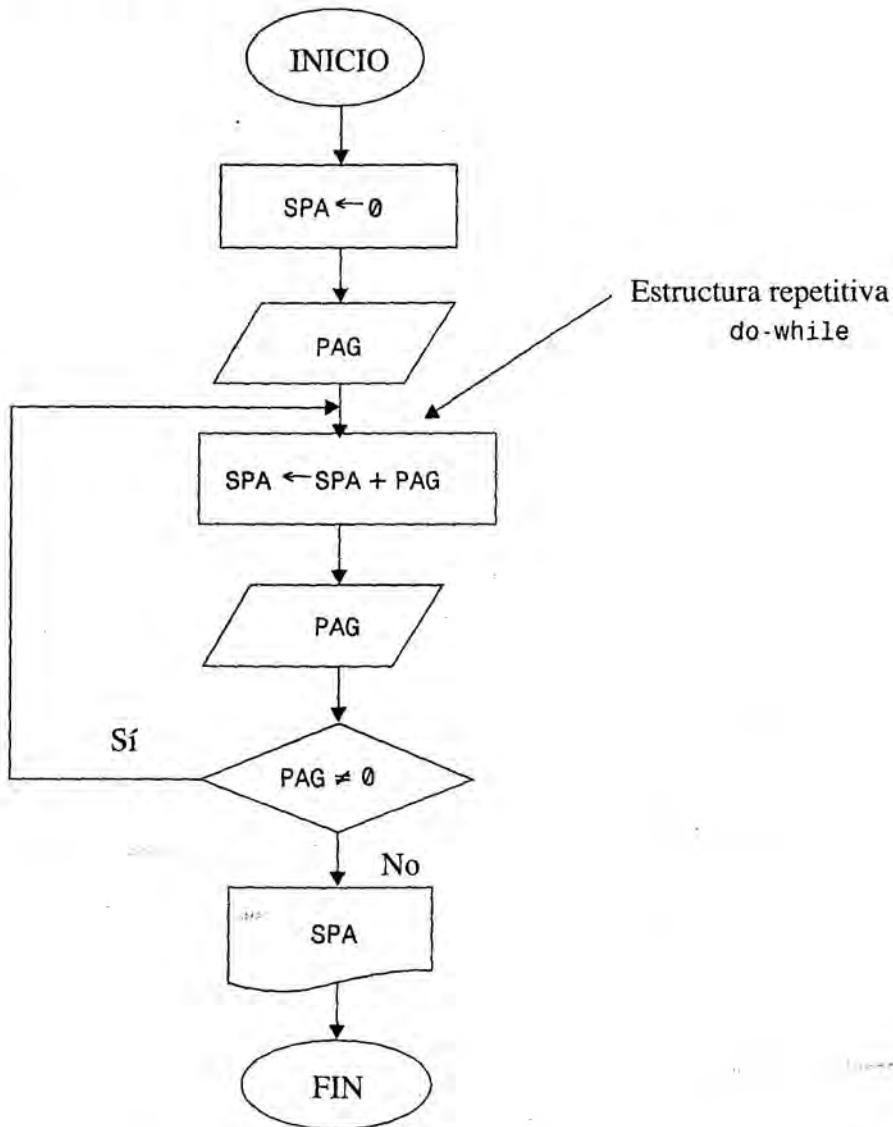
```

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: $PAG_1, PAG_2, \dots, 0$

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);
}
```

3

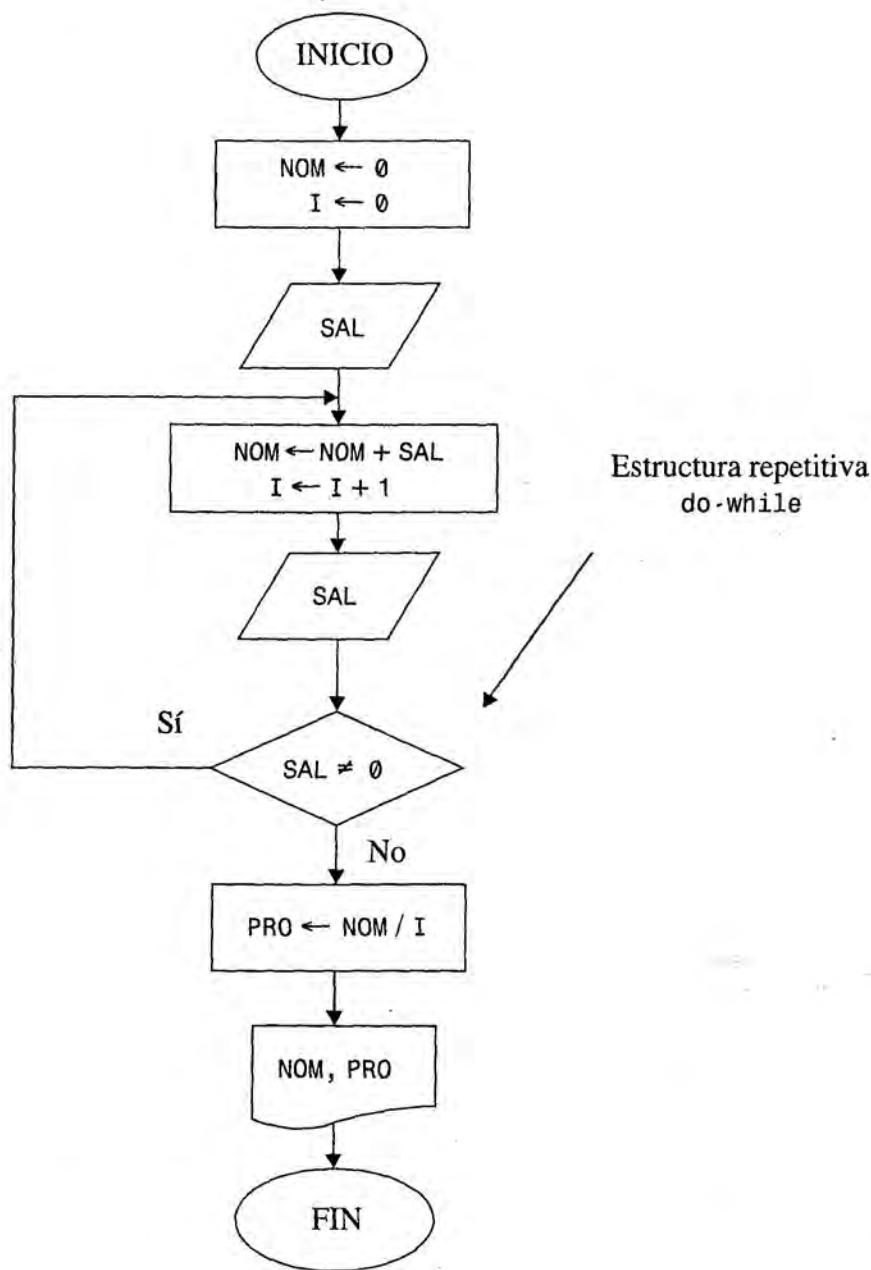
EJEMPLO 3.6

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: SAL₁, SAL₂, SAL₃, ..., 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);
}
```

3

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

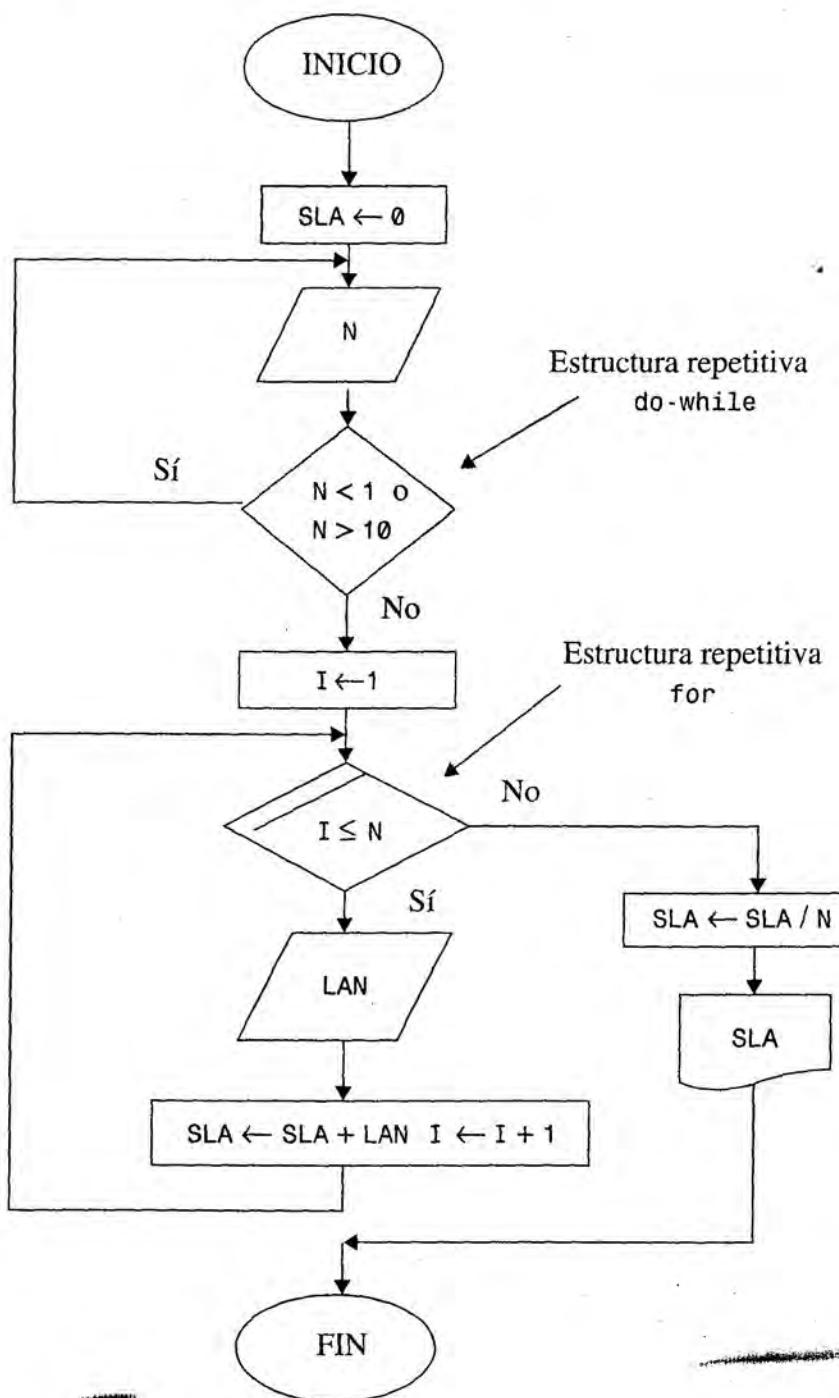
Escribe un diagrama de flujo y el correspondiente programa en C que, al recibir como datos los N lanzamientos del martillo de la atleta cubana ganadora de la medalla de oro en las últimas olimpiadas celebradas en Atenas, calcule el promedio de dichos lanzamientos.

Datos: N, LAN₁, LAN₂, ..., LAN_N

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

LAN_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);
```

3

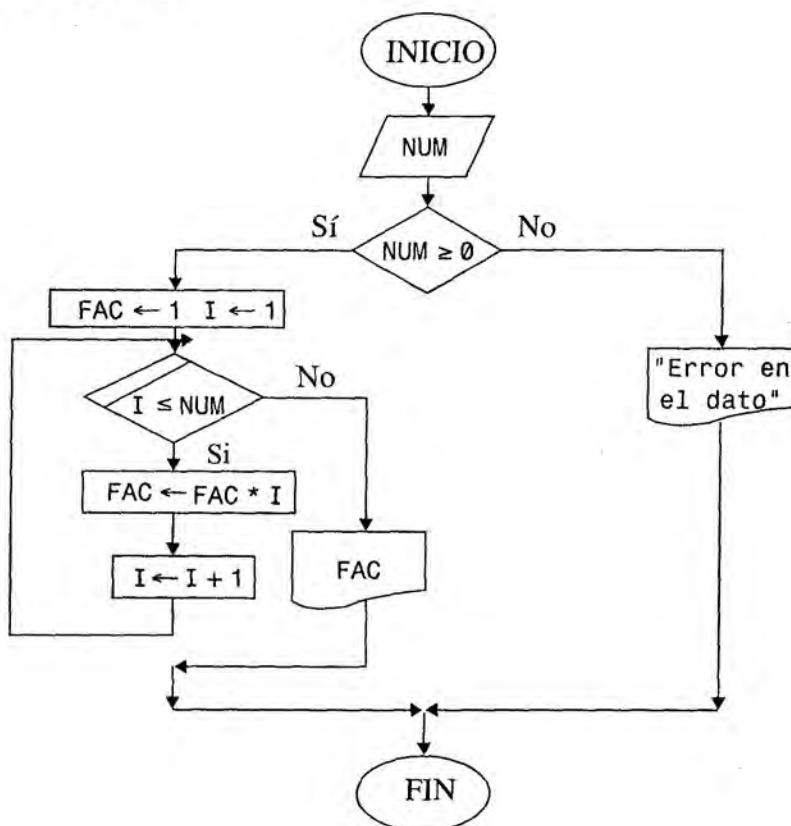
Problemas resueltos

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 = FAC * I;
    printf("El factorial es: %ld", FAC);
}
  
```

```

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

```

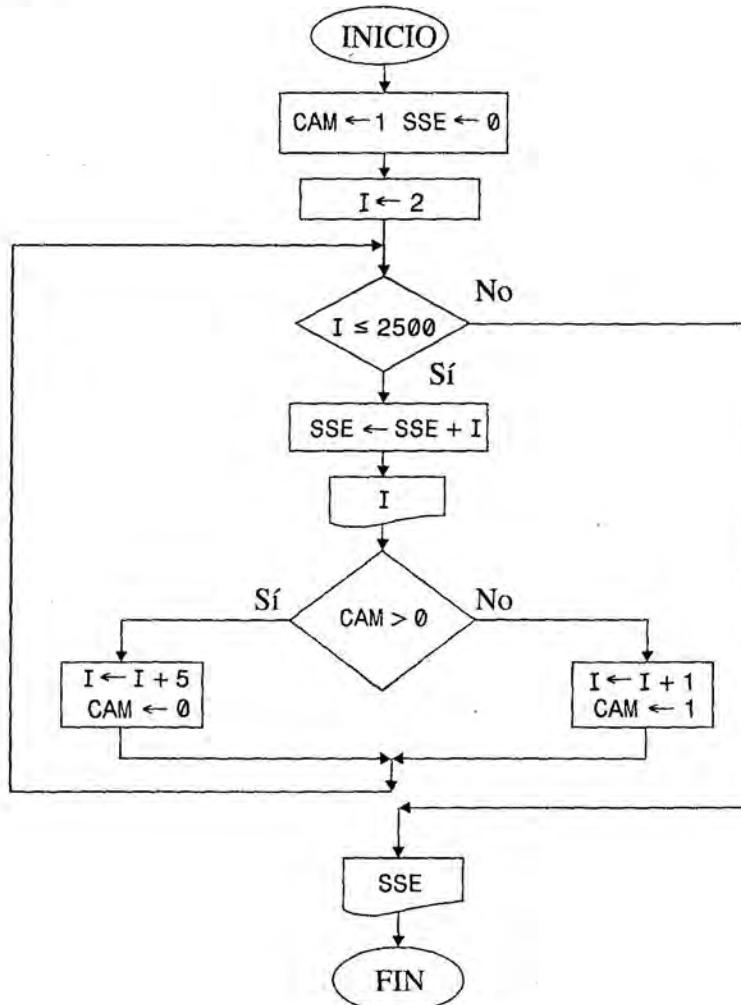
Problema PR3.2

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

3

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.

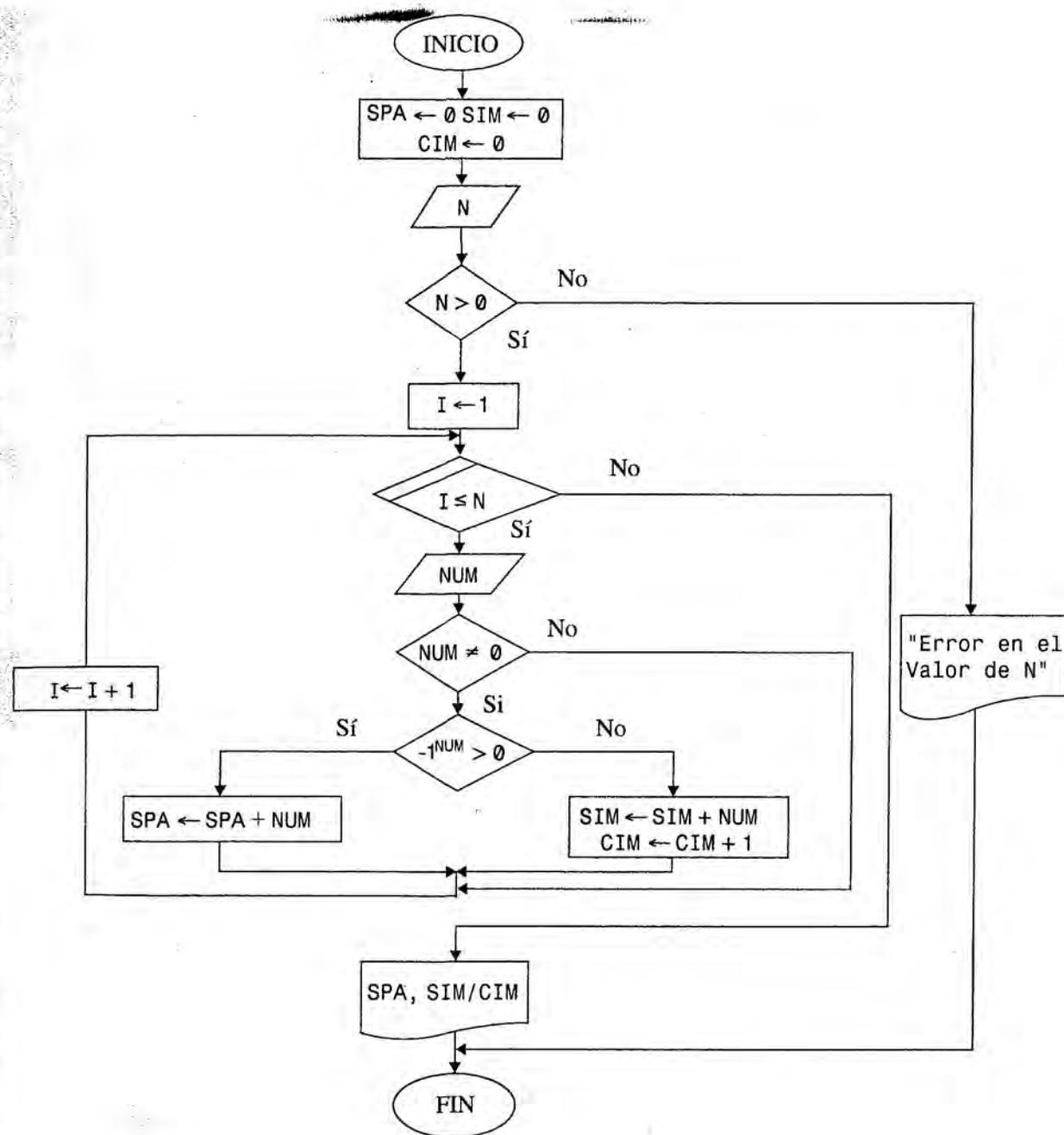
Datos: N, NUM₁, NUM₂, ..., NUM_N

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

3



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");
}

```

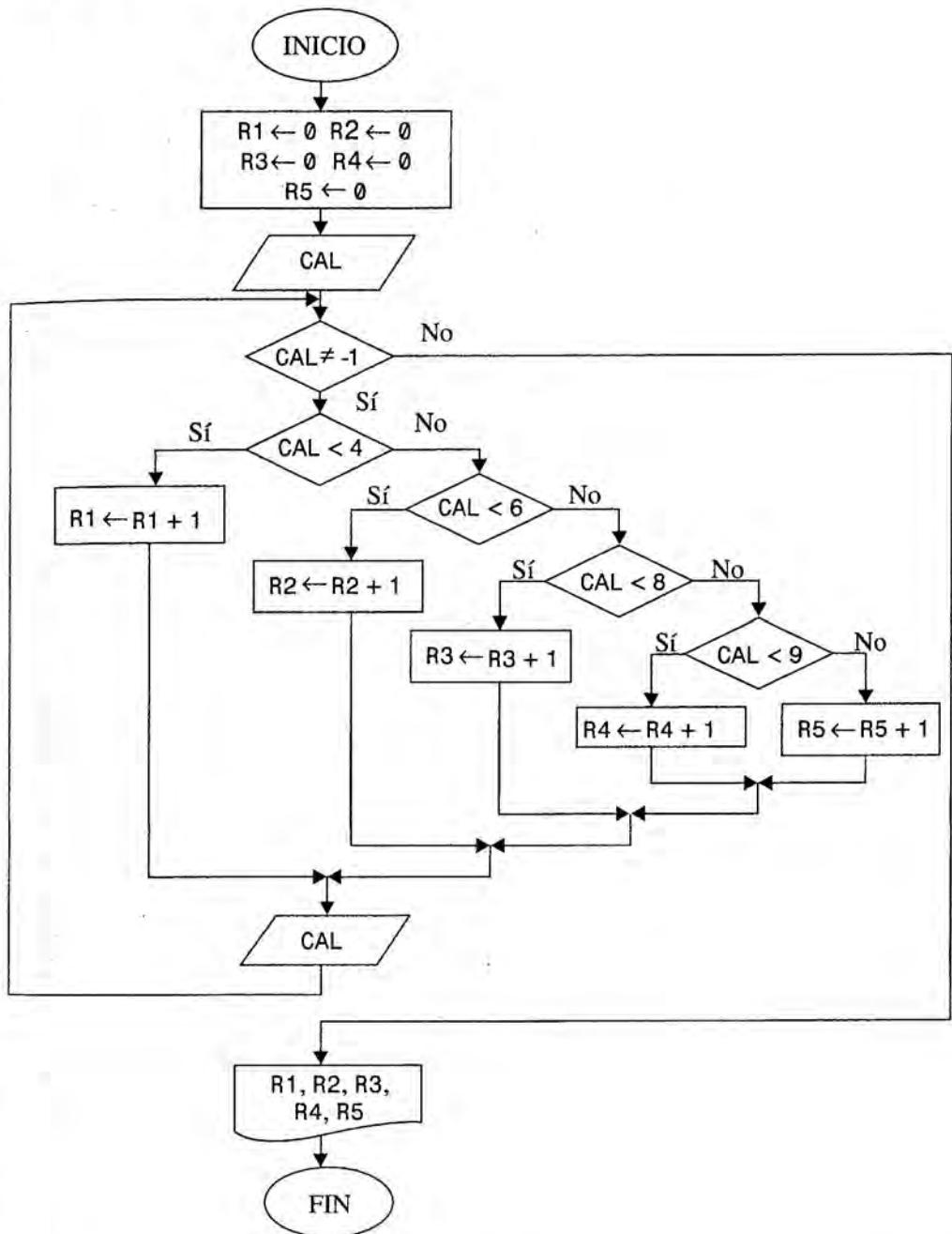
Problema PR3.4

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: $CAL_1, CAL_2, \dots, -1$ (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.

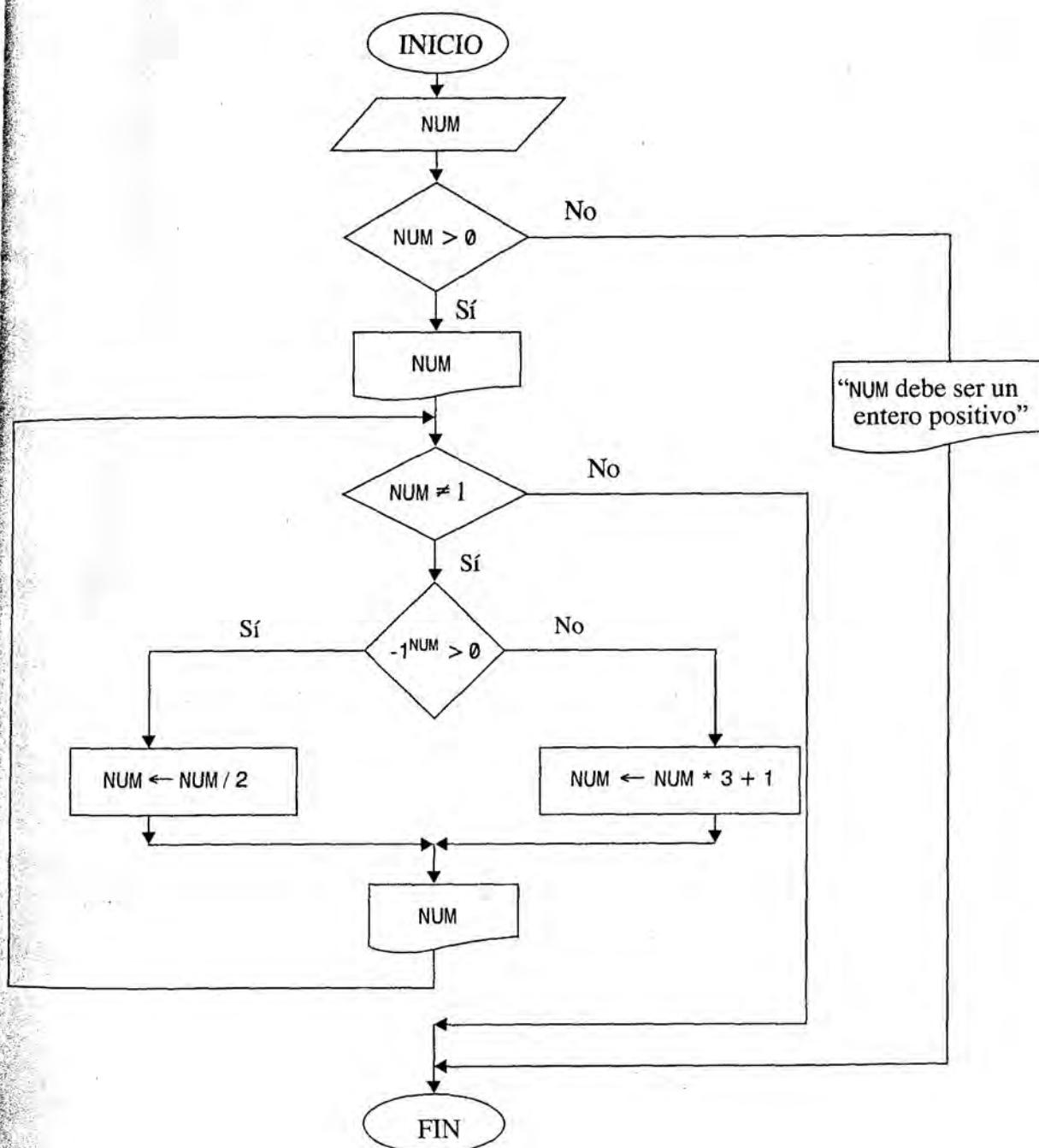
Sucesión de ULAM

1. Inicia con cualquier entero positivo.
2. Si el número es par, divídalo 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");
}

```

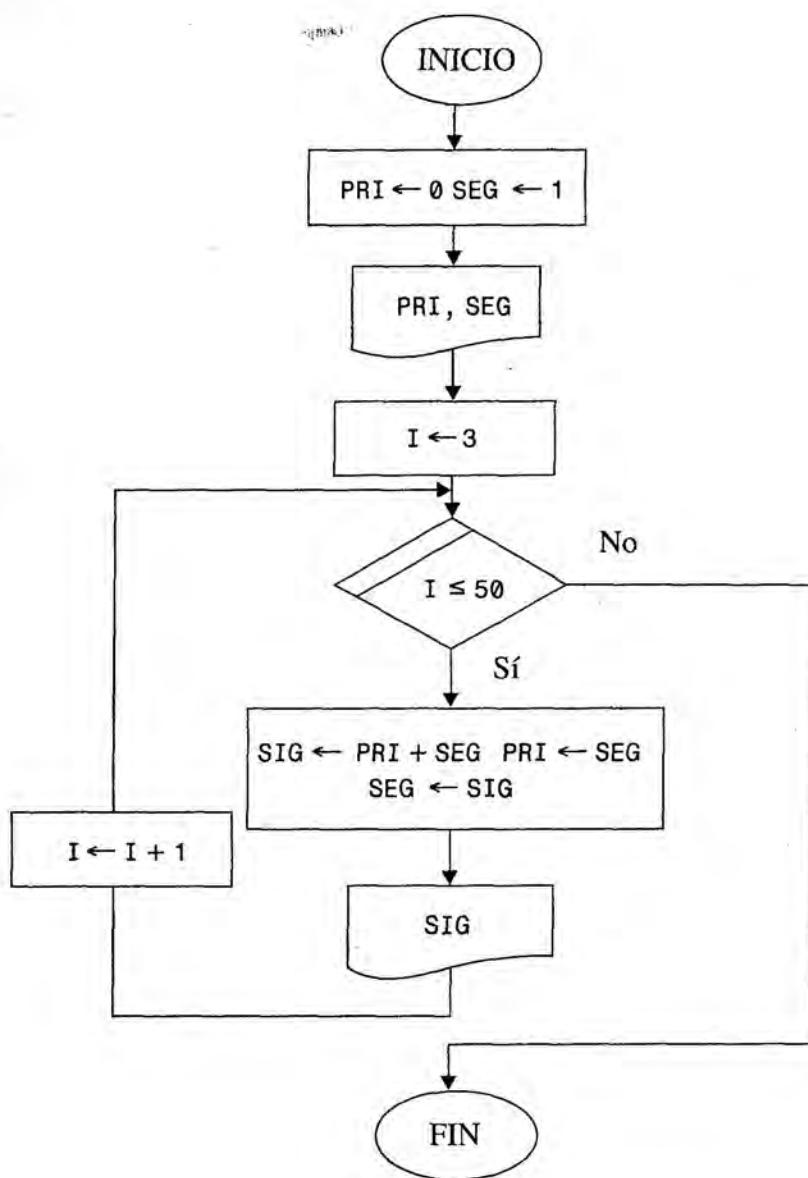
Problema PR3.6

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

3



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

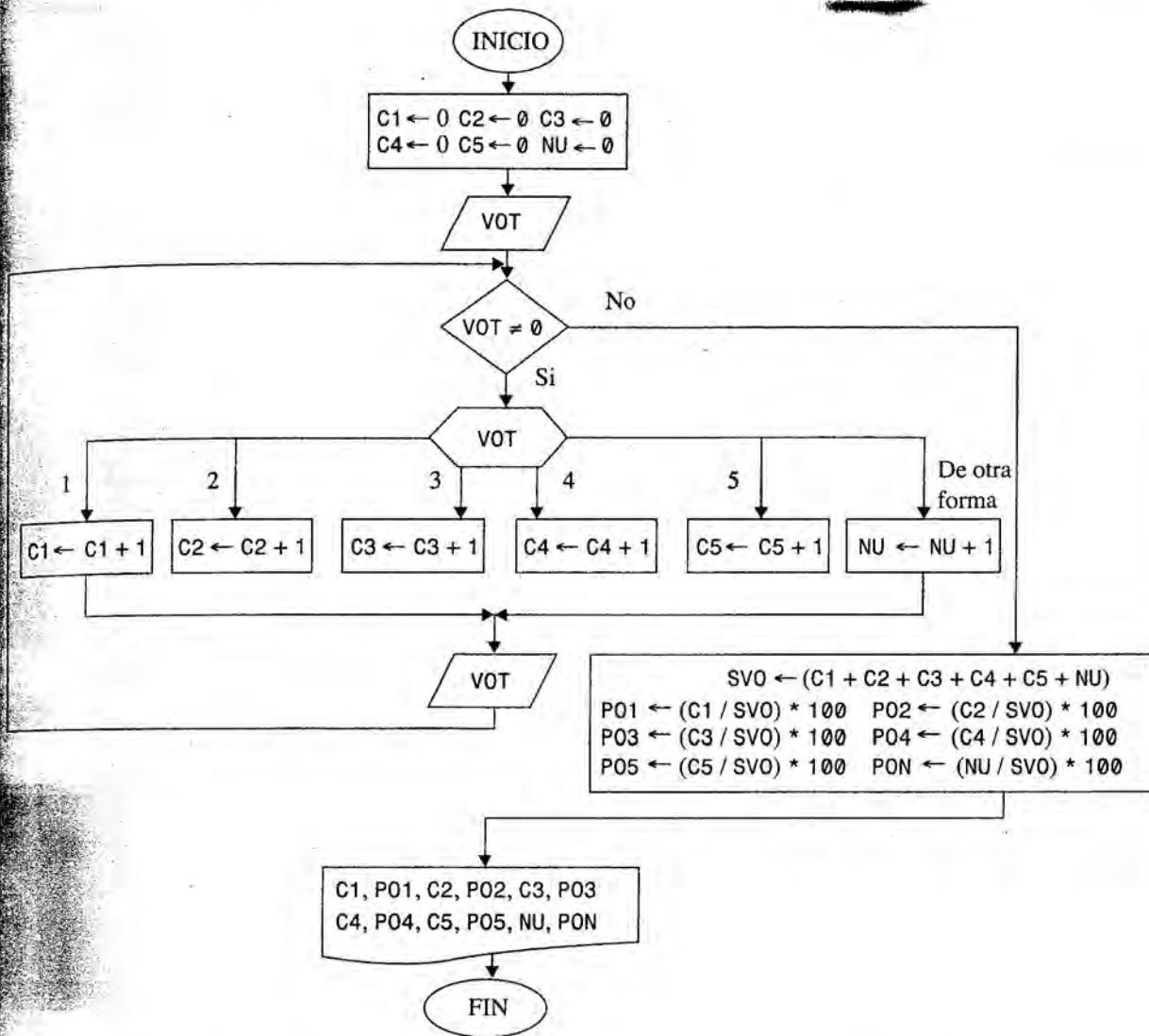
Los organizadores de un acto electoral solicitaron realizar un programa de cómputo para manejar el conteo de los votos. En la elección hay cinco candidatos, los cuales se representan con los valores comprendidos de 1 a 5. Construye un diagrama de flujo y el correspondiente programa en C que permita obtener 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



3

Donde: c_1, c_2, c_3, c_4, c_5 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.

$P_{01}, P_{02}, P_{03}, P_{04}$, P_{05} 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.
P01, P02, P03, P04, P05, PON: variables de tipo real.*/

void main(void)
{
    int VOT, C1 = 0, C2 = 0, C3 = 0, C4 = 0, C5 = 0, NU = 0, SVO;
    float P01, P02, P03, P04, P05, 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;
    P01 = ((float) C1 / SVO) * 100;
    P02 = ((float) C2 / SVO) * 100;
    P03 = ((float) C3 / SVO) * 100;
    P04 = ((float) C4 / SVO) * 100;
    P05 = ((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, P01);
    printf("\nCandidato 2: %d votos -- Porcentaje: %5.2f", C2, P02);
    printf("\nCandidato 3: %d votos -- Porcentaje: %5.2f", C3, P03);
    printf("\nCandidato 4: %d votos -- Porcentaje: %5.2f", C4, P04);
    printf("\nCandidato 5: %d votos -- Porcentaje: %5.2f", C5, P05);
    printf("\nNulos:      %d votos -- Porcentaje: %5.2f", NU, PON);
}
```

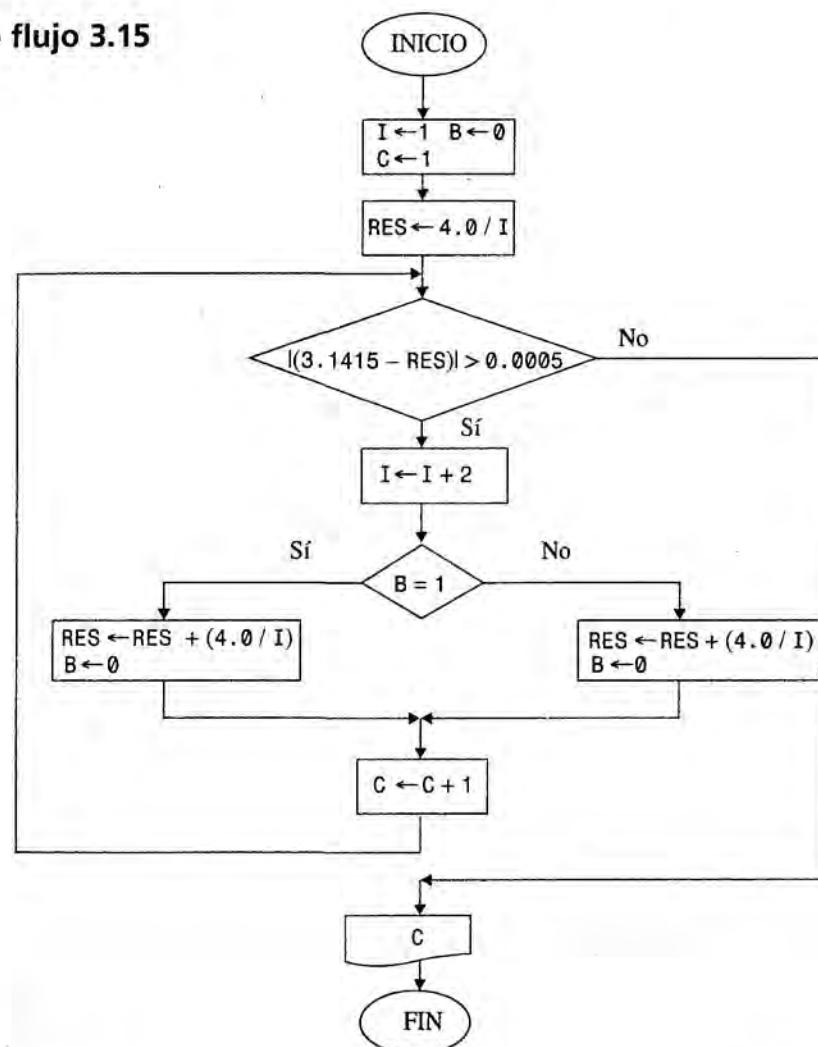
Problema PR3.8

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

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

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



3

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: MAT₁, CAL_{1,1}, CAL_{1,2}, CAL_{1,3}, CAL_{1,4}, CAL_{1,5}
MAT₂, CAL_{2,1}, CAL_{2,2}, CAL_{2,3}, CAL_{2,4}, CAL_{2,5}

...

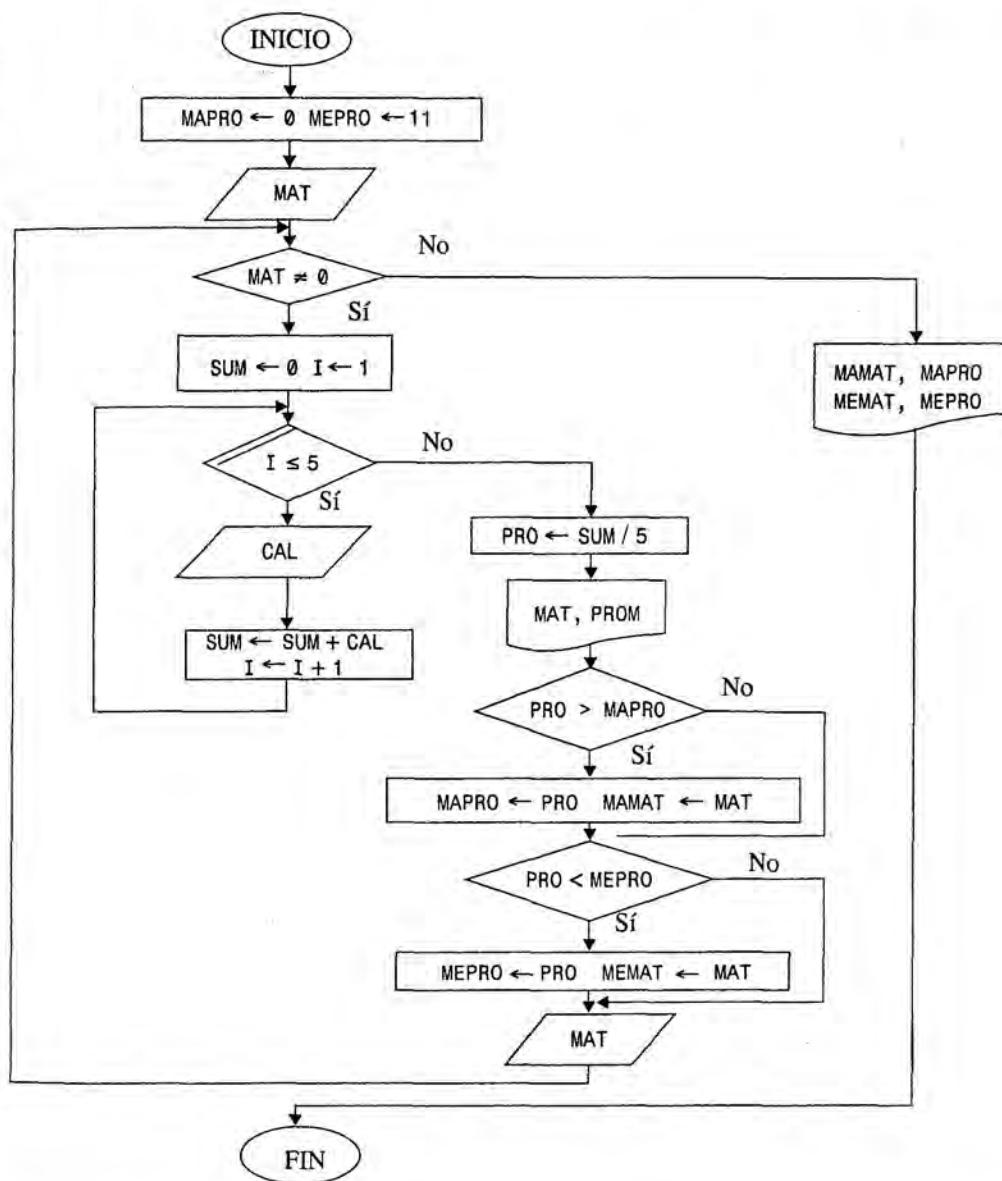
0

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



3

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 MEPROM 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 pro-
medio de calificaciones de cada uno de ellos y, además, los alumnos con el mejo-
y peor promedio.

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

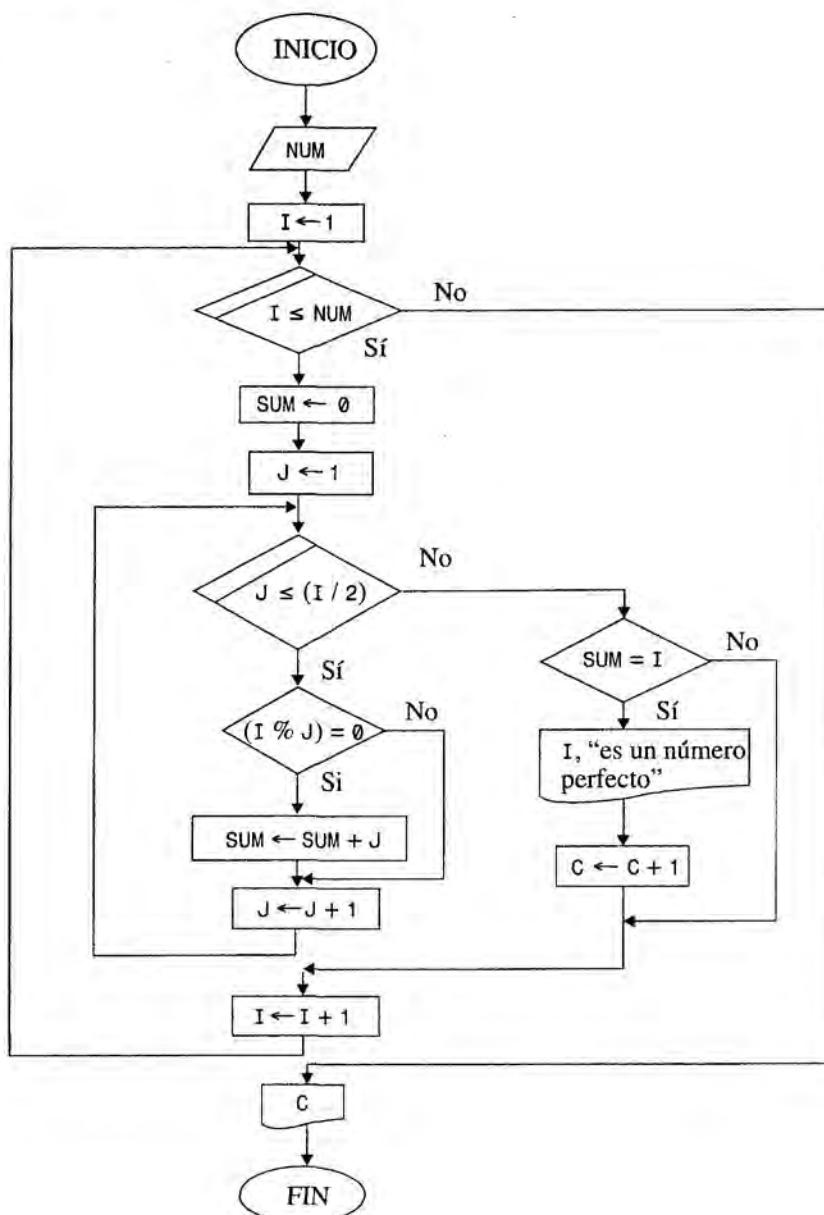
void main(void)
{
    int I, MAT, MAMAT, MEMAT;
    float SUM, PRO, CAL, MAPRO = 0.0, MEPROM = 11.0;
    printf("\nIngrese la matrícula del primer alumno:\t");
    scanf("%d", &MAT);
    while (MAT)
    {
        SUM = 0;
        for (I = 1; I <= 5; I++)
        {
            printf("\tIngrese la calificación 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 < MEPROM)
        {
            MEPROM = 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, MEPROM);
}
```

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 supplementarios

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:

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

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:

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

3

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**, **NUM₁**, **NUM₂**, ..., **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, \dots, 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:

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

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, \dots, 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:

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

Datos: $\text{NOR}_1, \text{CEN}_1, \text{SUR}_1, \text{GOL}_1, \text{PAC}_1, \text{CAR}_1, \dots, \text{NOR}_{12}, \text{CEN}_{12}, \text{SUR}_{12}, \text{GOL}_{12}, \text{PAC}_{12}, \text{CAR}_{12}$.

Donde: $\text{NOR}_i, \text{CEN}_i, \text{SUR}_i, \text{GOL}_i, \text{PAC}_i, \text{CAR}_i$ son variables de tipo real que representan los promedios de temperaturas en cada una de las regiones.

3

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: L_1, L_2, L_3, L_4, L_5 y L_6

$\text{CLA}_1, \text{CAN}_1$

$\text{CLA}_2, \text{CAN}_2$

... ...

0 , 0

Donde: L_1, L_2, L_3, L_4, L_5 y L_6 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:

- Calcule el monto correspondiente de cada venta.
- Obtenga el número de boletos vendidos para cada una de las localidades.
- 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:

- El total producido de cada tipo de vino en los últimos años.
- 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 $\sin(x)$ utilizando la siguiente serie:

$$\sin(x) = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

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 n_1 y n_2 . El MCD entre dos números es el natural más grande que divide a ambos.

Datos: n_1 , n_2 (variables de tipo entero que representan los números que se ingresan).

3

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:

3

```
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.