

# *Programación estructurada y orientada a objetos*

UN ENFOQUE ALGORÍTMICO *3<sup>a</sup> Edición*

LEOBARDO LÓPEZ ROMÁN



 Alfaomega

# ***Programación estructurada y orientada a objetos***

UN ENFOQUE ALGORÍTMICO 3<sup>a</sup> Edición

# ***Programación estructurada y orientada a objetos***

UN ENFOQUE ALGORÍTMICO 3<sup>a</sup> Edición

LEOBARDO LÓPEZ ROMÁN



Datos catalográficos:

López Román, Leobardo  
Programación estructurada y orientada a objetos. Un enfoque algorítmico  
Tercera edición  
Alfaomega Grupo Editor S.A. de C.V., México

**ISBN 978-607-707-211-9**

Formato: 21 x 24 cm

Páginas: 576

**Programación estructurada y orientada a objetos. Un enfoque algorítmico**

Leobardo López Román

Derechos reservados © Alfaomega Grupo Editor, S.A. de C. V., México

Tercera Edición: Alfaomega Grupo Editor, México, agosto de 2011

**© 2011 Alfaomega Grupo Editor, S.A. de C.V.**

Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana

Registro Nº 2317

Página Web: <http://www.alfaomega.com.mx>

E-mail: atencionalcliente@alfaomega.com.mx

**ISBN: 978-607-707-211-9**

**Derechos reservados:**

Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

**Nota importante:**

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V., no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Esta edición puede venderse en todos los países del mundo.

**Impreso en México. Printed in Mexico.**

**Empresas del grupo:**

**Argentina:** Alfaomega Grupo Editor Argentino, S.A.

Paraguay 1307 P.B. “11”, Buenos Aires, Argentina, C.P. 1057

Tel.: (54-11) 4811-7183 / 0887 - E-mail: ventas@alfaomegagroupeditor.com.ar

**México:** Alfaomega Grupo Editor, S.A. de C.V.

Pitágoras 1139, Col. Del Valle, México, D.F., México, C.P. 03100

Tel.: (52-55) 5575-5022 - Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396

E-mail: atencionalcliente@alfaomega.com.mx

**Colombia:** Alfaomega Colombiana S.A.

Carrera 15 No. 64 A 29, Bogotá, Colombia - PBX (57-1) 2100122 - Fax: (57-1) 6068648

E-mail: sccliente@alfaomega.com.mx

**Chile:** Alfaomega Grupo Editor, S.A.

Dr. La Sierra 1437, Providencia, Santiago, Chile

Tel.: (56-2) 235-4248 - Fax: (56-2) 235-5786 - E-mail: agechile@alfaomega.cl

A mis padres Socorro y Celedonio †  
A mi esposa Maricela Villa Acosta  
A mis hijos Leobardo y Eleazar

Leobardo López Román

## Acerca del Autor

Leobardo López Román

Nació en Guamúchil, Sinaloa, México.

Es Licenciado en Informática en Sistemas de Información, egresado del Instituto Tecnológico de Culiacán, Sinaloa, México, en 1981.

En 1982 obtuvo el Diplomado en Ciencias de la Computación; y en 1990 la Maestría en Ciencias de la Computación en la Fundación Arturo Rosenblueth, A.C. En México, D.F.

Profesionalmente se ha desempeñado como programador, programador-analista y analista de sistemas en empresas públicas, privadas, de investigación y educativas.

Desde 1982 ha sido profesor de programación de computadoras en diversas instituciones de educación superior:

- Instituto Tecnológico de Ciudad Guzmán
- Instituto Tecnológico de Culiacán
- Instituto Tecnológico de Hermosillo
- Actualmente es Maestro de Tiempo Completo Titular con perfil PROMEP en la Universidad de Sonora.

Ha recibido el Premio Anual de Profesor Distinguido de la División de Ingeniería de la Universidad de Sonora en 1995, 1998, 2003, 2005 y 2006.

Asimismo ha desempeñado otras actividades académicas, como revisión curricular y diseño y creación de postgrados en Computación e Informática, presidente de academia de Computación y Tecnología de la Información, Coordinador de la Licenciatura en Informática, ha dirigido más de 40 trabajos profesionales para titulación, y ha dictado conferencias a nivel nacional e internacional.

Es autor de los libros:

- "Metodología de la Programación Orientada a Objetos", Alfaomega, México, 2006.
- "Programación Estructurada en Lenguaje C", Alfaomega, México, 2005.
- "Programación Estructurada un enfoque algorítmico 2da. Edición", Alfaomega, México, 2003.
- "Programación Estructurada en Turbo Pascal 7", Alfaomega, México, 1998.
- "Programación Estructurada un enfoque algorítmico", Alfaomega, México, 1994.

## Mensaje del Editor

Los conocimientos son esenciales en el desempeño profesional. Sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecerles a estudiantes y profesionales conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (IT) para facilitar el aprendizaje. Libros como éste tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales, información actualizada, pruebas (test) de autoevaluación, diapositivas y vínculos con otros sitios Web relacionados.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento.

Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y metas propuestas. Cada capítulo concluye con diversas actividades pedagógicas para asegurar la asimilación del conocimiento y su extensión y actualización futuras.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos guía en diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.

## Agradecimientos

A Dios, que es mi guía e inspiración, le doy las gracias por su bondad y generosidad al darme salud, fuerza de voluntad y capacidad para desarrollar mi obra editorial.

Mi gratitud para esa gran y noble institución que es la Universidad de Sonora, que en su seno he encontrado la oportunidad de desarrollar mi obra editorial; la cual, ya es parte de su esencia misma.

También quiero agradecer a mis compañeros maestros y alumnos del Departamento de Ingeniería Industrial y de Sistemas de la Universidad de Sonora; y del Instituto Tecnológico de Hermosillo quienes de una u otra forma han contribuido para que mi obra editorial sea una realidad.

Asimismo agradezco a mis maestros y amigos Manuel Sáiz y María Luisa López López (Malichi) sus consejos y su guía.

Un agradecimiento muy especial para mi esposa Maricela Villa Acosta, y para mis hijos Leobardo y Eleazar, a quienes les he quitado mucho tiempo y paciencia para desarrollar esta obra.

Y especialmente a Usted, amigo lector, que me ha dado la oportunidad,quiero serle útil a través de mi obra editorial: En primera instancia a través de mis libros anteriores, ahora a través de este libro; y, en un futuro espero poner a su disposición un libro que implemente esta metodología en lenguaje C, C++, Java u otro.

Indudablemente habrán quedado errores u omisiones las cuales espero subsanar en una futura edición, así que, amable lector, si tiene alguna sugerencia para el mejoramiento del mismo, mucho le agradeceré me la haga llegar. Gracias.

*"Escribir un libro, es el más alto honor de un maestro"*



Leobardo López Román

Universidad de Sonora  
Hermosillo, Sonora, México  
llopez@industrial.uson.mx

# Contenido

Convenciones utilizadas en el texto.....	10
Registro en la Web de apoyo.....	11
Prefacio.....	13
<b>1. Introducción a la programación.....</b>	<b>19</b>
1.1 Conceptos generales.....	21
1.2 Evolución de los paradigmas de programación.....	27
1.3 El proceso de programación.....	31
1.4 El algoritmo.....	34
1.5 Ejercicios propuestos .....	37
1.6 Resumen de conceptos que debe dominar.....	38
1.7 Contenido de la página Web de apoyo El material marcado con asterisco (*) sólo está disponible para docentes.	
1.7.1 Resumen gráfico del capítulo	
1.7.2 Autoevaluación	
1.7.3 Power Point para el profesor (*)	
<b>2. Elementos para solucionar problemas en seudocódigo .....</b>	<b>39</b>
2.1 Estructuras de datos.....	41
2.2 Operaciones primitivas elementales .....	43
2.3 Estructuras de control .....	50
2.4 Resumen de conceptos que debe dominar.....	50
2.5 Contenido de la página Web de apoyo El material marcado con asterisco (*) sólo está disponible para docentes.	
2.5.1 Resumen gráfico del capítulo	
2.5.2 Autoevaluación	
2.5.3 Power Point para el profesor (*)	
<b>3. La secuenciación.....</b>	<b>51</b>
3.1 Nuestro primer problema.....	53
3.2 Estructura y diseño de un algoritmo .....	53
3.3 Nuestro primer algoritmo .....	55
3.4 Funciones matemáticas.....	56
3.5 Ejercicios resueltos .....	62
3.6 Ejercicios propuestos .....	65
3.7 Resumen de conceptos que debe dominar.....	67
3.8 Contenido de la página Web de apoyo El material marcado con asterisco (*) sólo está disponible para docentes.	
3.8.1 Resumen gráfico del capítulo	
3.8.2 Autoevaluación	
3.8.3 Programas	
3.8.4 Power Point para el profesor (*)	
<b>4. La selección .....</b>	<b>69</b>
4.1 La selección doble (if-then-else).....	71
4.1.1 Sangrado (indentación) y etiquetas.....	74
4.1.2 Expresiones lógicas .....	75
4.1.3 If's anidados.....	78
4.1.4 Ejercicios resueltos para la selección doble (if-then-else) .....	81
4.2 La selección simple (if-then) .....	86
4.2.1 Ejercicios resueltos para la selección simple (if-then) .....	87
4.3 La selección múltiple (switch) .....	90
4.3.1 Ejercicio resuelto para la selección múltiple (switch) .....	94
4.4 Ejercicios propuestos .....	95
4.5 Resumen de conceptos que debe dominar.....	99
4.6 Contenido de la página Web de apoyo El material marcado con asterisco (*) sólo está disponible para docentes.	
4.6.1 Resumen gráfico del capítulo	
4.6.2 Autoevaluación	
4.6.3 Programas	
4.6.4 Ejercicios resueltos	
4.6.5 Power Point para el profesor (*)	
<b>5. La repetición .....</b>	<b>101</b>
5.1 La repetición do...while .....	103
5.1.1 Contadores y acumuladores .....	105
5.1.2 Ejercicios resueltos para la repetición do...while .....	108
5.2 Ejercicios propuestos para la repetición do... while .....	114
5.3 La repetición for .....	123
5.3.1 for anidados .....	127
5.3.2 Ejercicios resueltos para la repetición for .....	127
5.3.3 Simulación del for con do...while .....	135
5.4 Ejercicios propuestos para la repetición for.....	136
5.5 La repetición while.....	144
5.5.1 Simulación del do...while con while.....	146
5.5.2 Simulación del for con while .....	147
5.5.3 Ejercicios resueltos para la repetición while .....	150
5.6 Ejercicios propuestos para la repetición while.....	155
5.7 Resumen de conceptos que debe dominar.....	158

5.8	Contenido de la página Web de apoyo	240
	El material marcado con asterisco (*) sólo está disponible para docentes.	
5.8.1	Resumen gráfico del capítulo	
5.8.2	Autoevaluación	
5.8.3	Programas	
5.8.4	Ejercicios resueltos	
5.8.5	Power Point para el profesor (*)	
<b>6.</b>	<b>Arreglos .....</b>	<b>159</b>
6.1	Arreglos unidimensionales .....	161
6.1.1	Ejercicios resueltos para unidimensionales .....	165
6.2	Arreglos bidimensionales.....	169
6.2.1	Ejercicios resueltos para bidimensionales .....	173
6.3	Arreglos tridimensionales.....	179
6.3.1	Ejercicios resueltos para tridimensionales .....	183
6.4	Arreglos tetradimensionales .....	183
6.4.1	Ejercicios resueltos para tetradimensionales .....	188
6.5	Ejercicios propuestos .....	189
6.6	Resumen de conceptos que debe dominar.....	197
6.7	Contenido de la página Web de apoyo	
	El material marcado con asterisco (*) sólo está disponible para docentes.	
6.7.1	Resumen gráfico del capítulo	
6.7.2	Autoevaluación	
6.7.3	Programas	
6.7.4	Ejercicios resueltos	
6.7.5	Power Point para el profesor (*)	
<b>7.</b>	<b>Diseño descendente (Top Down Design) .....</b>	<b>199</b>
7.1	Proceso de modularización .....	201
7.2	Forma de utilizar el diseño descendente con seudocódigo .....	204
7.3	Funciones que no regresan valor (void) .....	205
7.4	Variables globales, locales y parámetros.....	211
7.4.1	Variables globales .....	211
7.4.2	Variables locales .....	212
7.4.3	Parámetros.....	216
7.4.3.1	Parámetro por referencia .....	216
7.4.3.2	Parámetro por valor .....	218
7.5	Funciones estándar .....	220
7.5.1	Funciones cadena de caracteres .....	220
7.5.2	Validación de la entrada de datos.....	225
7.5.3	Funciones especiales.....	227
7.6	Funciones que regresan valor.....	229
7.7	Ejercicios resueltos .....	232
7.8	Ejercicios propuestos .....	237
7.9	Resumen de conceptos que debe dominar.....	240
7.10	Contenido de la página Web de apoyo	
	El material marcado con asterisco (*) sólo está disponible para docentes.	
7.10.1	Resumen gráfico del capítulo	
7.10.2	Autoevaluación	
7.10.3	Programas	
7.10.4	Ejercicios resueltos	
7.10.5	Power Point para el profesor (*)	
<b>8.</b>	<b>Registros y archivos .....</b>	<b>241</b>
8.1	Organización de archivos .....	246
8.2	Manejo de registros en seudocódigo .....	248
8.3	Operaciones para el manejo de archivos en seudocódigo .....	259
8.4	Proceso de un archivo secuencial .....	266
8.5	Proceso de un archivo directo .....	285
8.6	Ejercicios resueltos .....	297
8.7	Ejercicios propuestos .....	297
8.8	Resumen de conceptos que debe dominar.....	312
8.9	Contenido de la página Web de apoyo	
	El material marcado con asterisco (*) sólo está disponible para docentes.	
8.9.1	Resumen gráfico del capítulo	
8.9.2	Autoevaluación	
8.9.3	Programas	
8.9.4	Ejercicios resueltos	
8.9.5	Power Point para el profesor (*)	
<b>9.</b>	<b>Otros tipos de datos y otros temas .....</b>	<b>313</b>
9.1	Clasificación (ordenación) de datos.....	315
9.2	Tipos de datos definidos por el usuario (Tipos).....	323
9.3	Apuntadores (Pointer) .....	327
9.4	Recursividad en seudocódigo .....	334
9.5	Ejecución de otros programas en código ejecutable .....	336
9.6	Graficación en seudocódigo.....	337
9.7	Incluir archivos de programas.....	339
9.8	Resumen de conceptos que debe dominar.....	340
9.9	Contenido de la página Web de apoyo	
	El material marcado con asterisco (*) sólo está disponible para docentes.	
9.9.1	Resumen gráfico del capítulo	
9.9.2	Autoevaluación	
9.9.3	Programas	
9.9.4	Power Point para el profesor (*)	

<b>10. Programación orientada a objetos usando el diagrama de clases .....</b>	341
10.1 Objetos .....	343
10.1.1 Qué son los objetos .....	343
10.1.2 Cómo están formados los objetos .....	344
10.1.3 Cuándo y cómo identificar los objetos.....	344
10.2 Clases y su relación con los objetos.....	346
10.2.1 Determinar las clases.....	346
10.2.2 Representación de la clase y sus instancias.....	347
10.3 Métodos y encapsulación.....	348
10.3.1 Métodos .....	348
10.3.2 Encapsulación.....	348
10.4 Diseño del diagrama de clases.....	349
10.4.1 Modificadores de acceso (visibilidad) .....	350
10.5 Generar instancias de una clase.....	353
10.6 Arquitectura modelo-vista-controlador.....	354
10.7 Resumen de conceptos que debe dominar.....	356
10.8 Contenido de la página Web de apoyo	
El material marcado con asterisco (*) sólo está disponible para docentes.	
10.8.1 Resumen gráfico del capítulo	
10.8.2 Video sobre instalación de Eclipse	
10.8.3 Autoevaluación	
10.8.4 Programas	
10.8.5 Power Point para el profesor (*)	
<b>11. Programación orientada a objetos aplicando la estructura de secuenciación.....</b>	357
11.1 Nuestro primer problema.....	359
11.2 Diseño de algoritmos OO usando la secuenciación en seudocódigo .....	361
11.3 Constructores y destructores .....	371
11.4 Ejercicios resueltos .....	372
11.5 Ejercicios propuestos .....	382
11.6 Resumen de conceptos que debe dominar.....	382
11.7 Contenido de la página Web de apoyo	
El material marcado con asterisco (*) sólo está disponible para docentes.	
11.7.1 Resumen gráfico del capítulo	
11.7.2 Autoevaluación	
11.7.3 Programas	
11.7.4 Ejercicios resueltos	
11.7.5 Power Point para el profesor (*)	
<b>12. Programación orientada a objetos aplicando las estructuras de selección .....</b>	383
12.1 Diseño de algoritmos OO usando la selección doble (if then else).....	385
12.2 Diseño de algoritmos OO usando la selección simple (if then).....	389
12.3 Diseño de algoritmos OO usando la selección múltiple (switch) .....	392
12.4 Ejercicios resueltos.....	394
12.5 Ejercicios propuestos .....	404
12.6 Resumen de conceptos que debe dominar.....	404
12.7 Contenido de la página Web de apoyo	
El material marcado con asterisco (*) sólo está disponible para docentes.	
12.7.1 Resumen gráfico del capítulo	
12.7.2 Autoevaluación	
12.7.3 Programas	
12.7.4 Ejercicios resueltos	
12.7.5 Power Point para el profesor (*)	
<b>13. Programación orientada a objetos aplicando las estructuras de repetición .....</b>	405
13.1 Diseño de algoritmos OO usando la repetición do...while.....	407
13.2 Contadores y acumuladores.....	410
13.2.1 Ejercicios resueltos para do...while .....	414
13.3 Ejercicios propuestos para do...while .....	427
13.4 Diseño de algoritmos OO usando la repetición for ...428	
13.4.1 Ejercicios resueltos para for.....	430
13.5 Ejercicios propuestos para for .....	435
13.6 Diseño de algoritmos OO usando la repetición while.....	435
13.6.1 Ejercicios resueltos para while.....	439
13.7 Ejercicios propuestos para while .....	451
13.8 Resumen de conceptos que debe dominar.....	451
13.9 Contenido de la página Web de apoyo	
El material marcado con asterisco (*) sólo está disponible para docentes.	
13.12.1 Resumen gráfico del capítulo	
13.12.2 Autoevaluación	
13.12.3 Programas	
13.12.4 Ejercicios resueltos	
13.12.5 Power Point para el profesor (*)	
<b>Apéndices</b>	
A. Algoritmos sin usar etiquetas .....	453
B. Diagramas de flujo .....	459
C. Diagramas Warnier.....	525
D. Diagramas Chapin (Nassi-Schneiderman) .....	543
E. Seudocódigo castellanizado (español estructurado)....559	
<b>Bibliografía.....</b>	568

## Convenciones utilizadas en el texto

	Conceptos para recordar: bajo este ícono se encuentran definiciones importantes que refuerzan lo explicado en la página.
	Comentarios o información extra: este ícono ayuda a comprender mejor o ampliar el texto principal
	Contenidos interactivos: indica la presencia de contenidos extra en la Web.

## Registro en la Web de apoyo

Para tener acceso al material de la página Web de apoyo del libro:

1. Ir a la página <http://virtual.alfaomega.com.mx>
2. Registrarse como usuario del sitio y propietario del libro.
3. Ingresar al apartado de inscripción de libros y registrar la siguiente clave de acceso
  
4. Para navegar en la plataforma virtual de recursos del libro, usar los nombres de Usuario y Contraseña definidos en el punto número dos. El acceso a estos recursos es limitado. Si quiere un número extra de accesos, escriba a [webmaster@alfaomega.com.mx](mailto:webmaster@alfaomega.com.mx)

Estimado profesor: Si desea acceder a los contenidos exclusivos para docentes, por favor contacte al representante de la editorial que lo suele visitar o escribanos a:

[webmaster@alfaomega.com.mx](mailto:webmaster@alfaomega.com.mx)

## Prefacio

---

Usted tiene en sus manos un libro que contiene la metodología de la programación de computadoras conocida como programación estructurada; y su evolución al paradigma conocido como programación orientada a objetos. La metodología se presenta en dos fases.

En la primera fase, la metodología está basada en el uso de un pseudolenguaje llamado seudocódigo integrado con la técnica diseño descendente (Top Down Design). Dicha metodología permite diseñar programas bien estructurados, bien documentados eficaces, eficientes y fáciles de darles mantenimiento. El método, ha sido plasmado en forma que conduce el proceso enseñanza-aprendizaje de la programación de computadoras en forma didáctica, simple, completa, consistente y práctica con una gran cantidad y variedad de ejercicios cuidadosamente seleccionados y probados en clase por el autor; y, que va desde un nivel de principiante, pasando por intermedio y deja las bases para el nivel avanzado.

Lo relevante de este método es que enseña a programar computadoras utilizando un pseudolenguaje, es decir, sin utilizar la computadora. Esto permite desarrollar las capacidades mentales que una persona debe tener para programar computadoras y sienta las bases de disciplina y buena estructura. Este enfoque se le dificulta a mucha gente, sin embargo, hay que enfrentarlo, porque siendo la programación una actividad intelectual que requiere mucha creatividad, capacidad de abstracción, de análisis y de síntesis; éstas, no se pueden desarrollar operando un lenguaje en la computadora, sino ejercitando la mente en forma apropiada.

Se ha logrado una metodología de la programación que contiene en forma natural los conceptos, estructuras, filosofía y postulados de la programación estructurada, rescatando las bases de la programación en las que se sustenta, y, que han hecho posible los nuevos avances en la programación como lo es la programación orientada a objetos.

En la segunda fase, se presenta la evolución de la programación estructurada a la programación orientada a objetos. Esta metodología es el resultado de la integración y adaptación de varias técnicas, como son; los conceptos y estructuras de la programación orientada a objetos: objetos, clases, encapsulación; el diagrama de clases de UML (Unified Modeling Language, desarrollado

por G. Booch, I. Jacobson y J. Rumbaugh); la arquitectura modelo-vista-controlador; algunos conceptos introducidos por el lenguaje Java; y los conceptos y bases lógicas de la programación estructurada en seudocódigo, que se tratan en la primera fase de este libro.

En la actualidad se ha observado que mucha gente desea aprender la programación orientada a objetos dejando de lado la programación estructurada, esto es un error, porque la programación estructurada es la base de la programación orientada a objetos; y, si deseamos aprender la programación orientada a objetos, primero debemos dominar la programación estructurada.

## Haciendo un poco de historia

---

En los años 50 surgió la programación de computadoras con lenguajes de alto nivel; las estructuras de control que manejaban estos primeros lenguajes (como por ejemplo FORTRAN) eran muy limitadas. Poco a poco se fué ampliando y diversificando el uso de la computadora gracias al desarrollo tecnológico y la aparición de otros lenguajes como COBOL , PL/1, etcétera; al aumentar en tamaño y complejidad las aplicaciones que se desarrollaban, empezaron a presentarse muchos problemas en el desarrollo y mantenimiento de los programas, provocados por su inadecuada estructura y documentación. Por otro lado, el proceso enseñanza-aprendizaje de la programación de computadoras era una actividad extremadamente difícil. En estos tiempos los lenguajes de programación sólo soportaban la estructura de repetición FOR (DO en FORTRAN). A raíz de esto, se gestó una revolución en la programación y se añadieron las estructuras de control DO-UNTIL y DOWHILE; además se demostró la importancia de que los programas fueran estructurados en pequeños módulos. Con la inclusión de estos nuevos conceptos

La programación de computadoras se transformó en PROGRAMACION ESTRUCTURADA. Entre los padres de la programación estructurada destacan Dijkstra, Mills, Hoare, Kernigan, Plauger, Yourdon, Warnier, Searson, Wirth, entre otros. Los postulados de la Programación Estructurada son:

- Toda persona debe aprender a programar utilizando un seudolenguaje estructurado.

- Se deben usar sólo las estructuras de control estructuradas: SEQUENCE, IF-THEN, IF-THEN-ELSE, CASE, DO-UNTIL, FOR, DOWHILE.
- Los programas deben ser organizados en pequeños módulos.
- Se debe utilizar el lenguaje Pascal como prototipo para el aprendizaje de estos conceptos, es decir, como primer lenguaje.
- Donde se aplique la programación se debe hacer énfasis en el diseño de los programas antes de codificar.
- Seguir un estilo que haga más entendible el algoritmo (y el programa), como usar nombres de variables apropiados, dejar sangría para denotar subordinación, entre otros.

## Problemática de la enseñanza de la programación en los 80 y 90

En los 90, todo mundo quería hacer y hablar sobre programación orientada a objetos, sin embargo, se observó que en realidad la programación estructurada no penetró en las instituciones de educación. En la mayor parte de las escuelas, universidades e institutos donde se enseña la programación, se incurria en alguno de estos errores:

1. Enseñaban a programar directamente con algún lenguaje; aunque éste fuera el Pascal no es bueno enseñar la lógica directamente con un lenguaje por el doble problema que significa aprender la lógica de programación y la sintaxis del lenguaje al mismo tiempo.
2. Enseñaban la lógica con diagramas de flujo; los cuales resultan obsoletos porque no soportan todas las estructuras de control de la programación estructurada en forma natural.
3. Enseñaban la lógica en pseudocódigos que utilizan las estructuras lógicas de control en forma castellanizada, es decir, en lugar de IF-THEN-ELSE utilizan SI-ENTONCES-SINO, en lugar de FOR utilizan PARA, en lugar de DOWHILE utilizan MIENTRAS; he usado estos conceptos y he comprobado que no es buena idea. Las estructuras de la programación estructurada tienen sus palabras reservadas originales y no deben ser cambiadas, porque causan mucha confusión al estudiar los lenguajes de programación.

4. Se utilizaba C (o C++) como primer lenguaje; que permite la programación estructurada pero no es un lenguaje estructurado en forma natural, además de que no es un lenguaje orientado para principiantes.
5. En las materias en las que se aplica la programación (estructura de datos, base de datos, etcétera), los maestros hacen énfasis en el código del programa y no en el diseño del mismo; lo cual es esencial en la programación estructurada.

Otro gran problema con la enseñanza de la programación estructurada es el gran ruido que ha causado la programación orientada a objetos; todo mundo habla de ella y se ha vendido la idea de que ha venido a reemplazar a la programación estructurada; y que debe enseñarse la programación desde un inicio con lenguajes orientados a objetos como Java, C++, etcétera. Sin embargo, la programación orientada a objetos es una herramienta que se le ha añadido a la programación estructurada, es decir, no ha venido a reemplazarla, sino que se basa en la estructurada y va más allá, pero se nutre de ésta. Porque las estructuras de la programación estructurada: Tipos de datos; entero, real, cadena, carácter, arreglos, registros y archivos; Estructuras de control; secuenciación, if-then, if-then-else, case, do-while, for, while; módulos y funciones; también son los elementos básicos de la programación orientada a objetos. Así que los conceptos y filosofía de la programación estructurada están hoy más vigentes que nunca.

Ahora en que se nos dice que debemos correr a través del uso de la programación orientada a objetos, visual, etcétera, yo les digo ¡¡¡ALTO!!!, primero tenemos que reconocer que no hemos aprendido a caminar a través de la asimilación adecuada de la programación estructurada: ¡Debemos primero aprender a caminar para poder aprender a correr!

Este autor ha desarrollado una metodología que viene a coadyuvar en la solución de la problemática antes referida, la cual ha sido publicada en el libro intitulado Programación estructurada un enfoque algorítmico en su primera y segunda edición. Ahora se presenta la tercera edición.

## Mejoras de esta tercera edición

Las mejoras realizadas en esta tercera edición se dividen en tres partes:

### Primera:

En la metodología de la programación presentada en la segunda edición, las estructuras lógicas de control: Secuenciación, IF THEN, IF THEN ELSE, CASE, DO UNTIL, FOR y DOWHILE, se utilizan en el formato en que originalmente fueron inventadas. Sin embargo, en lenguaje C la estructura CASE, se implementa como switch; la estructura DO UNTIL se implementa como do...while; y la estructura DOWHILE se implementa como while. Considerando que las estructuras lógicas del lenguaje C son la base de C++, Java y de la mayoría de los lenguajes actuales, es que la forma en que las implementa el lenguaje C, se han convertido prácticamente en el estándar de las estructuras de control. Esto, en cierta medida ha debilitado y desactualizado a la metodología de la segunda edición.

En esta tercera edición, se le han hecho ajustes a la metodología para actualizarla y fortalecerla, a continuación se describen:

1. Los identificadores como nombres de variables, funciones, etcétera; ahora se manejan con un estilo más actualizado.
2. La estructura CASE del capítulo cuatro de la segunda edición, ahora se implementa como switch.
3. La estructura DO UNTIL del capítulo cinco de la segunda edición, ahora se implementa como do...while.
4. La estructura DOWHILE del capítulo siete de la segunda edición, ahora se implementa como while.

Los cambios anteriores, implicaron hacer ajustes en todos los capítulos, de tal manera que la metodología ha sido reestructurada para quedar acorde con la forma en que los lenguajes actuales implementan las estructuras de control. Logrando una metodología de la programación estructurada más actualizada, completa y consistente.

### Segunda:

Se han agregado cuatro capítulos en los que se presenta la programación orientada a objetos, conjuntando lo que es pertinente de los primeros siete capítulos, con

los conceptos de objetos, clases y encapsulación, planteando una evolución natural de la programación estructurada a la orientada a objetos.

Cabe aclarar que la programación orientada a objetos se trata de una forma introductoria y hasta un nivel medio, si desea mayor profundidad, puede hacerlo estudiando el libro Metodología de la programación orientada a objetos, de este mismo autor y publicado por esta misma editorial, que actualmente está en su primera edición, y en un futuro se publicará la segunda edición; en el cual se presenta la metodología de la programación orientada a objetos en forma más completa y con mayor profundidad.

### Tercera:

Aunque el propósito de este libro no es estudiar ningún lenguaje de programación; en la dirección <http://virtual.alfaomega.com.mx> que es una web de ayuda que la editorial Alfaomega ha puesto a su disposición, podrá encontrar los programas correspondientes de los algoritmos del libro, codificados en lenguaje C los de los capítulos 3 al 9, que corresponde a la programación estructurada; y en Java, los de los capítulos del 3 al 7 y del 11 al 13. Cabe aclarar, que no obstante que Java es un lenguaje orientado a objetos, es posible programar en Java los algoritmos desarrollados mediante la programación estructurada. Los de los capítulos 8 y 9 no se presentan en Java porque va más allá del propósito y nivel de este libro. Además, en la web del libro, se encuentran ejercicios resueltos de los capítulos 3 al 8 y del 11 al 13.

## Problemática actual y reivindicación de la programación estructurada

Hasta la aparición del lenguaje Java, la enseñanza de la programación se realizaba primero usando alguna metodología de la programación estructurada y luego el lenguaje C o C++ en la parte estructurada; después se enseñaba la programación orientada a objetos en C++. Cuando aparece Java, este empieza a enseñarse como un segundo o tercer lenguaje; ya sea después de C o después de C++. Sin embargo, en los últimos años, en muchas instituciones de educación adoptaron Java como primer lenguaje, eliminando la programación estructurada, el lenguaje C y C++.

Actualmente, cada día son más las instituciones que han comprobado que enseñar Java como primer lenguaje no ha sido una buena idea; y se han dado cuenta que es mejor enseñar primero la programación estructurada implementada en C o C++, y después de esas bases, enseñar la programación orientada a objetos en Java. Esto, está llevando a reivindicar al paradigma de la programación estructurada, como lo que es, la base en la que se sustenta la programación orientada a objetos, así que, la programación estructurada esta hoy más vigente que nunca. Es por ello que he decidido revitalizar esta metodología y además, añadirle su evolución natural hacia la programación orientada a objetos. Lo cual ha permitido desarrollar esta nueva edición, logrando una metodología más actualizada, completa y consistente.

## Organización del libro

El material de la presente obra esta dividido en trece capítulos y cinco apéndices. En los capítulos del 1 al 9, es lo que se considera como la primera parte del libro, aquí se presenta la metodología de la programación estructurada.

En el capítulo uno se presenta una introducción a la programación. En el capítulo dos se presentan los elementos para solucionar problemas en seudocódigo, como son tipos de datos, identificadores, variables, operaciones aritméticas, lectura y escritura de datos, etcétera. En el capítulo tres se estudia la estructura de control secuenciación, su definición, estructura de un algoritmo y algunos ejemplos. El capítulo cuatro trata la selección simple,

doble y múltiple, su definición, formato en seudocódigo y utilización con ejercicios resueltos y propuestos.

En el capítulo cinco se explican las estructuras de repetición do...while, for y while; su definición, formato, uso, ejercicios resueltos y propuestos.

En el capítulo seis se presentan los arreglos unidimensionales, bidimensionales, tridimensionales y tetradimensionales; su definición, formato, manipulación, uso, ejercicios resueltos y propuestos.

En el capítulo siete se trata la modularización de programas a través del diseño descendente; funciones que no regresan valor (módulos), funciones que regresan valor (funciones definidas por el usuario), el paso de parámetros entre funciones y algunas otras funciones estándar.

El capítulo ocho trata lo referente a registros y archivos, donde se presenta definiciones, organización de archivos, como manejarlos en seudocódigo, proceso de un archivo secuencial, proceso de un archivo directo, obtención de reportes, ejercicios resueltos y propuestos.

El capítulo nueve, como si fuera un apéndice, en el que se presentan una serie de temas sueltos: Clasificación (ordenación) de datos, definición de tipos de datos, apuntadores, recursividad, ejecución de otros programas ejecutables y graficación, donde cada tema puede estudiarse en el momento en que se considere necesario. Y, permiten que la metodología de la programación estructurada se utilice en temas más avanzados de programación.

A partir del capítulo diez se explica lo que se considera como la segunda parte del libro, es decir, se presenta la metodología integrando algunos conceptos y estructuras de la programación orientada a objetos: objetos, clases, encapsulación; el diagrama de clases de UML (Unified Modeling Language); la arquitectura modelo-vista-controlador; algunos conceptos introducidos por el lenguaje Java; y los conceptos y bases lógicas de la programación estructurada en seudocódigo estudiados en los capítulos del 1 al 7. Dicha metodología permite diseñar programas o algoritmos orientados a objetos, bien estructurados, bien documentados eficaces, eficientes y fáciles de darles mantenimiento.

En el capítulo diez, se estudian los conceptos de objetos, clases, métodos y encapsulación, y se explica la forma de cómo utilizarlos para diseñar el diagrama de clases. Asimismo se involucra el uso de la arquitectura modelo-vista-controlador.

En el capítulo once, se estudia cómo diseñar la lógica de cada una de las clases usando seudocódigo, incorporando en esta técnica los conceptos de clases, objetos y encapsulación. Se presenta el diseño de algoritmos orientados a objetos aplicando la estructura de secuenciación con ejemplos que se estudiaron en el capítulo tres.

En el capítulo doce se presenta el diseño de algoritmos orientados a objetos aplicando las estructuras de selección if then else, if then y switch, con ejemplos que se estudiaron en el capítulo cuatro.

En el capítulo trece se presenta el diseño de algoritmos orientados a objetos aplicando las estructuras de repetición do...while, for y while, con ejemplos que se estudiaron en el capítulo cinco.

En el apéndice A se presentan algunos algoritmos sin usar etiquetas. En el apéndice B se muestran algunos de los algoritmos en diagramas de flujo, de tal manera que al estudiar los algoritmos en seudocódigo, podrá auxiliarse de los diagramas de flujo correspondientes, y así facilitar su comprensión. En el apéndice C se explica otro método alternativo de diseño de programas usando los Diagramas Warnier. En el apéndice D se expone otro método para el diseño de programas usando los Diagramas Chapin (Nassi-Schneiderman). Y en el apéndice E se presentan algunos algoritmos usando una versión de seudocódigo castellanizado o español estructurado. A quien va dirigido

Este trabajo requiere que el lector tenga conocimientos previos sobre conceptos generales relacionados con las computadoras, en caso de no ser así, se recomienda la lectura de algún libro que sirva como introducción a las computadoras, a las tecnologías de la información o a la informática.

El libro puede ser utilizado como texto o consulta en materias de fundamentos de programación, metodología de la programación, programación de computadoras y similares, que se cursan en los primeros dos o tres semestres de carreras como Informática, Informática administrativa, Computación, Ciencias de la computación, Sistemas Computacionales, Sistemas de Información, Ingeniería Industrial y de Sistemas, Ingeniería Mecatrónica, Ingeniería Electrónica, Ingeniería Eléctrica, entre otras; también se puede usar en preparatorias, bachilleratos y carreras de nivel técnico.

## Advertencia didáctica

Se ha tomado el ejemplo de calcular sueldo(s) de empleado(s) -pago de sueldo o nómina- como un problema pivote para facilitar la explicación de las estructuras que integran la metodología. Esto quiere decir, que al estudiar cada elemento o estructura, el primer problema que se plantea es el de pago de sueldo, primero de una manera muy simple, y luego, con una variante apropiada para la aplicación de lo que se está explicando en cada momento. Esto es con fines didácticos, ya que es una situación fácilmente entendible, que permite al estudiante (lector) familiarizarse con ella porque se trata desde el inicio y durante todo el libro; y así podemos dedicar todo nuestro esfuerzo mental al aspecto lógico de la metodología. Es decir, que los problemas no deben ser muy complicados, para dirigir todo nuestro esfuerzo mental a comprender la lógica de la metodología y no a entender problemas innecesariamente complejos.

En consecuencia, es probable que el lector, perciba que este es un libro con una orientación administrativa, sin embargo, después de explicarle cada estructura con el ejemplo antes referido; en los ejercicios resueltos, se presenta la aplicación de la estructura con otros tipos de problemas, dándole a la metodología un enfoque de aplicación general, es decir, tanto para el área administrativa, como para la ingeniería.

## Resumiendo

Aprender a programar no es fácil ni rápido; es un proceso que debe iniciar con el desarrollo de la lógica usando un pseudolenguaje de diseño de programas o algoritmos. Con el estudio de la metodología y fundamentos de programación que le presento en este libro, el estudiante aprenderá la lógica de la programación estructurada y una introducción a la programación orientada a objetos sin estar "casado" con ningún lenguaje; y de aquí en adelante podrá aprender y comprender cualquier lenguaje estructurado u orientado a objetos como C, C++, Java, C#, UML, etcétera.

# 1

## Introducción

### Contenido

- 1.1 Conceptos generales
- 1.2 Evolución de los paradigmas de programación
- 1.3 El proceso de programación
- 1.4 El algoritmo
- 1.5 Ejercicios propuestos
- 1.6 Resumen de conceptos que debe dominar
- 1.7 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*)  
sólo está disponible para docentes.
  - 1.7.1 Resumen gráfico del capítulo
  - 1.7.2 Autoevaluación
  - 1.7.3 Power Point para el profesor (\*)

### Objetivos del capítulo

- Repasar conceptos generales de computación.
- Comprender los fundamentos del funcionamiento de la computadora y sus componentes.
- Comprender el concepto de programa.
- Entender las características principales de un lenguaje de programación.
- Entender cuáles son las características de un buen programa.
- Conocer los diversos paradigmas de programación.
- Entender cuáles son los pasos que integran el proceso de programación.
- Entender qué es el algoritmo, cuáles son las características que tiene y aplicar los conceptos aprendidos en situaciones de la vida cotidiana.

## Introducción

En este primer capítulo el lector efectuará un repaso de los principales conceptos acerca de las computadoras. Se presenta la computadora como una herramienta que se utiliza para resolver problemas en el accionar cotidiano de las empresas, organizaciones o instituciones. Esto se hace mediante el esquema del procesamiento electrónico de datos que es E-P-S (Entrada-Proceso-Salida), es decir: entran datos como materia prima, luego se procesan para transformarlos en información que se emite como salida.

Se estudian también los elementos funcionales básicos que componen una computadora, que son: la unidad central de proceso, la unidad de memoria, la unidad de entrada y la unidad de salida. Cabe aclarar que es deseable que estos conceptos ya los domine el estudiante, o bien, que los estudie en algún otro libro de introducción a la computación, a la informática o tecnologías de la información.

Se expone el concepto de programa, que es un conjunto de instrucciones que guían a la computadora para realizar alguna actividad o resolver algún problema. También se detalla que el programa se compone de estructuras de datos, operaciones primitivas elementales y estructuras de control.

Se explica que un lenguaje de programación es el medio a través del cual le comunicamos a la computadora la secuencia de instrucciones que debe ejecutar para llevar a cabo actividades, tareas o para solucionar problemas. Además se expone que todo lenguaje está compuesto por un alfabeto, un vocabulario y una gramática, luego se revisa el concepto de lo que es la programación y se enumeran las características de un buen programa.

Se presenta la evolución de los paradigmas de programación, que inicia con las primeras estructuras que se inventaron cuando aparece la programación tradicional, luego, sobre esas bases se gestó la programación estructurada, para dar lugar a la programación modular, enseguida se agrega la programación con abstracción de datos, para llegar al desarrollo de la programación orientada a objetos.

Se detallan los pasos que integran el proceso de programación: definición del problema, análisis del problema, diseño del programa, codificación del programa, implantación del programa y mantenimiento del programa.

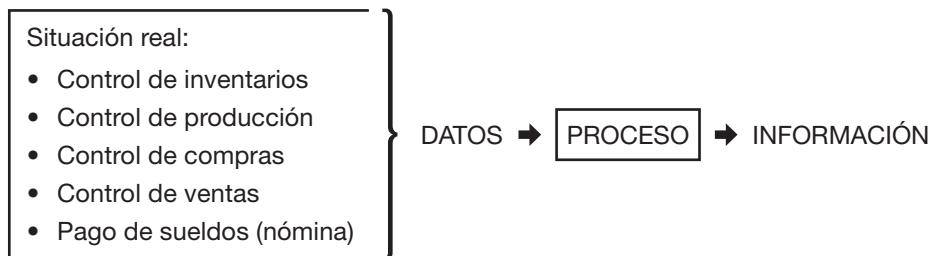
Se define que el algoritmo es una secuencia de pasos que llevan a la solución de un problema o a la ejecución de una tarea o actividad. Se plantea que los pasos del algoritmo deben tener las siguientes características: ser simples, claros, precisos, exactos; tener un orden lógico; tener un principio y un fin.

En el siguiente capítulo se estudian los elementos para solucionar problemas en pseudocódigo.

## 1.1 Conceptos generales

### La computadora

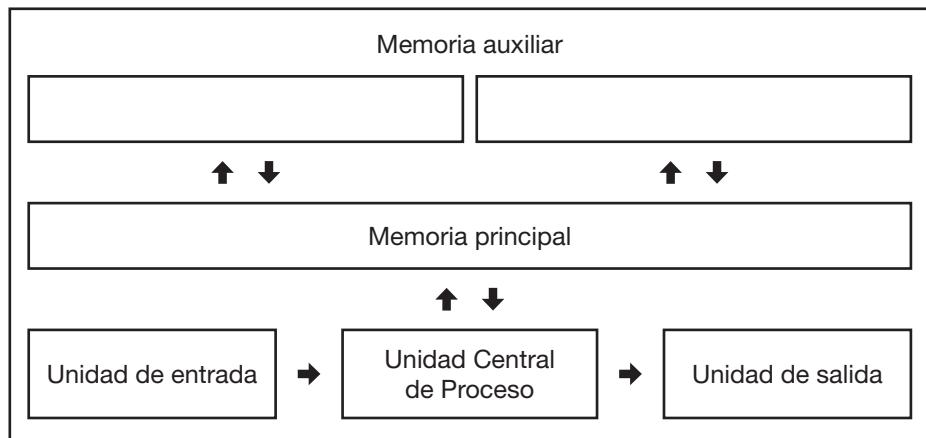
La computadora es una herramienta que se utiliza para representar cualquier situación de la realidad en forma de datos, los cuales se procesan después para generar información, esquemáticamente:



Esto quiere decir que toda situación que pueda ser abstraída y representada en forma de datos, puede manejarse mediante la computadora, porque el esquema del proceso de datos es E-P-S (Entrada-Proceso-Salida), es decir, datos entran como materia prima, se procesan para transformarlos en la información que se da como salida. Por ejemplo, en una situación de pago de sueldos (nómina), un trabajador puede representarse mediante los datos: Nombre del empleado, número de horas trabajadas y cuota por hora. El sueldo se obtiene multiplicando número de horas trabajadas por la cuota por hora. Y se da como salida el nombre y el sueldo. Tanto los datos como el procedimiento necesario para generar la información, se suministran a la computadora en forma de un programa constituido por instrucciones. La computadora interpreta y ejecuta las instrucciones del programa de acuerdo con ciertas reglas de sintaxis que conforman el lenguaje de programación, mediante el cual podemos comunicarle lo que debe hacer.

Los elementos básicos que componen una computadora son la unidad central de proceso, la unidad de memoria, la unidad de entrada y la unidad de salida.

La *unidad central de proceso* es el “cerebro” que controla el funcionamiento de los componentes y ejecuta las operaciones aritméticas y lógicas. Las operaciones del procesador central son muy simples, pero ejecutadas a una velocidad muy alta –del orden de millones por segundo– permiten la ejecución de tareas simples o complejas.



**Fig. 1.1** Diagrama que describe la organización funcional de una computadora.

La memoria se utiliza para almacenar los datos, y a éstos se les aplican las operaciones del procesador. Existen dos tipos de memoria: la principal y la auxiliar. La memoria principal permite al procesador extraer y almacenar datos a una velocidad comparable con la propia. Cada operación propicia por lo menos un acceso a la memoria. Para que el procesador pueda avanzar de una operación a la siguiente sin retraso, el programa de instrucciones se almacena en esta memoria; en otras palabras, la memoria principal guarda tanto las instrucciones como los datos sobre los que actúa el procesador central. La memoria principal está limitada por su alto costo; debido a esto no es posible conservar en ella grandes cantidades de datos e instrucciones y, en consecuencia, sólo se usa para guardar lo que el procesador esté utilizando por el momento. Además, tiene la característica de que no permite almacenar datos permanentemente, pues si se apaga la computadora se pierde lo que haya en memoria. Por tales razones, las computadoras están equipadas con memorias auxiliares para almacenamiento masivo y permanente de datos, tales como discos magnéticos fijos, disquetes (discos flexibles) magnéticos removibles, discos compactos, cintas magnéticas, entre otros. Estos dispositivos tienen más capacidad que la memoria principal, pero son más lentos. Los datos pueden almacenarse en ellos de manera permanente, es decir, pueden guardarse para usos posteriores.



**Fig. 1.2** Computadora personal.

La *unidad de entrada* se utiliza para introducir datos del exterior en la memoria de la computadora a través de dispositivos periféricos de entrada como teclados de terminales, ratón (mouse), discos, módem, lector de código de barras, etc. Esta unidad realiza automáticamente la traducción de símbolos inteligibles para la gente, en símbolos que la máquina pueda manejar.



**Fig. 1.3** Pantalla o monitor, dispositivo periférico de salida estándar.

La *unidad de salida* permite transferir datos de la memoria al exterior, a través de dispositivos periféricos de salida como impresoras, pantallas de video, módem, etc. Esta unidad realiza automáticamente la traducción de símbolos que puede manejar la máquina, en símbolos inteligibles para la gente.



**Fig. 1.4** Teclado, dispositivo periférico de entrada estándar.



**Fig. 1.5** Impresora, dispositivo periférico de salida impresa.



**Fig. 1.6** Mouse, dispositivo periférico de entrada.



**Fig. 1.7** Unidad de DVD, dispositivo periférico de entrada/salida.

## El programa

Un programa es un conjunto de instrucciones que guían a la computadora para realizar alguna actividad o resolver algún problema; en el programa se ejecutan diferentes acciones de acuerdo con los datos que se estén procesando. El programa debe incluir instrucciones para las acciones que deban ejecutarse sobre cada uno de los tipos de datos admitidos, además instrucciones que identifiquen los datos erróneos y recuperarse ante la aparición de éstos.

Cuando se ejecuta un programa con un tipo de datos específico, es probable que no se ejecuten todas las instrucciones, sino sólo las que sean pertinentes a los datos en cuestión. Un programa se compone de estructuras de datos, operaciones primitivas elementales y estructuras de control, como se muestra a continuación:

- Programa = Estructuras de datos**
  - + Operaciones primitivas elementales
  - + Estructuras de control

**Estructuras de datos.** Son las formas de representación interna de la computadora. Los hechos reales, representados en forma de datos, pueden estar organizados de diferentes maneras llamadas estructuras de datos. Por ejemplo el nombre, las horas trabajadas y el sueldo por hora son los datos mediante los cuales se representa un empleado en una situación de pago de sueldos (nómina).

**Operaciones primitivas elementales.** Son las acciones básicas que la computadora “sabe” hacer, y que se ejecutan sobre los datos para darles entrada, transformarlos y darles salida convertidos en información. Por ejemplo, el sueldo de un empleado se calcula multiplicando las horas trabajadas por la cuota horaria.

**Estructuras de control.** Son las formas lógicas de funcionamiento de la computadora mediante las que se dirige el orden en que deben ejecutarse las instrucciones del programa. Las estructuras de control son: La secuenciación, que es la capacidad de ejecutar instrucciones secuenciales una tras otra. La selección es la capacidad de escoger o seleccionar si algo se ejecuta o no, optar por una de dos o más alternativas, y la repetición, que es la capacidad de realizar

en más de una ocasión (es decir, varias veces) una acción o conjunto de acciones; por ejemplo calcular el sueldo a un empleado, pero repitiendo el cálculo  $n$  veces para  $n$  empleados.

## El lenguaje de programación

Un lenguaje de programación es el medio a través del cual le comunicamos a la computadora la secuencia de instrucciones que debe ejecutar para llevar a cabo actividades, tareas o solución de problemas. Todo lenguaje permite el manejo de los tres elementos que componen un programa, a saber: estructuras de datos, operaciones primitivas elementales y estructuras de control.

Recordemos que mediante un programa podemos representar en forma de datos cualquier situación de nuestra realidad, a los datos se les da entrada a la computadora mediante dispositivos de entrada como teclado, lector óptico de caracteres, ratón, etc.; una vez que los datos están en la computadora, se procesan para convertirlos en información, la cual será emitida hacia el exterior de la computadora mediante dispositivos de salida como son la pantalla, impresora, etcétera.

### • Características de los lenguajes de programación

Todo lenguaje está compuesto por un alfabeto, un vocabulario y una gramática. A continuación se describen estos componentes.

**Alfabeto o conjunto de caracteres.** Es el conjunto de elementos estructurales del lenguaje:

- a) Caracteres alfabéticos (letras minúsculas y mayúsculas).
- b) Caracteres numéricos (dígitos del 0 al 9).
- c) Caracteres especiales (símbolos especiales tales como [.], [,], [:], [:], [\$], [#], [/] y muchos otros).

**Vocabulario o léxico.** Es el conjunto de palabras válidas o reservadas en el lenguaje. Por ejemplo, las palabras program, begin, end, if, then, else, integer, real, string, repeat, for, while, char, procedure, function, byte, boolean tienen un significado predeterminado en el lenguaje Turbo Pascal, es decir, son las palabras reservadas del lenguaje Turbo Pascal. Así, cada lenguaje tiene sus propias palabras reservadas.

**Gramática.** Es el conjunto de lineamientos que se deben seguir para construir frases, oraciones o instrucciones. Mediante la gramática o sintaxis logramos transmitirle a la computadora lo que deseamos. Por ejemplo, para leer datos debemos seguir cierto lineamiento, también para imprimir, etcétera.

## La programación

Generalmente se consideran sinónimos los conceptos *programación* y *codificación*, lo cual constituye un error. Debemos tener presente que la finalidad de un programa es realizar algún proceso sobre ciertos datos para obtener ciertos resultados. La preparación de un programa implica aspectos tales como: ¿para qué sirve el proceso que se desea representar?, ¿qué datos usará, qué resultados producirá y cómo se realizará el proceso sobre los datos para obtener los resul-

tados esperados? Una vez identificado lo anterior se procede a diseñar la manera de cómo la computadora deberá hacerlo, tomando en cuenta su estructura interna y su funcionamiento. Hasta ese momento se tiene representada la solución de una manera convencional (algoritmo), pero enseguida se procede a codificar el programa que solucionará el problema, utilizando un lenguaje de programación.

#### • Características de un buen programa

Un programa bien escrito debe tener ciertas características básicas que le permitan operar correctamente; las principales serían las siguientes:

**Operatividad.** Lo mínimo que debe hacer un programa es funcionar; es decir, producir los resultados esperados independientemente de cualquier otra característica.

**Legibilidad.** Un programa puede hacerse más legible dándole cierto formato al código, utilizando el sangrado (indentación) para reflejar las estructuras de control del programa, e insertando espacios o tabuladores. Es conveniente diseñar reglas propias para darle uniformidad a todos los programas.

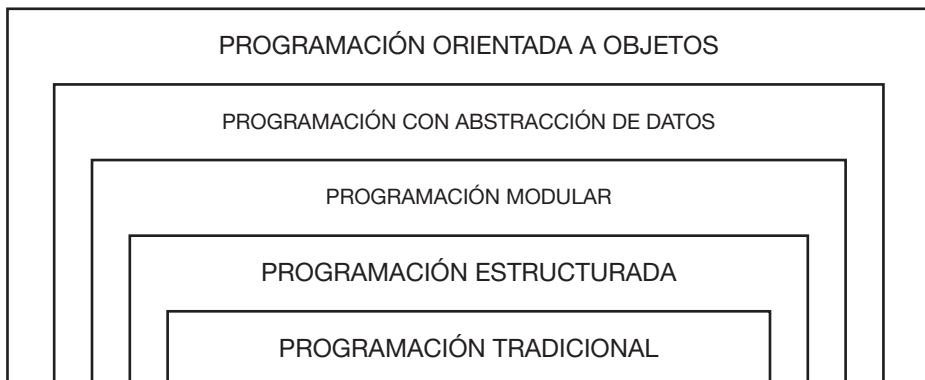
**Transportabilidad.** Un programa transportable es el que puede ejecutarse en otro entorno sin hacerle modificaciones importantes. Mientras menos modificaciones se hagan será más transportable, así que es conveniente no utilizar características especiales del hardware ni “facilidades” especiales del software.

**Claridad.** Esta característica se refiere a la facilidad con la que el texto del programa comunica las ideas subyacentes. El programa debe indicar claramente lo que el programador desea. Una buena programación es similar a la elaboración de un documento legal; por ejemplo, conviene utilizar nombres adecuados para los identificadores, hacer comentarios correctos, claros y concisos, etcétera.

**Modularidad.** Dividir el programa en un número de módulos pequeños y fáciles de comprender puede ser la contribución más importante a la calidad del mismo. Cada módulo debe realizar sólo una tarea específica, y no más. Los módulos tienen la virtud de minimizar la cantidad de código que el programador debe comprender a la vez, además de que permiten la reutilización de código.

## 1.2 Evolución de los paradigmas de programación

Desde que la programación de computadoras apareció como tal, la forma, el paradigma o modelo que se usa ha evolucionado constantemente. Sin embargo, las bases de la programación no han cambiado, simplemente se han ido añadiendo nuevos conceptos y nuevas estructuras. Todo inicia con las primeras estructuras que se inventaron cuando aparece la programación tradicional, luego, sobre esas bases se gestó la programación estructurada, para dar lugar a la programación modular, enseguida se agrega la programación con abstracción de datos, para llegar al desarrollo de la programación orientada a objetos. En la siguiente figura se esquematiza la evolución de los paradigmas de programación.



### Características de los paradigmas de programación

La evolución de los paradigmas de programación ha tenido tres grandes pasos, el primer gran paso es cuando la programación aparece como tal, es lo que se está esquematizando como la programación tradicional. Luego se dio un segundo gran paso y surge la programación estructurada. Después se experimentó un pequeño paso que dio lugar a la programación modular. Enseguida vino otro pequeño paso que permitió el surgimiento de la programación con abstracción de datos. Luego vino el tercer gran paso que es la aparición de la programación orientada a objetos.

#### Tradicional

La programación tradicional tuvo sus inicios a principios de la década de 1950. Los lenguajes de programación que se utilizaban eran los predecesores de FORTRAN, COBOL y BASIC. Las estructuras lógicas de control que se utilizaban eran: la secuenciación, IF-THEN, IF-THEN-ELSE y DO (en la actualidad conocido como FOR). La técnica de diseño de programas utilizada eran los diagramas de flujo.

La arquitectura de un programa consistía de un solo módulo, como se muestra a continuación:



Este módulo estaba formado por una secuencia ordenada de instrucciones:

- Instrucción 1
- Instrucción 2
- Instrucción 3
- Instrucción 4
- Instrucción 5
- Instrucción 6

Instrucción 7

Instrucción 8

- - -

- - -

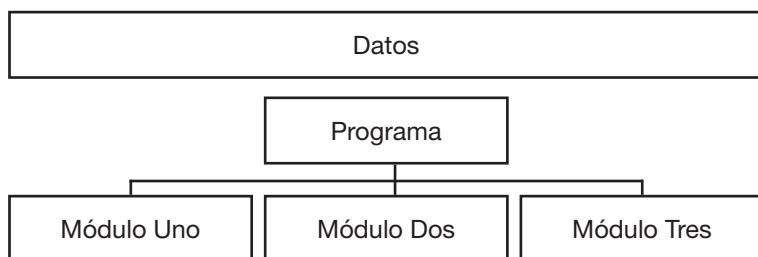
- - -

Instrucción N

### Estructurada

La programación estructurada tuvo sus inicios a mediados de la década de 1960. Los lenguajes de programación que se utilizaban eran PASCAL, COBOL estructurado, BASIC estructurado, FORTRAN con estilo estructurado, FORTRAN 90 y Lenguaje C. Las estructuras de control utilizadas eran la secuenciación, IF-THEN, IF-THEN-ELSE, CASE, FOR, DO-UNTIL y DOWHILE. Otras características son: Dividir un programa en módulos y funciones y estilo de programación. Las técnicas de diseño de programas que se utilizaban eran diagramas Warnier, diagramas estructurados, diagramas Chapin, seudocódigo y Top Down Design, entre otras.

La arquitectura de un programa consistía en datos y en un conjunto de módulos jerarquizados, como se muestra a continuación:



Cada módulo estaba formado por un conjunto ordenado de instrucciones:

#### Módulo Uno

Instrucción 1

Instrucción 2

Instrucción 3

- - -

- - -

- - -

Instrucción N

#### Módulo Dos

Instrucción 1

Instrucción 2

Instrucción 3

- - -

- - -

- - -

Instrucción N

#### Módulo Tres

Instrucción 1

Instrucción 2

Instrucción 3

- - -

- - -

- - -

Instrucción N

Al diseñar la solución en módulos, la programación estructurada permite solucionar problemas más grandes y complejos, de una mejor forma que como se hacía anteriormente.

### Modular

La programación modular tuvo sus inicios a fines de la década de 1970 y principios de la de 1980. El lenguaje de programación que se utilizó fue MODULA 2. Emerge el concepto de encapsulación, que en un módulo o paquete se encapsulan los datos y las funciones que los manipulan.

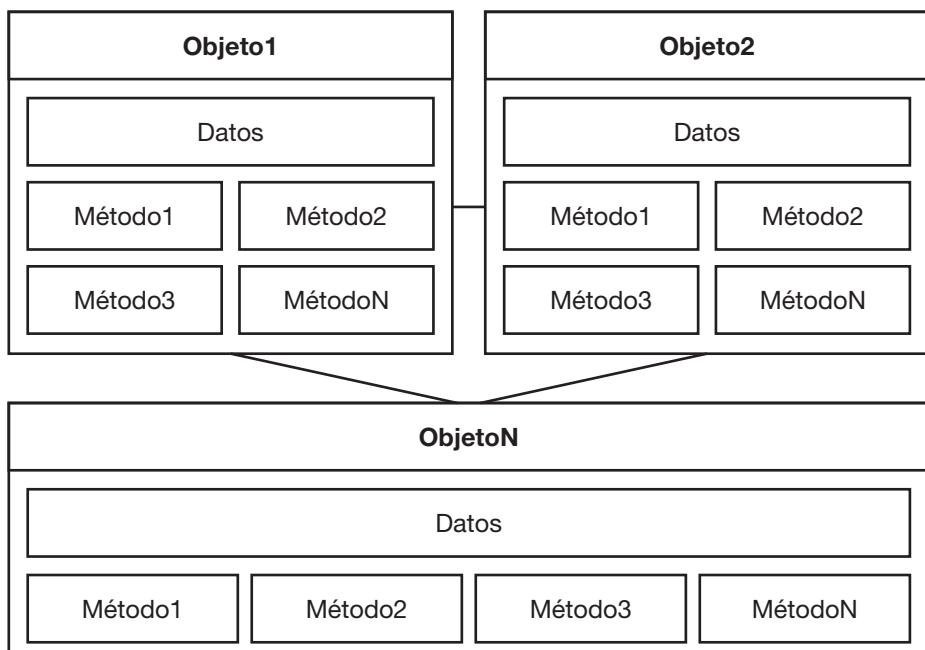
### Con abstracción de datos

La programación con abstracción de datos se generó en la década de 1980. El lenguaje de programación que se utilizó fue ADA. Con éste emerge el concepto de Tipos Abstractos de Datos (TAD).

### Orientada a objetos

La programación orientada a objetos, aunque el concepto se generó muchos años antes, es a finales de la década de 1980 y principios de la de 1990 que se pone en boga, la caracterizan los conceptos objetos, clases, encapsulación, herencia y polimorfismo. Los principales lenguajes de programación que se utilizan son C++, Java y C#. Las técnicas de diseño que se utilizan son Booch, Rumbaugh, Jacobson, Yourdon y UML (Unified Modeling Language), entre otras.

La arquitectura de un programa consiste en un conjunto de objetos, y cada objeto se compone por datos y un conjunto de métodos, donde cada método está formado por un conjunto de instrucciones, como se muestra a continuación:



La programación orientada a objetos permite manejar mejor la complejidad de los programas, porque permite una mayor pulverización o segmentación de los programas a través de los objetos, esto, de una forma más eficiente que como se hacía anteriormente con la programación estructurada.

### Resumiendo

Sobre las estructuras básicas que se crearon con la programación tradicional, se gestó una revolución de nuevas estructuras que se añadieron a la programación, a lo que se le llamó programación estructurada; ésta aportó las bases lógicas sobre la que se sustentó la programación modular; ésta añadió otro concepto más para dar lugar a la programación con abstracción de datos, y sobre todas esas bases, se sustenta la programación orientada a objetos.

## 1.3 El proceso de programación

Elaborar un programa de computadora implica llevar a cabo una serie de pasos secuenciales y cronológicos que comienzan con la detección y definición del problema y conducen a la implantación del programa que lo soluciona. A continuación se describen los pasos a seguir.

### • Definición del problema

Este proceso inicia cuando surge la necesidad de resolver algún problema mediante la computadora. Para empezar, se debe de identificar el problema y comprender la utilidad de la solución que se alcance. Es necesario tener una visión general del problema estableciendo las condiciones iniciales (los puntos de partida) y, además, los límites del problema, es decir, dónde empieza y dónde termina. Por ejemplo, si tenemos que calcular el sueldo de un empleado, la solución que se logre permitirá obtener la cantidad que debe pagársele y servirá precisamente para pagarle. La situación anterior consiste en un pago de sueldos, y parte de que cada empleado tiene algunos atributos como su nombre, el tiempo trabajando y el sueldo que percibe por unidad de tiempo dedicada a su labor.

### • Análisis del problema

A continuación es necesario entender con detalle el problema en cuestión, para obtener una radiografía del mismo en términos de los DATOS disponibles como materia prima, y definir el PROCESO necesario para convertir los datos en la INFORMACIÓN requerida.

La primera etapa consiste en definir los resultados esperados, es decir, la INFORMACIÓN que deberá producirse como salida. Respecto al problema de pago de salarios tenemos que se requiere la siguiente salida:

Nombre del empleado: XXXXXXXXXXXXXXXXXXXXXXXX

Sueldo: 99,999,999.99

La segunda etapa consiste en identificar los DATOS que se tienen como materia prima y que constituirán la entrada del programa. En este ejemplo tenemos:

- El nombre del empleado.
- El número de horas trabajadas.
- La cuota por hora.

La tercera etapa tiene como finalidad determinar el PROCESO necesario para convertir los datos de entrada en la información que se tendrá como salida. Volviendo al ejemplo, puesto que se requieren dos datos de salida, determinemos el proceso de la siguiente manera:

- a) ¿Cómo se calcula el nombre del empleado?  
No implica ningún cálculo, pues es un dato que se obtiene como entrada y que no se modifica.
- b) ¿Cómo se calcula el sueldo?  
El sueldo es un dato que no existe como entrada, pero que se obtiene o genera: multiplicando las horas trabajadas por la cuota horaria.

En este momento ya se tiene una comprensión clara del problema, y podemos avanzar hacia el siguiente paso.

#### • Diseño del programa

Durante este paso se procede a diseñar la lógica para la solución al problema, haciendo dos cosas:

**Elaborar el algoritmo.** Se diseña el algoritmo de la solución al problema, es decir, se estructura la secuencia lógica y cronológica de los pasos que la computadora deberá seguir, utilizando alguna técnica convencional como el pseudocódigo, los diagramas de flujo (en desuso con la programación estructurada), los diagramas Warnier, los diagramas Chapin, etc. Equiparando esta actividad con la construcción de una casa, equivale a diseñar el plano arquitectónico de la misma.

**Prueba de escritorio.** Se simula el funcionamiento del algoritmo con datos propios respecto al problema, y se comprueban a mano los resultados con el fin de validar la correcta operación del algoritmo. Si quedamos satisfechos con los resultados de la prueba habremos agotado este punto, pero en caso contrario se deberá modificar el algoritmo y posteriormente volverlo a probar hasta que esté correcto. Es posible que se deba retroceder a cualquier paso precedente.

En este momento se tiene ya diseñada la solución al problema, y estamos listos para pasar al siguiente punto.

#### • Codificación del programa

En este paso se procede a codificar el programa en el lenguaje de programación que vayamos a utilizar. Este proceso es sumamente sencillo, dado que ya tenemos diseñado el programa, sólo nos concretamos a convertir las acciones del

algoritmo en instrucciones de computadora. El programa codificado debe editarse, compilarse, probarse y depurarse, es decir, se ejecuta para verificar su buen funcionamiento y se hacen las correcciones o los ajustes pertinentes hasta que quede correcto. Si aparecen errores difíciles de corregir en este paso, quiere decir que debemos retroceder al paso 3 o al paso 2.

Para que un programa pueda ser entendido y ejecutado por la computadora, debe estar en lenguaje máquina o código objeto, el programa que nosotros hacemos en papel a lápiz o pluma, debe ser traducido por un **compilador**, a código asequible para la máquina mediante la **compilación**. El proceso de compilación es el siguiente: Una vez que tenemos codificado el programa en papel, debe ser introducido mediante el proceso de **edición**, para lo cual, se utiliza un **editor** que nos permite crear un archivo en el cual introducimos el programa, creándose el **programa fuente** con las instrucciones que nosotros elaboramos en el lenguaje que estemos utilizando en este momento. El programa fuente es sometido al proceso de **compilación**, que mediante un compilador (traductor del lenguaje), se traduce instrucción por instrucción a código objeto, creándose un archivo con el **programa objeto**, el cual es entendible directamente por la máquina. Si en el proceso de traducción se encuentra algún error, se suspende el proceso; el programador debe corregir el error en el programa fuente y luego someterlo de nuevo al proceso de compilación.

Una vez que el proceso de compilación ha terminado con éxito, se tiene el programa objeto, el cual puede ser ejecutado por la computadora, misma que seguirá las instrucciones paso a paso llevando a cabo las acciones que se le indican emitiendo los resultados correspondientes, si éstos no son satisfactorios o existen errores, el proceso de programación debe ser repetido desde alguno de las pasos precedentes.

#### • **Implantación del programa**

Una vez que el programa está correcto, se instala y se pone a funcionar, entrando en operación normalmente dentro de la situación específica para la que se desarrolló. Debe ser supervisado continuamente para detectar posibles cambios o ajustes que sea necesario realizar.

#### • **Mantenimiento del programa**

Un programa que está en operación, por un lado podría presentar errores, los cuales deben corregirse; por otro lado podría requerir cambios o ajustes en sus datos, proceso o información; esto implica que eventualmente necesitará mantenimiento para adecuarlo a los cambios que le impongan la dinámica cambiante de las empresas o de los problemas.

Lo anterior nos sitúa en una dinámica infinita, ya que si surge la necesidad de darle mantenimiento tendremos que regresar a algún paso precedente; al 4, al 3, al 2 o al punto uno para definir de nuevo el problema.

## 1.4 El algoritmo

---

En el proceso de programación hay un paso que es crucial al momento de desarrollar un programa, que es el diseño del programa, en otras palabras, diseñar o elaborar el algoritmo de la solución.

El *algoritmo* es una secuencia ordenada y cronológica de pasos que llevan a la solución de un problema o a la ejecución de una tarea (o actividad). Los pasos del algoritmo deben tener las siguientes características:

- Ser simples, claros, precisos, exactos.
- Tener un orden lógico.
- Tener un principio y un fin.

Aplicando el concepto de algoritmo a situaciones de nuestra vida cotidiana, tenemos que, ejemplos de algoritmos son las señas para encontrar una dirección, las recetas de cocina, los planos de construcción, las instrucciones para armar o utilizar un juguete, etc. Cuando diseñemos algoritmos deberemos considerar que los pasos cumplan con las características antes mencionadas.

A continuación aplicaremos estos conceptos en una situación de nuestra vida cotidiana, para efectos de resaltar las características de los pasos de todo algoritmo; posteriormente (en los capítulos subsecuentes) estaremos aplicándolos al desarrollo de algoritmos que deberá ejecutar la computadora.

### Ejercicios

A) Elaborar un algoritmo para que guíe a una persona normal –como usted lector– a *cambiar un foco fundido*, considerando que algún foco de nuestra casa (sala, comedor, baño, recámara, etc.) está fundido, piense en los pasos que deberá seguir.

Algoritmo *Cambiar foco fundido*

1. Quitar el foco fundido
2. Colocar el foco nuevo
3. Fin

Si usted pensó en los pasos a seguir, es probable que no coincidan exactamente con éstos, pero si conducen de manera efectiva a ejecutar el cambio del foco, entonces estará correcto, como es el caso de este algoritmo. Si bien es cierto que son pocos pasos, si usted los entiende y los ejecuta para lograr cambiar el foco, entonces está correcto.

Ahora bien, supóngase que estamos tratando de entrenar a un robot para que haga la tarea; en tal caso no funcionará el algoritmo, tendremos que ser más específicos y claros tomando en cuenta las capacidades elementales del robot. Preguntémonos: ¿qué sabe hacer el robot?, y con base en ello elaboraremos el algoritmo.

Capacidades del robot:

- Colocar la escalera  
*Suponemos que se tiene una escalera especial para que la maniobre el robot. Sabe traer la escalera desde su lugar y colocarla debajo del foco fundido.*
- Subir a la escalera  
*Sabe subir por los peldaños de la escalera hasta alcanzar el foco fundido o el lugar del foco.*
- Quitar el foco fundido  
*Sabe girar el foco a la izquierda hasta que salga.*
- Obtener foco de repuesto  
Suponemos que se tiene una caja con focos de repuesto de todas las medidas, y que el robot sabe dónde se encuentra la caja y puede obtener un foco nuevo y en buenas condiciones, igual al que traiga en su brazo.
- Colocar el foco de repuesto  
Sabe colocar y girar el foco de repuesto a la derecha hasta que esté apretado.
- Bajar de la escalera  
Sabe bajarse de la escalera hasta estar en el piso.
- Guardar la escalera  
Sabe guardar la escalera en el lugar correspondiente.
- Fin  
Sabe que ha terminado de ejecutar la tarea.

A continuación tenemos el algoritmo:

Algoritmo *Cambiar foco fundido*.

1. Colocar la escalera
2. Subir a la escalera
3. Quitar el foco fundido
4. Bajarse de la escalera
5. Obtener foco de repuesto
6. Subirse a la escalera
7. Colocar el foco de repuesto
8. Bajar de la escalera
9. Guardar la escalera
10. Fin

B) Ahora pensemos en otro robot que tiene las siguientes capacidades:

- Colocar la escalera  
*Sabe traer la escalera desde su lugar y colocarla debajo del foco fundido.*
- Subir un peldaño  
*Sabe subir un peldaño de la escalera y sabe detectar cuando alcanza el foco o el lugar del foco.*

- Dar vuelta a la izquierda  
*Sabe girar el foco una vuelta a la izquierda y detectar cuando sale.*
- Obtener foco de repuesto  
Suponemos que se tiene una caja con focos de repuesto de todas las medidas, y que el robot sabe dónde se encuentra la caja y puede obtener un foco nuevo y en buenas condiciones, igual al que traiga en su brazo.
- Dar vuelta a la derecha  
*Sabe girar el foco una vuelta a la derecha y detectar cuando está apretado.*
- Bajar un peldaño  
*Sabe bajar un peldaño de la escalera y sabe detectar cuando está en el piso.*
- Guardar la escalera  
*Sabe guardar la escalera en el lugar correspondiente.*
- Fin  
*Sabe que ha terminado de ejecutar la tarea.*



Cuando se diseña un algoritmo se anotan *paso a paso*, en secuencia, las acciones que se ejecutarán, en ocasiones hay que repetir uno o varios pasos cierto número de veces (14 por ejemplo); en tal caso tenemos que controlar el primer paso, el segundo, el tercero, y así sucesivamente hasta el décimo cuarto para que pueda terminar el proceso. Esto se conoce como *ciclo repetitivo*.

En otras ocasiones tenemos que llegar a un resultado partiendo de dos o más situaciones. En este caso debemos tomar en cuenta –por un lado– cómo se llega desde una parte, y por el otro cómo se llegaría desde la otra parte, la alternativa; en estas circunstancias se utiliza la *selección*.

Es obvio que el algoritmo que hicimos anteriormente no serviría para este nuevo robot. Es necesario hacer otro algoritmo considerando las capacidades de este robot.

A continuación tenemos el algoritmo:

**Algoritmo Cambiar foco fundido**

1. Colocar la escalera
2. Repetir
  - Subir un peldaño
  - Hasta alcanzar el foco
3. Repetir
  - Dar vuelta a la izquierda
  - Hasta que el foco salga
4. Repetir
  - Bajar un peldaño
  - Hasta estar en el piso
5. Obtener foco de repuesto
6. Repetir
  - Subir un peldaño
  - Hasta alcanzar el lugar del foco
7. Repetir
  - Dar vuelta a la derecha
  - Hasta que el foco esté apretado
8. Repetir
  - Bajar un peldaño
  - Hasta estar en el piso
9. Guardar la escalera
10. Fin

Todas las actividades que llevamos a cabo los seres humanos son algoritmos que hemos aprendido a seguir. Caminar y lavarse los dientes, por ejemplo, son secuencias lógicas de pasos. La civilización está basada en el orden de las cosas y de acciones; éstas se organizan conforme a secuencias lógicas, y a esto se le llama *programación*.

Durante el desarrollo de algoritmos para computadora, es necesario idear los pasos que la máquina deberá seguir para realizar alguna tarea, actividad o resolver algún problema de nuestra vida cotidiana. Cuando se presenta el problema de realizar cierta tarea, debemos efectuar diversas actividades subyacentes tales como hacer preguntas, buscar objetos conocidos o dividir el problema en partes. A menudo hacemos todo eso inconscientemente, pero al elaborar algoritmos para solucionar problemas con la computadora debemos detallar esas actividades de manera explícita.

Si se nos solicita una tarea verbalmente, por lo general hacemos preguntas hasta que quede claro lo que debemos hacer. Por ejemplo, preguntamos cuándo, por qué, dónde y otros aspectos, hasta que la tarea queda completamente especificada. Si las instrucciones están por escrito podemos anotar preguntas en los márgenes, subrayar palabras, resaltar frases u oraciones, o indicar de alguna otra forma que la idea no está clara y necesita precisión.

Si la tarea nos la imponemos nosotros mismos, puede ser que la pregunta se plantee a nivel subconsciente. Algunas preguntas comunes en el contexto de la programación son:

- ¿Qué datos tenemos para trabajar? y ¿Cuál es su apariencia?
- ¿Cuántos datos hay?
- ¿Cómo sabremos que ya están procesados todos los datos?
- ¿Cuál debe ser el formato de la salida?
- ¿Cuántas veces debe repetirse el proceso?
- ¿Qué condiciones especiales de error pueden presentarse?

## 1.5 Ejercicios propuestos

1. Elaborar un algoritmo para hacer palomitas de maíz en una cacerola puesta al fuego, usando sal y maíz.
2. Elaborar un algoritmo que permita cambiar un vidrio roto de una ventana.
3. Elaborar un algoritmo para cambiar un neumático pinchado.
4. Elaborar un algoritmo para hacer una llamada telefónica.
5. Definir “su” robot, es decir, lo que sabe hacer tomando como referencia lo que usted hace, y elaborar un algoritmo que lo lleve desde que se despierta por la mañana hasta que llega a la escuela, trabajo o algún otro lugar.

## 1.6 Resumen de conceptos que debe dominar

- Qué es una computadora y sus componentes principales.
- Cuáles son los paradigmas de programación.
- Cuáles son las características de los lenguajes de programación.
- Qué es un programa y cuáles son las características de un buen programa.
- Cuáles son los pasos del proceso de programación.
- Qué es el algoritmo y cuáles son sus características.

## 1.7 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

### 1.7.1 Resumen gráfico del capítulo

### 1.7.2 Autoevaluación

### 1.7.3 Power Point para el profesor (\*)

# 2

## Elementos para solucionar problemas en seudocódigo

### Contenido

- 2.1 Estructuras de datos
- 2.2 Operaciones primitivas elementales
- 2.3 Estructuras de control
- 2.4 Resumen de conceptos que debe dominar
- 2.5 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 2.5.1 Resumen gráfico del capítulo
  - 2.5.2 Autoevaluación
  - 2.5.3 Power Point para el profesor (\*)

### Objetivos del capítulo

- Conocer los elementos para diseñar algoritmos en seudocódigo: las estructuras de datos, datos numéricos enteros y reales, datos carácter, cadena de caracteres, etc.
- Conocer las operaciones primitivas elementales: declaraciones de variables, constantes y tipos, lectura de datos (entrada), escritura de datos (salida).
- Conocer las operaciones aritméticas fundamentales (sumar, restar, multiplicar y dividir).
- Conocer las estructuras de control: la secuenciación, la selección (if-then, if-then-else, switch), y la repetición (do...while, for, while).

## Introducción

Con el estudio del capítulo 1, usted ya conoce los principales conceptos acerca de las computadoras: la computadora y sus componentes, el programa, el lenguaje de programación y la programación; las características de un buen programa; la evolución de los paradigmas de programación; los pasos que integran el proceso de programación; lo que es un algoritmo y sus características.

En este capítulo, el lector aprenderá los elementos para diseñar algoritmos en seudocódigo. Se define que seudocódigo es una técnica para diseñar programas o elaborar algoritmos. Se explica que las estructuras de datos son las formas de cómo la computadora representa los datos internamente, qué son los tipos de datos simples: datos numéricos enteros y reales, datos carácter, datos cadena de caracteres y booleanos.

Se enseña qué son, cómo se representan y cómo se usan en seudocódigo las operaciones primitivas elementales, qué son: declaraciones de variables, constantes y tipos, lectura de datos (entrada), escritura de datos (salida).

También se especifica qué son, cómo se representan y cómo se usan las operaciones aritméticas fundamentales (sumar, restar, multiplicar y dividir), mediante las cuales se hacen cálculos.

Asimismo, se expone que las estructuras de control son las formas lógicas con las que funciona la computadora y mediante las cuales se dirige el funcionamiento de la misma, y se le da orden lógico a las operaciones primitivas elementales que actúan sobre los datos.

Se explica que las estructuras de control son: la secuenciación, la selección, que a su vez tiene tres formas: simple (if-then), doble (if-then-else) y múltiple (switch); la repetición, que también tiene tres formas: do...while, for y while.

En el siguiente capítulo se estudia la estructura de control: la secuenciación, qué es, cómo se representa y cómo se usa en seudocódigo.

Ya sabemos lo que es un algoritmo y conocemos la manera de diseñar soluciones para problemas de la vida cotidiana. Ahora aplicaremos estos conceptos a la elaboración de algoritmos para programar computadoras, utilizando el seudocódigo.

El seudocódigo es una técnica para diseño de programas (algoritmos) que permite definir las estructuras de datos, las operaciones que se aplicarán a los datos y la lógica que tendrá el programa de computadora para solucionar un determinado problema. Utiliza un seudolenguaje muy parecido a nuestro idioma, pero que respecta las directrices y los elementos de los lenguajes de programación.

En este capítulo analizaremos las capacidades básicas o elementales de una computadora, ya que es indispensable conocerlas para poder desarrollar algoritmos.

Los elementos básicos para solucionar problemas son las estructuras de datos, las operaciones primitivas elementales y las estructuras de control. A continuación explicaremos cada una de ellas y su forma de representación en seudocódigo.

## 2.1 Estructuras de datos

Las estructuras de datos son las formas de representación interna de datos de la computadora, mediante las que se representa cualquier situación en la computadora, es decir, son los tipos de datos que maneja la máquina. Por ejemplo, podemos representar a un trabajador mediante los datos nombre del empleado, número de horas trabajadas, cuota por hora, etcétera.

### 2.1.1 Tipos de datos

Los tipos de datos más comunes son los *numéricos* (entero y real), cadena de caracteres (alfabéticos o alfanuméricos), carácter y boolean, cada uno de los cuales puede manejarse como una *constante* o como una *variable*. A continuación se describe cada tipo:

#### Datos numéricos

Estos datos se dividen a su vez en **ENTEROS** y **REALES**. Los ENTEROS son los números que no contienen componentes fraccionarios y, por tanto, no incluyen el punto decimal; pueden ser positivos o negativos, como por ejemplo 450, -325, 4 o -4. Se representa como *Entero*. Los REALES son los números que contienen una parte fraccionaria y, por tanto, incluyen el punto decimal; pueden ser positivos o negativos, como por ejemplo 465.0, 42.325, 800.02, -24.5, o -42.3. Se representa como *Real*.

#### Datos cadena de caracteres

Los datos cadena de caracteres están compuestos por una hilera (cadena) de caracteres alfabéticos, numéricos y especiales, que sirven para representar y manejar datos tales como nombres de personas o de empresas, descripciones de artículos o productos de inventarios, direcciones de personas o empresas, entre otros. Se representa como *Cadena*.

Este tipo de datos también se conoce como alfabético o alfanumérico, ya que su contenido –sea cual fuere– siempre se considera como una serie de caracteres; los valores (constantes) de este tipo se encierran entre comillas (""). Por ejemplo:

“Universidad de Sonora”  
“Socorro Román Maldonado”  
“Dr. de la Torre y Vicente Guerrero # 75”  
“Guamúchil, Sinaloa, México”  
“Tornillo X25 ¾”  
“Bicicleta Cross Z-47”  
“Cualquier otra cosa que sea una constante literal”



Estos conceptos de *Entero* y *Real* son acuñados y válidos en el ámbito computacional, mas no para el contexto de la teoría general de los números. Los lenguajes de programación como C, C++ y otros, proporcionan más de un tipo de dato de tipo entero y real. Por ejemplo, Java tiene los tipos de datos: byte, short, int y long, todos son de tipo entero; asimismo, proporciona los tipos: float y double, ambos son de tipo real. Sin embargo, aquí en seudocódigo sólo nos interesa manejar Entero y Real; en el paso que nos corresponda traducir los algoritmos a instrucciones en un lenguaje, ahí sí usaremos la variante más apropiada al problema que estamos resolviendo.

### Datos carácter

El tipo de dato carácter, utiliza 1 byte, puede almacenar un carácter; cualquier carácter válido para la computadora según el código ASCII. Se representa como Carácter; los valores (constants) de este tipo se encierran entre apóstrofos (''). Por ejemplo: 's', 'n', 'S', 'N' y se representan como Carácter.



Los nombres de las variables y otros identificadores deben definirse de tal manera que indiquen lo que están representando, ya sea datos, funciones, métodos, clases, objetos, es decir, que sea un mnemónico de lo que representa. Por ejemplo, si se va a manejar el dato nombre del empleado, se podría usar Nombre o NombreEmpleado. El dato horas trabajadas se podrá nombrar HorasTrabajadas. El dato cuota por hora se podría manejar como CuotaHora. Usted, como programador, puede adoptar su propio estilo para dar nombres a variables, sin embargo, considere lo siguiente:

Hay autores de lenguajes de programación que usan sólo letras mayúsculas, ejemplos: NOMBRE, HORASTRABAJDAS, CUOTAHORA, X, Y, Z.

Otros autores usan sólo letras minúsculas, ejemplos: nombre, horastrabajadas, cuotahora, x, y, z.

Otros utilizan tanto mayúsculas como minúsculas, ejemplo: CuotaHora.

### Datos boolean

Este tipo de dato está compuesto por los valores False (falso) y True (verdadero); este tipo no puede ser leído o escrito, sólo asignado, es útil para plantear cierto tipo de condiciones en el manejo de las estructuras lógicas de control. Se representa como Boolean.

#### 2.1.2 Variables

Las variables sirven para representar y manejar datos. Todo dato que vaya a ser introducido a la computadora, y todo dato que vaya a ser generado o calculado a partir de otros datos para obtener algún resultado, tiene que identificarse y manejarse en forma de variable; una variable tiene las siguientes características:

a) **Nombre.** Es el identificador de la variable y que servirá para referenciarla.

Reglas para asignar nombres de variables y otros identificadores como métodos, funciones, clases, objetos, etcétera:

1. Se pueden utilizar combinaciones de letras mayúsculas y minúsculas (A...Z, a...z); dígitos (0...9) y el símbolo de subrayado [o guión bajo (\_)] aunque por cuestiones de estilo no lo usaremos en este libro].
2. El nombre debe iniciar con una letra.
3. Es conveniente que la longitud no pase de 20 caracteres.
4. No debe ser palabra reservada (como if, then, else, for, do, while, etcétera).
5. Estilo de programación: En este libro se utilizará el estilo que maneja el lenguaje Java y los demás lenguajes modernos, dicho estilo es el siguiente:

a) En nombres de clases:

- La primera letra mayúscula y las demás minúsculas, ejemplos: Empleado, Alumno, Cliente.
- Si se juntan varias palabras, se aplica el punto anterior a cada una de las palabras, ejemplos: EmpleadoConfianza, EmpleadoSindicalizado, EmpleadoPorHoras.

b) En nombres de variables, funciones, objetos, métodos y otros:

- Si el elemento se compone por una sola palabra, se usan minúsculas, ejemplos: nombre, sueldo, x, y, z, x1, y2, z1, x25, y11, z12.
- Si se juntan varias palabras, de la segunda palabra en adelante, la inicial es mayúscula y las demás minúsculas, ejemplos: nombreEmpleado, cuotaPorHora, cuotaHora, sueldoEmpleado, objetoEmpleado, objetoAlumno, calcularSueldo.



En general los lenguajes de programación no admiten letras acentuadas, por lo tanto, nombres de variables como: opción, producción, tamaño, año, número, no son correctos. Los nombres correctos serían opcion, produccion, tamano, anio, numero. Observe que en lugar de la ñ se coloca ni para evitar nombres de variables altisonantes.

- b) **Contenido** Se puede considerar que toda variable tiene una “casilla” donde se almacena el valor que toma en cada ocasión. Esto es similar a tener un casillero donde las casillas sirven para guardar objetos y tienen su respectiva identificación que permite referenciarlas. Por ejemplo:

nombreEmp	
cuotaHora	
horasTrab	

- c) **Tipo de datos.** Toda variable debe estar asociada con un tipo de datos Entero, Real, Cadena, etc., y de acuerdo con esto sólo podrá tomar valores válidos para el tipo de dato definido.

### 2.1.3 Constantes

Las constantes son valores específicos, en consecuencia invariables. Por ejemplo, constantes de tipo entero son: 5, 10, -56 y 20; constantes de tipo real son 3.1416, 40.5, -1.5 y 2.718; constantes de tipo cadena de caracteres son “Universidad de Sonora”, “Tornillo X25¾”, “Rosales # 245 Sur”; constantes de tipo carácter son ‘s’, ‘n’, ‘S’, ‘N’; y constantes de tipo boolean son true y false.



Cuando se va a dar nombre a un dato como número de horas trabajadas, el nombre más entendible es numeroDeHorasTrabajadas, sin embargo, es demasiado grande, entonces lo que debemos hacer es “abreviar” algunas de las palabras. Podría quedar: numHrsTrab, éste es un buen nombre, porque se entiende lo que representa y no es muy largo. Con nombres como éste debe tenerse cuidado, porque un error muy común de quienes recién se inicien en este tema, es ponerle puntos (como cuando se abrevia aplicando las reglas gramaticales del español). Para esta situación, podrían definir num. Hrs.Trab., lo que es erróneo, porque en nombres de identificadores no se admite el punto. Otro problema común en los principiantes es poner espacios, por ejemplo num Hrs Trab lo que constituye también un error.

## 2.2 Operaciones primitivas elementales

Las operaciones primitivas elementales son las acciones básicas que la computadora puede ejecutar. A continuación estudiaremos las características de cada una.

### 1. Declarar

Es una acción que se considera como no ejecutable, ya que mediante esta se define el entorno en el que se ejecutará el algoritmo (programa), en esta parte se declaran las variables, constantes, tipos, objetos, etc. A continuación se explican.

- a) **Constantes.** Como ya se explicó, las constantes son valores específicos, sin embargo, se pueden definir constantes simbólicas, es decir, mediante un identificador, con el siguiente formato:

```
Constantes
NomConstante = Valor
```

*En donde:*

Constantes	Identifica la acción que se va a realizar, la cual es definición de constantes simbólicas.
NomConstante	Es el identificador de la constante.
Valor	Es el valor que toma la constante.



Es común que los identificadores de constantes se definan en mayúsculas, el ejemplo anterior quedaría:

Declarar  
Constantes  
PI = 3.14159265  
CIEN = 100  
COMENTARIO1 = "Aprobado"  
COMENTARIO2 = "Reprobado"

## Ejemplos:

```
Declarar  
    Constantes  
        Pi = 3.14159265  
        Cien = 100  
        Comentario1 = "Aprobado"  
        Comentario2 = "Reprobado"
```

**b) Tipos.** Se definen nuevos tipos de datos. Formato:

Tipos  
NombreTipo = (Valor1, Valor2, , ValorN)

*En donde:*

**NombreTipo** Es el nombre del nuevo tipo de dato.

Valor1, Valor2, ..., ValorN  
Son los valores que integran el nuevo tipo de dato.

## Ejemplo:

```
Declarar
    Tipos
        Colores = (Negro, Blanco, Azul, Rojo, Amarillo)
    Variables
        colFondo: Colores
```

Se ha definido el tipo de dato Colores, y luego la variable colFondo del tipo Colores, es decir, que la variable colFondo podrá tomar los valores definidos en el tipo Colores. Este tipo será utilizado más ampliamente con arreglos, registros y archivos, en los capítulos correspondientes.

**c) Variables.** Esta acción permite representar en un programa las estructuras de datos necesarias para abstraer y representar en la computadora la situación real del problema o de la tarea que se manejará, en otras palabras, se definen las variables necesarias mediante el siguiente formato:

```
Variables
    nomVariable1: Tipo de dato
    nomVariable2: Tipo de dato
    .
    .
    nomVariableN: Tipo de dato
```

**En donde:**

Variables	Identifica la acción que se va a realizar, la cual es definición de variables.
nomVariable1, nomVariable2, nomVariableN	Son los nombres de las variables, de acuerdo con los lineamientos antes expuestos.
Tipo de dato	Indica el tipo de datos que tendrá la variable, a saber: Entero, Real, Cadena, etcétera.

**Ejemplo:**

Para representar la situación de pago de sueldos; que ya se ha mencionado, tendremos que hacer lo siguiente:

```
Declarar
Variables
    nombreEmp: Cadena
    horasTrab: Entero
    cuotaHora: Real
    sueldo: Real
```



Cuando no se indica el número de caracteres, queda abierto y le pueden caber hasta 255 caracteres. En la mayoría de los capítulos de este libro vamos a utilizar Cadena; sin embargo, en los capítulos 8 y 9 habrá algunos ejemplos en los que es mejor indicar la cantidad, para llevar un mejor control del espacio en memoria utilizado por las estructuras usadas en ese momento.

En el tipo de dato Cadena puede indicarse la longitud en número de caracteres que le caben a la variable, en el ejemplo podría quedar así:

nombreEmp: Cadena[30]

Es decir, a la variable *nombreEmp* le caben 30 caracteres.

**d) Objetos.** Aunque aquí también se pueden declarar objetos, en este libro lo haremos como se explica en el capítulo 11, ya que es a partir de ahí, que los usaremos.

## 2. Lectura de datos (Entrada)

Esta operación nos permite introducir los datos a la computadora, es decir, introducir la materia prima para el proceso. La introducción de datos puede hacerse desde un teclado, lector de código de barras, un archivo en un disco, un ratón (mouse) o desde cualquier otro dispositivo de entrada. El dispositivo estándar de entrada es el teclado. El formato para leer datos es:

```
Ler nomVariable1, nomVariable2, nomVariableN
```

**En donde:**

Ler	Indica la acción que se va a realizar, la cual es lectura de datos.
nomVariable1, nomVariable2, nomVariableN	Son los nombres de las variables donde se leerán los datos.

Ejemplo:

Para la situación de pago de sueldos que venimos siguiendo se deben leer el nombre del empleado, las horas trabajadas y la cuota por hora:

```
Leer nombreEmp, horasTrab, cuotaHora
```

Esta acción espera a que se tecleen los datos correspondientes, los cuales se almacenarán en las variables especificadas. En virtud de que los nombres de las variables son mnemónicos con respecto a los datos que representan, se facilita la comprensión de la acción. Con esta acción entendemos que se leerán el nombre, las horas trabajadas y la cuota por hora, y además que la lectura implica un proceso interactivo en el que se piden los datos, en la pantalla. Ejemplo:

```
Teclee Nombre del empleado:_
```

```
Teclee Número de horas trabajadas:_
```

```
Teclee Cuota por hora:_
```

Esta acción de lectura de datos puede hacerse de una forma más detallada, haciendo que vaya acompañada por una solicitud de los datos para enfatizar la interactividad entre la persona que introducirá los datos y la computadora; en tal caso la lectura queda así:

```
Solicitar Nombre del empleado, Número de horas  
trabajadas y Cuota por hora
```

```
Leer nombreEmp, horasTrab, cuotaHora
```

También se podría leer así:

```
Solicitar Nombre del empleado
```

```
Leer nombreEmp
```

```
Solicitar Número de horas trabajadas
```

```
Leer horasTrab
```

```
Solicitar Cuota por hora
```

```
Leer cuotaHora
```



Solicitar los datos es equivalente a imprimir en la pantalla haciendo la petición de que se introduzcan los datos; se podría colocar así:

Imprimir "Teclee Nombre del empleado"

Imprimir "Teclee Número de Horas trabajadas"

Imprimir "Teclee Cuota por hora"

### 3. Operaciones aritméticas fundamentales

Estas operaciones permiten modificar la apariencia de los datos y generar información; en otras palabras, procesar los datos que entran como materia prima para convertirlos en información que saldrá de la computadora.

Las operaciones aritméticas son:

+ Suma

- Resta

\* Multiplicación

/ División real

\ División entera

Mod      Residuo de una división entera  
 =        Asignación

Mediante tales operaciones se forman expresiones aritméticas para realizar los cálculos.

Formato:

Calcular variable = expresión

*En donde:*

variable      Es el nombre de la variable en la que se asignará el resultado de expresión.  
 expresión      Es un valor constante, una variable o una expresión algebraica (para calcular un valor), el cual se le asigna a la variable.  
 =              Es el símbolo que indica la asignación del valor de expresión a la variable.

Ejemplo:

Calcular sueldo = horasTrab \* cuotaHora

*Explicación:*

Se calcula horasTrab \* cuotaHora y el resultado se coloca en sueldo.



La palabra Calcular puede omitirse, quedando la expresión así:

sueldo = horasTrab \* cuotaHora

Otros ejemplos:

a = 1	a toma el valor 1
b = a + 1	b toma el resultado de 2
b = b - 1	b toma el resultado de b - 1
y = 5 / 2	y toma el valor 2.5. División real (hasta decimales)
z = 5 \ 2	z toma el valor 2. División entera (no decimales)
r = 5 Mod 2	r toma el valor 1, es decir, el residuo
observacion = "Aprobado"	observacion toma el valor "Aprobado"

Ejemplo:

Las expresiones aritméticas deben escribirse en una línea para que la computadora pueda evaluarlas. Por ejemplo, la expresión:

$$N = \frac{X + Y}{Y - 1}$$

nosotros la entendemos perfectamente y la podríamos evaluar así:

$$\begin{aligned} \text{si} \quad & X = 5 \\ & Y = 3 \end{aligned}$$

Luego haríamos la sustitución de valores y la resolveríamos:

$$N = \frac{5 + 3}{3 - 1} = \frac{8}{2} = 4$$

Sin embargo, la computadora no entiende este formato, por lo cual debemos escribir la expresión original en una línea para que la reconozca la computadora; después de hacerlo nos queda de la siguiente forma:

$$n = (x + y) / (y - 1)$$

La computadora examina toda la expresión y va evaluando cada componente de acuerdo con cierto orden de precedencia que tienen las operaciones aritméticas, que es el siguiente:

#### 1. Operaciones entre paréntesis.

Primero busca lo que está entre paréntesis y lo evalúa. En caso de haber paréntesis anidados, evalúa primero los más internos y luego prosigue con los externos.

#### 2. Operaciones unarias.

A continuación evalúa las operaciones unarias, es decir, de cambio de signo. Ejemplo:  $-4$ ; le cambia el signo a  $4$ .

#### 3. Multiplicación y división.

Posteriormente evalúa las operaciones de multiplicación y división, las cuales tienen el mismo nivel de precedencia.

#### 4. Suma y resta.

Estas operaciones se evalúan al final, ya que tienen el más bajo nivel de precedencia.

Otros ejemplos de conversión en una línea:

$$x = A+B - \frac{Y}{Z} \quad x = a+b - y/z \quad \text{o} \quad x = a+b - (y/z)$$

$$w = \frac{A \cdot C}{D} + \frac{B}{C} \quad w = (a * c/d) + b/c$$

$$z = \frac{T(Y-2) + R}{S} \quad z = (t * (y-2) + r) / s$$

$$x = \frac{CE - BF}{AE - BD} \quad x = ((c*e) - (b*f)) / ((a*e) - (b*d))$$



Cuando existen varias operaciones de un mismo nivel, se evalúan de izquierda a derecha.

$$F = \frac{9}{5} C + 32 \quad f = 9/5 * c + 32$$

#### 4. Escritura de datos (Salida)

Mediante la escritura damos salida a los datos de la computadora hacia un medio periférico, como por ejemplo la pantalla de video, la impresora, disco u otro. La salida estándar es el video.

Formato:

```
Imprimir nomVariable1, nomVariable2, nomVariableN
```

*En donde:*

Imprimir                      Identifica la acción de escritura.

nomVariable1, nomVariable2, nomVariableN

Son los nombres de las variables que contienen los datos que serán impresos.

Ejemplo:

En el ejemplo que venimos siguiendo se debe imprimir como resultado el nombre del empleado y su sueldo. Con la acción o instrucción:

```
Imprimir nombreEmp, sueldo
```

Se indica que se imprimen el nombre y el sueldo, que es la información requerida en este problema. Por el momento no se hace énfasis en el formato de salida, sino sólo se indica lo que sale –en términos de las variables– aprovechando que estamos utilizando nombres mnemónicos. Cuando se analiza el problema es donde se define el formato de la salida, por ejemplo si será un cheque, un recibo o algún otro documento.

*Explicación:*

Se imprime el letrero Nombre, luego se imprime el contenido de la variable nombreEmp, luego se imprime el letrero Sueldo y por último se imprime el contenido de la variable sueldo.



En caso de que se desee acompañar los datos con un letrero, se hace posiblemente entre comillas, ejemplo:

Imprimir

"Nombre = ", nombreEmp, "Sueldo = ", sueldo



Si se quiere indicar que la salida es hacia la impresora se puede colocar la palabra Impresora, ejemplo:

Imprimir Impresora, nombreEmp, sueldo

## 2.3 Estructuras de control

Son las formas lógicas como puede comportarse una computadora y mediante las cuales se dirige el funcionamiento de la misma. Se le da orden lógico a las operaciones primitivas elementales que actúan sobre los datos. Las estructuras de control son la secuenciación, la selección, que a su vez tiene tres formas: simple (if-then), doble (if-then-else) y múltiple (switch), y la repetición, que también tiene tres formas: do...while, for, while. En los capítulos subsecuentes se estudiarán detalladamente cada una de estas estructuras.

## 2.4 Resumen de conceptos que debe dominar

- Estructuras de datos.
  - Tipos de datos en seudocódigo: datos numéricos enteros y reales; datos carácter y cadena de caracteres. Los conceptos de constantes y variables y sus características.
- Operaciones primitivas elementales.
  - Declarar variables, constantes y tipos.
  - Lectura de datos (Entrada).
  - Escritura de datos (Salida).
  - Operaciones aritméticas fundamentales (suma, resta, multiplicación y división).
- Cuáles son las estructuras de control.
  - Secuenciación.
  - Selección (if-then, if-then-else, switch).
  - Repetición (do...while, for, while).

## 2.5 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

### 2.5.1 Resumen gráfico del capítulo

### 2.5.2 Autoevaluación

### 2.5.3 Power Point para el profesor (\*)

# 3

## La secuenciación

### Contenido

- 3.1 Nuestro primer problema
- 3.2 Estructura y diseño de un algoritmo
- 3.3 Nuestro primer algoritmo
- 3.4 Funciones matemáticas
- 3.5 Ejercicios resueltos
- 3.6 Ejercicios propuestos
- 3.7 Resumen de conceptos que debe dominar
- 3.8 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 3.8.1 Resumen gráfico del capítulo
  - 3.8.2 Autoevaluación
  - 3.8.3 Programas
  - 3.8.4 Power Point para el profesor (\*)

### Objetivos del capítulo

- Dominar y utilizar la estructura de control secuenciación.
- Aprender también a utilizar las funciones matemáticas (seno, coseno, arco tangente, logaritmo natural, etc.).
- Ejercitarse en los ejercicios resueltos y propuestos.

## Introducción

Después de haber estudiado el capítulo 2, usted ya domina los elementos para diseñar algoritmos en seudocódigo: los tipos de datos; cómo declarar variables y constantes; cómo solicitar y leer datos; cómo hacer cálculos y cómo imprimir datos; además: cuáles son las estructuras de control.

En este capítulo se inicia el estudio de las estructuras de control, primero se aborda la estructura de control más elemental que es la secuenciación, ésta es la capacidad lógica que tiene la computadora de ejecutar un conjunto de instrucciones secuencialmente, una a una, desde la primera hasta la última.

Se explica que los lenguajes de programación proporcionan funciones estándar, como son las funciones matemáticas: seno, coseno, arco tangente, logaritmo natural, etcétera; se especifica cómo representarlas y usarlas en seudocódigo.

Algo esencial en el aprendizaje de la programación de computadoras es la aplicación de los conceptos estudiados en la elaboración de algoritmos para solucionar problemas; si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que se ejerzte estudiando los problemas planteados en la sección de ejercicios resueltos y propuestos.

Cuando estudie los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la propuesta en el libro; luego, verifique sus resultados con los del texto; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo que hay ciertas diferencias que puede haber, sobre todo en lo que respecta al nombre de las variables.

En el siguiente capítulo se estudia la estructura de control: la selección, qué es, cómo se representa y cómo se usa en seudocódigo.

**La secuenciación** es una estructura que permite controlar la ejecución de un conjunto de acciones, en orden secuencial, esto es, ejecuta la primera acción, luego la que sigue y así sucesivamente hasta la última, como se muestra a continuación:

1. Acción uno
2. Acción dos
3. Acción tres
4. Acción cuatro
5. Fin

Es decir, es una secuencia de acciones, donde se ejecuta primero la acción uno, después la dos, luego la tres, enseguida la cuatro y por último el fin. Dichas acciones pueden consistir en operaciones primitivas elementales como declaraciones de variables, leer datos, imprimir datos o calcular alguna expresión de acuerdo con los lineamientos descritos en el capítulo anterior. Como puede notarse, es conveniente etiquetar cada acción con números desde el 1 en forma ascendente de uno en uno, para denotar el orden secuencial.

### 3.1 Nuestro primer problema

A continuación se presenta un ejemplo para aplicar los conceptos antes descritos, y además para explicar la forma como se arma un algoritmo.

Ejercicio:

Elaborar un algoritmo para calcular e imprimir el sueldo de un empleado. Siguiendo el proceso de programación se hace lo siguiente:

1. *Definir el problema.*

Calcular el sueldo de un empleado.

2. *Analizar el problema.*

Información por producir: Nombre, Sueldo.

Datos disponibles: Nombre, Horas trabajadas y Cuota por hora.

Proceso a seguir: Sueldo = Horas trabajadas x Cuota por hora.

3. *Diseñar el programa.*

Se diseña la estructura de la solución, elaborando el algoritmo de acuerdo con los lineamientos que se explican en la sección siguiente.

### 3.2 Estructura y diseño de un algoritmo

En esta sección se explica el uso de los elementos que integran la estructura de un algoritmo, aplicándolos al problema definido en la sección anterior.

#### 1. Encabezado

Todo algoritmo debe tener un encabezado como identificación, el cual debe empezar con la palabra **Algoritmo** seguida por una breve descripción de lo que hace. Para el problema que nos ocupa puede ser:

Algoritmo CALCULAR SUELDO DE UN EMPLEADO

#### 2. Declarar

El primer paso en el diseño de un algoritmo consiste en hacer las declaraciones que se necesiten como variables, constantes, tipos de datos, etc. En el caso que nos ocupa se requieren las **variables** siguientes:

nombreEmp tipo Cadena de caracteres para manejar el dato nombre.  
horasTrab tipo Entero para manejar el número de horas trabajadas.  
cuotaHora tipo Real para manejar la cuota que se le paga por hora.  
sueldo tipo Real para manejar el sueldo a pagar al empleado.

Esto se representa en pseudocódigo de la manera siguiente:

```
Declarar
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora: Real
        sueldo: Real
```

### 3. Leer, calcular e imprimir

Los pasos segundo, tercero, etc., pueden consistir en acciones tales como Leer datos, Calcular alguna expresión aritmética e Imprimir datos tantas veces como se requieran y en el orden apropiado para resolver el problema en cuestión.

*Lectura de datos.* En este punto se empiezan a introducir los datos disponibles como materia prima, mediante una operación de lectura precedida por una solicitud de los datos. En nuestro problema esto quedaría así:

```
Solicitar Nombre del empleado, Número de horas trabajadas
y Cuota por hora
Leer nombreEmp, horasTrab, cuotaHora
```

*Hacer cálculos.* El siguiente punto es procesar la entrada para producir la salida, mediante la ejecución de cálculos basados en expresiones aritméticas. En el ejemplo esta acción se expresa así:

```
Calcular sueldo = horasTrab * cuotaHora
```

*Impresión de datos.* El último punto estriba en dar salida a la información requerida, imprimiendo las variables que la contienen. En el ejemplo se expresa así:

```
Imprimir nombreEmp, sueldo
```

### 4. Fin del algoritmo

El último paso del algoritmo consiste en incluir la indicación de Fin.

### 3.3 Nuestro primer algoritmo

A continuación armaremos nuestro primer algoritmo en seudocódigo. El algoritmo servirá para calcular el sueldo de un empleado utilizando la estructura de control SECUENCIACIÓN y todos los demás conceptos explicados hasta el momento.

```

Algoritmo CALCULA SUELDO DE UN EMPLEADO
1. Declarar
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
2. Solicitar Nombre del empleado, Número de horas
   trabajadas y Cuota por hora
3. Leer nombreEmp, horasTrab, cuotaHora
4. Calcular sueldo = horasTrab * cuotaHora
5. Imprimir nombreEmp, sueldo
6. Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C301.C y Programa en Java: Empleado1.java

#### Explicación:

Este algoritmo –como todos– comienza con el encabezado, en el cual se establece una identificación de lo que hace: CALCULA SUELDO DE UN EMPLEADO.

1. Se hacen las declaraciones; se definen las variables:

nombreEmp tipo Cadena para manejar el dato nombre.  
horasTrab tipo Entero para manejar el número de horas trabajadas.  
cuotaHora tipo Real para manejar la cuota que se le paga por hora.  
sueldo tipo Real para manejar el sueldo a pagar al empleado.

Esquemáticamente, en la memoria, las variables quedan definidas así:

nombreEmp	<input type="text"/>
cuotaHora	<input type="text"/>
horasTrab	<input type="text"/>
sueldo	<input type="text"/>

2. Se solicitan los datos nombre del empleado, número de horas trabajadas y cuota por hora; esto significa que en la pantalla se imprime lo siguiente:

Teclee Nombre del empleado:\_  
Teclee Número de horas trabajadas:\_  
Teclee Cuota por hora:\_



El propósito de este libro es enseñar a desarrollar algoritmos en seudocódigo, por tanto, aquí no se explica nada del lenguaje C, ni del lenguaje Java, sin embargo, en la dirección <http://virtual.alfaomega.com.mx> podrá encontrar los programas correspondientes de los algoritmos del libro, codificados en lenguaje C los de los capítulos 3 al 9, que corresponden a la programación estructurada; y en Java, los de los capítulos del 3 al 7 y del 11 al 13. Cabe aclarar, que no obstante que Java es un lenguaje orientado a objetos, es posible programar en Java los algoritmos desarrollados mediante la programación estructurada. Los de los capítulos 8 y 9 no se presentan en Java porque va más allá del propósito y nivel de este libro.



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C301.C y Programa en Java: Empleado1.java



Cuando hay más de una variable de un mismo tipo de dato, se pueden definir juntas separándolas por coma; como cuotaHora, sueldo: Real.

3. Se leen el nombre del empleado, número de horas trabajadas y cuota por hora en las variables nombreEmp, horasTrab y cuotaHora: Esta acción espera a que se le teclee el dato correspondiente a cada petición y los datos tecleados se guardan en las variables. Esquemáticamente queda así:

nombreEmp	Socorro Román Maldonado
cuotaHora	250.00
horasTrab	40
sueldo	

4. Se calcula el sueldo = horasTrab \* cuotaHora. Se multiplican los contenidos de las variables horasTrab por cuotaHora; es decir, 250.00 por 40, y el resultado se coloca en sueldo: Esquemáticamente queda así:

sueldo	10000.00
--------	----------

5. Se Imprimen los datos nombreEmp y sueldo. Es decir, se imprime el contenido de dichas variables, quedando impreso:  
 El contenido de la variable nombreEmp, que es:  
 Socorro Román Maldonado  
 Y el contenido de la variable sueldo, que es: 10000.00

6. Fin del algoritmo. Se termina el algoritmo.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos.

### 3.4 Funciones matemáticas

Todo lenguaje de programación proporciona una gran variedad de funciones estándar, es decir, que ya están definidas por el lenguaje, por ejemplo, las funciones matemáticas, las cuales son funciones de carácter general que nos facilitan la ejecución de ciertos cálculos de índole técnica o científica, ejemplos de estas funciones son: seno, coseno, raíz cuadrada, etc. A continuación se explican con detalle:

Función: Seno

Efecto: Calcula el seno.

Formato:

Seno (x)

En donde:

Seno Es la palabra que identifica a la función.

x Es el parámetro que indica el valor al que se le obtiene el seno. Debe darse en radianes y debe ser de tipo real. El tipo de dato del resultado que devuelve es real.

Ejemplo:

$y = \text{Seno}(x)$

Se calcula el seno de  $x$ , y se le asigna a  $y$ .

Función: Coseno

Efecto: Calcula el coseno.

Formato:

$\text{Coseno}(x)$

*En donde:*

Coseno Es la palabra que identifica a la función.

$x$  Es el parámetro que indica el valor al que se le obtiene el coseno. Debe estar dado en radianes y debe ser de tipo real. El tipo de dato del resultado que devuelve es real.

Ejemplo:

$y = \text{Coseno}(x)$

Se calcula el coseno de  $x$ , y se le asigna a  $y$ .

Función: Arco tangente

Efecto: Calcula el arco tangente.

Formato:

$\text{ArcoTan}(x)$

*En donde:*

ArcoTan Es la palabra que identifica a la función.

$x$  Es el parámetro que indica el valor al que se le obtiene el arco tangente. Debe ser de tipo real. El tipo de dato del resultado es real.

Ejemplo:

$y = \text{ArcoTan}(x)$

Se calcula el arco tangente de  $x$ , y se le asigna a  $y$ .

Función: Logaritmo natural

Efecto: Calcula el logaritmo natural.

Formato:

$\text{Ln}(x)$

*En donde:*

Ln Es la palabra que identifica a la función.

x Es el parámetro que indica el valor al que se le obtiene el logaritmo natural (x debe ser mayor que 0). Debe ser de tipo real.

El tipo de dato del resultado es real.

Ejemplo:

$$y = \ln(x)$$

Se calcula el logaritmo natural de x, y se le asigna a y.

Función: Exponencial

Efecto: Calcula e elevada a la x potencia. Donde e es la base del sistema logarítmico natural ( $e = 2.7182818\dots$ ).

Formato:

$$\text{Exp}(x)$$

*En donde:*

Exp Es la palabra que identifica a la función.

x Es el parámetro que indica la potencia a la que se elevará la constante e. Puede ser de tipo entero o real. El tipo de dato del resultado es real.

Ejemplo:

$$y = \text{Exp}(4)$$

Se calcula e elevada a la potencia 4 y se le asigna a y.

Función: Absoluto

Efecto: Calcula el valor absoluto.

Formato:

$$\text{Absoluto}(x)$$

*En donde:*

Absoluto Es la palabra que identifica a la función.

x Es el parámetro que indica el valor al que se le obtiene el valor absoluto. Puede ser de tipo entero o real.

El tipo de dato del resultado es igual que el tipo del parámetro, es decir, si el parámetro es entero el resultado será entero; si el parámetro es real el resultado será real.

Ejemplo:

```
y = Absoluto(-24.56)
```

Se obtiene el valor absoluto de -24.56 y se le asigna a y.

Función: Raíz cuadrada

Efecto: Calcula la raíz cuadrada.

Formato:

```
RaizCuad(x)
```

*En donde:*

RaizCuad Es la palabra que identifica a la función.

x Es el parámetro que indica el valor al que se le calcula la raíz cuadrada. Puede ser de tipo entero o real.

El tipo de dato del resultado es real.

Ejemplo:

```
y = RaizCuad(16)
```

Se calcula la raíz cuadrada de 16 y se le asigna a y.

Función: Potencia

Efecto: Calcula la potencia de una base elevada a un exponente.

Formato:

```
Potencia(base, exponente)
```

*En donde:*

Potencia Es la palabra que identifica a la función.

base Es un parámetro que indica la base.

exponente Es un parámetro que indica la potencia a la que se elevará la base.

El tipo de dato puede ser entero o real.

Ejemplo:

```
y = Potencia(x, 3)
```

Se calcula la potencia de x elevada a la 3, y se le asigna a y.

Ejercicio 3.4.1

Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes, luego que calcule e imprima el seno y coseno.

*(Primero hágalo usted, después compare la solución.)*

Algoritmo CÁLCULOS LOGARÍTMICOS DE ÁNGULO

1. Declarar
- Variables
- tamAngulo, senAng, cosAng: Real
2. Solicitar Tamaño del ángulo en radianes
3. Leer tamAngulo
4. Calcular senAng = Seno(tamAngulo)  
cosAng = Coseno(tamAngulo)
5. Imprimir senAng, cosAng
6. Fin



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C302.C y Programa en Java: Angulo1.java

*Explicación:*

1. Se declaran las variables:  
tamAngulo para leer el tamaño del ángulo  
senAng para calcular el seno  
cosAng para calcular el coseno
2. Se solicita el Tamaño del ángulo en radianes
3. Se lee el dato en tamAngulo
4. Se calcula el Seno, Coseno
5. Se imprimen los datos de salida
6. Fin del algoritmo

**Ejercicio 3.4.2**

En los lenguajes de programación, las funciones seno, coseno y tangente, requieren que el argumento o parámetro que se les envíe, esté dado en radianes. Si el dato disponible está dado en grados, se puede hacer la conversión a radianes con la siguiente equivalencia:

$$1^\circ = \frac{\pi}{180} = 0.0174775 \text{ radianes}$$

Para convertir de radianes a grados:

$$1 \text{ radián} = \frac{180}{\pi} = 57.21 \text{ grados}$$

Elaborar un algoritmo que permita leer un número en radianes e imprima su equivalencia en grados; asimismo, leer un número en grados e imprima su equivalencia en radianes.

(Primero hágalo usted, después compare la solución.)

```
Algoritmo CONVIERTA RADIANES A GRADOS Y GRADOS A RADIANES
1. Declarar
    Constantes
        PI = 3.14159265
    Variables
        radianes, grados, numRadianes, numGrados: Real
2. Solicitar Número de radianes
3. Leer radianes
4. Solicitar Número de grados
5. Leer grados
6. Calcular numGrados = radianes*(180/PI)
    numRadianes = grados*(PI/180)
7. Imprimir radianes, "EQUIVALEN A", numGrados,
    grados, "EQUIVALEN A", numRadianes,
8. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C303.C y Programa en Java: RadianGrado1.java



*Explicación:*

1. Se hacen las declaraciones

Se define la constante

$$\text{PI} = 3.14159265$$

Se definen las variables

radianes para leer el número de radianes

grados para leer el número de grados

numRadianes para calcular el número de radianes

numGrados para calcular el número de grados

2. Se solicita el número de radianes

3. Se lee el dato en la variable radianes

4. Se solicita el número de grados

5. Se lee el dato en la variable grados

6. Se calcula el número de grados y el número de radianes

7. Se imprimen los datos de salida

8. Fin del algoritmo

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos y estudiar los ejemplos; y si considera que requiere más práctica, puede resolver algunos de los ejercicios propuestos.

### 3.5 Ejercicios resueltos

A continuación se presentan ejercicios resueltos; se recomienda que primero haga Usted el algoritmo, y después compare su solución con la del libro.

#### Ejercicio 3.5.1

Elaborar un algoritmo para calcular el área de un triángulo. Se requiere imprimir como salida el área del triángulo. Los datos disponibles para leer como entrada son la base y la altura del triángulo.

$$\text{El área se calcula: Área} = \frac{\text{Base} \times \text{Altura}}{2}$$

#### Diseño del algoritmo

(Primero hágalo usted, después compare la solución.)

```
Algoritmo ÁREA TRIÁNGULO
1. Declarar
    Variables
    area, base, altura: Real
2. Solicitar Base y Altura
3. Leer base, altura
4. Calcular area = (base * altura) / 2
5. Imprimir area
6. Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C304.C y Programa en Java: AreaTriangulo1.java

#### Explicación:

1. Se declaran las variables:  
base para leer y manejar la base del triángulo  
altura para leer y manejar la altura del triángulo.  
area para calcular el área
2. Se solicitan los datos base y altura

3. Se leen los datos en base y altura
4. Se calcula el área con la expresión:  $(\text{base} * \text{altura}) / 2$   
Nota: En este caso sólo se necesita un cálculo
5. Se imprime el dato obtenido area
6. Fin del algoritmo

#### Ejercicio 3.5.2

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4; de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el Nombre y el promedio de las calificaciones. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4.

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo CALCULA PROMEDIO DE UN ALUMNO
1. Declarar
    Variables
        nombreAlum: Cadena
        calif1, calif2, calif3, calif4, promedio: Real
2. Solicitar Nombre del alumno, calificaciones 1, 2, 3, 4
3. Leer nombreAlum, calif1, calif2, calif3, calif4
4. Calcular promedio = (calif1+calif2+calif3+calif4)/4
5. Imprimir nombreAlum, promedio
6. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C305.C y Programa en Java: Alumno1.java



#### Explicación:

1. Se declaran las variables:  
nombreAlum para leer y manejar el nombre del alumno  
calif1, calif2, calif3, calif4 para leer las cuatro calificaciones  
promedio para calcular el promedio
2. Se solicitan los datos nombre del alumno, y cuatro calificaciones
3. Se leen el nombre del alumno y las cuatro calificaciones
4. Se calcula el promedio final
5. Se imprimen los datos nombreAlum y promedio
6. Fin del algoritmo

**Ejercicio 3.5.3**

Elaborar un algoritmo que calcule e imprima el precio de venta de un artículo. Se tienen los datos Descripción del artículo y Costo de producción. El precio de venta se calcula añadiéndole al costo el 120% como utilidad y el 15% de impuesto.

*(Primero hágalo usted, después compare la solución.)*

```

Algoritmo PRECIO DE VENTA
1. Declarar
    Variables
        descripcion: Cadena
        costo, precioVta: Real
2. Solicitar Descripcion del articulo, Costo
3. Leer descripcion, costo
4. Calcular precioVta = costo + (costo*1.2)
   + ((costo+(costo*1.2))*0.15)
5. Imprimir descripcion, precioVta
6. Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C306.C y Programa en Java: PrecioArticulo1.java

**Explicación:**

1. Se declaran las variables  
descripcion para leer y manejar la descripción  
costo para leer el costo del artículo  
precioVta para calcular el precio de venta
2. Se solicitan los datos Descripción del artículo y Costo
3. Se leen los datos
4. Se calcula el precio de venta
5. Se imprimen los datos descripcion y precioVta
6. Fin del algoritmo

**Ejercicios**

En la Web del libro están disponibles los siguientes ejercicios resueltos.

**Tabla 3.1**

Ejercicio	Descripción
Ejercicio 3.5.4	Calcula la hipotenusa de un triángulo rectángulo
Ejercicio 3.5.5	Convierte temperaturas Fahrenheit
Ejercicio 3.5.6	Calcula el volumen de un cilindro
Ejercicio 3.5.7	Calcula el tamaño de un ángulo
Ejercicio 3.5.8	Calcula el cateto B de un triángulo rectángulo
Ejercicio 3.5.9	Realiza cálculos logarítmicos
Ejercicio 3.5.10	Realiza cálculos logarítmicos

**Código fuente:**

En la zona de descarga está disponible para su uso el código de los ejercicios resueltos:

Programa en C: C307.C y Programa en Java: Hipotenusa1.java.

Programa en C: C308.C y Programa en Java: Temperaturas.java

Programa en C: C309.C y Programa en Java: Cilindro.java

Programa en C: C310.C y Programa en Java: TamAngulo.java

Programa en C: C311.C y Programa en Java: CatetoB.java

Programa en C: C312.C y Programa en Java: Angulo2.java

Programa en C: C313.C y Programa en Java: Angulo3.java

### **3.6 Ejercicios propuestos**

1. Elaborar un algoritmo que calcule e imprima el costo de producción de un artículo, teniendo como datos la descripción y el número de unidades producidas. El costo se calcula multiplicando el número de unidades producidas por un factor de costo de materiales de 3.5 y sumándole al producto un costo fijo de 10 700.
2. Elaborar un algoritmo que calcule e imprima el costo de un terreno cuadrado o rectangular, teniendo como datos la anchura y la longitud en metros, y el costo del metro cuadrado.
3. Elaborar un algoritmo que lea una cantidad de horas e imprima su equivalente en minutos, segundos y días.
4. Similar al del alumno (Ejercicio 3.5.2), con la diferencia de que en lugar del promedio se obtiene una calificación final multiplicando las calificaciones 1,2,3 y 4 por los porcentajes 30, 20, 10 y 40%, respectivamente, y sumando los productos.
5. La velocidad de la luz es de 300 000 kilómetros por segundo. Elaborar un algoritmo que lea un tiempo en segundos e imprima la distancia que recorre en dicho tiempo.
6. Hacer un algoritmo que obtenga e imprima el valor de Y a partir de la ecuación:  
$$Y = 3X^2 + 7X - 15$$
solicitando como dato de entrada el valor de X.
7. Una temperatura en grados Celsius (C) se puede convertir a su equivalente Fahrenheit (F) con la fórmula:

$$F = \frac{9}{5}C + 32$$

De Fahrenheit a Celsius con la fórmula:

$$C = F - 32 - \frac{5}{9}$$

Elaborar un algoritmo que lea una temperatura en grados centígrados y obtenga e imprima la temperatura Fahrenheit equivalente.

8. Elabore un algoritmo que lea un número de pies y calcule e imprima su equivalente en yardas, pulgadas, centímetros y metros, de acuerdo con las siguientes equivalencias: 1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 metro = 100 cm.
9. Elaborar un algoritmo que lea el artículo y su costo; la utilidad es el 150% y el impuesto es el 15%; calcular e imprimir artículo, utilidad, impuesto y precio de venta.
10. Elaborar un algoritmo que lea el radio de un círculo e imprima el área.  

$$\text{ÁREA} = \pi r^2$$
11. Elaborar un algoritmo que lea la cantidad de dólares a comprar y el tipo de cambio en pesos (costo de un dólar en pesos); calcular e imprimir la cantidad a pagar en pesos por la cantidad de dólares indicada.
12. Elaborar un algoritmo que permita leer valores para X, Y, Z y W e imprima el valor de F.

$$F = \frac{(4x^2y^2 \sqrt{2zw})^2}{\frac{4x^{\frac{1}{2}}}{b^{\frac{3}{4}}}}$$

13. Elaborar un algoritmo que lea el radio(r) de una esfera, calcule e imprima el volumen y el área.

$$\text{VOLUMEN} = \frac{4\pi r^3}{3} \quad \text{ÁREA} = \pi r^2$$

14. Elaborar un algoritmo que lea el valor de W e imprima el valor de Z.

$$Z = \frac{1}{\sqrt{2\pi}} e^{\frac{w^2}{2}}$$

15. Elaborar un algoritmo que lea la cantidad de dólares a comprar y el tipo de cambio (costo de un dólar) en: yenes, pesetas, libras esterlinas y marcos; calcular e imprimir la cantidad a pagar en yenes, pesetas, libras esterlinas y marcos.
16. Elaborar un algoritmo que permita leer un valor e imprima el logaritmo natural, el exponencial, el valor absoluto y la raíz cuadrada.
17. Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes e imprima la tangente, cotangente, secante y cosecante.

$$\text{tangente} = \frac{\text{seno}}{\text{coseno}}$$

$$\text{secante} = \frac{1}{\text{coseno}}$$

$$\text{cotangente} = \frac{\text{coseno}}{\text{seno}}$$

$$\text{cosecante} = \frac{1}{\text{seno}}$$

18. Elaborar un algoritmo similar al anterior; sólo que el dato que se lee estará dado en grados. Debe convertirse los grados leídos a radianes antes de hacer los cálculos.
19. Elaborar un algoritmo que permita leer el tamaño de un ángulo en grados e imprima el seno y coseno. Debe convertirse los grados leídos a radianes antes de hacer los cálculos.
20. Elaborar un algoritmo que permita leer valores para A y B e imprima Y, Z y W.

$$Y = 3a^2b^2 \sqrt{2a} \quad W = 4\sqrt{2^a a} (3a^2b^2 - \sqrt{2a})$$

$$Z = \frac{12a^{\frac{1}{2}}}{b^{\frac{3}{4}}}$$

### **3.7 Resumen de conceptos que debe dominar**

- La secuenciación.
- Estructura y diseño de un algoritmo en seudocódigo.
- Funciones matemáticas (seno, coseno, arco tangente, logaritmo natural, exponencial, elevar a potencias, raíz cuadrada, etcétera).

### **3.8 Contenido de la página Web de apoyo**



El material marcado con asterisco (\*) sólo está disponible para docentes.

#### **3.8.1 Resumen gráfico del capítulo**

#### **3.8.2 Autoevaluación**

#### **3.8.3 Programas**

#### **3.8.4 Power Point para el profesor (\*)**

# 4

## La selección

### Contenido

- 4.1 La selección doble (if-then-else)
  - 4.1.1 Sangrado (indentación) y etiquetas
  - 4.1.2 Expresiones lógicas
  - 4.1.3 if's anidados
  - 4.1.4 Ejercicios resueltos para la selección doble (if-then-else)
- 4.2 La selección simple (if-then)
  - 4.2.1 Ejercicios resueltos para la selección simple (if-then)
- 4.3 La selección múltiple (switch)
  - 4.3.1 Ejercicio resuelto para la selección múltiple (switch)
- 4.4 Ejercicios propuestos
- 4.5 Resumen de conceptos que debe dominar
- 4.6 Contenido de la página Web de apoyo
  - El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 4.6.1 Resumen gráfico del capítulo
  - 4.6.2 Autoevaluación
  - 4.6.3 Programas
  - 4.6.4 Ejercicios resueltos
  - 4.6.5 Power Point para el profesor (\*)

### Objetivos del capítulo

- Elaborar algoritmos utilizando la estructura de control selección doble (if-then-else).
- Elaborar algoritmos utilizando expresiones lógicas simples ( $>$ ,  $<$ ,  $\geq$ ,  $\leq$ ,  $\neq$ ,  $=$ ) y complejas (AND, OR, XOR, NOT),
- Elaborar algoritmos utilizando la anidación de if's, la selección simple (if-then) y la selección múltiple (switch).

## Introducción

Con el estudio del capítulo anterior, usted ya domina la secuenciación y cómo diseñar algoritmos usando esa estructura de control.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando la estructura de control la selección.

Se explica que la selección es una estructura que permite tomar decisiones y que tiene tres formas: la selección doble (if-then-else), la selección simple (if-then) y la selección múltiple(switch).

Se enseña que la selección doble (if-then-else) sirve para plantear situaciones en las que se tienen dos alternativas de acción, de las cuales se debe escoger una. Se describe cómo plantear expresiones lógicas simples usando los operadores relacionales ( $>$ ,  $<$ ,  $\geq$ ,  $\leq$ ,  $\neq$ ,  $\equiv$ ); y expresiones lógicas complejas usando los operadores lógicos (AND, OR, XOR, NOT).

Se detalla que la selección simple (if-then) sirve para plantear situaciones en las que se tiene una alternativa de acción.

Se explica que la selección múltiple(switch) sirve para plantear situaciones en las que se tienen múltiples alternativas de acción, de las cuales se debe escoger una.

Asimismo, se plantea la anidación de if's; es decir, que un if tenga dentro otro if, y así sucesivamente.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejerzte estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que, a veces, aunque haya diferencias, las dos soluciones son correctas.

En el siguiente capítulo se estudia la estructura de control la repetición, qué es, cómo se representa y cómo se usa en pseudocódigo.

La *selección* es una estructura que permite controlar la ejecución de acciones que requieren de ciertas condiciones para su realización. De acuerdo con dichas condiciones se “selecciona” si las acciones se ejecutan o no. En ocasiones se tienen operaciones que son excluyentes, es decir, que sólo tiene que ejecutarse una o la otra, pero no ambas de manera simultánea; también puede presentarse el caso de que se tengan varias opciones de acción. En estos casos es necesario utilizar la estructura de control de selección. Para una mejor comprensión del concepto, veamos el siguiente ejemplo:

Ejemplo:

Calcular el sueldo de un empleado.

*Información a producir:* Nombre y Sueldo

*Datos disponibles:* Nombre, número de horas trabajadas y cuota por hora

*Proceso:* El sueldo se calcula de la forma siguiente:

Si el número de horas trabajadas es mayor que 40, el excedente de 40 se paga al doble de la cuota por hora.  
En caso de no ser mayor que 40, se paga a la cuota por hora normal.

En esta situación, el sueldo se calcula de dos formas distintas:

1. Si es menor o igual que 40:

Sueldo = Número de horas trabajadas X cuota por hora

2. Si es mayor que 40:

Sueldo =  $(40 \times \text{cuota por hora}) + ((\text{número de horas trabajadas} - 40) \times (\text{cuota por hora} \times 2))$

Por lo tanto, es necesario utilizar la estructura de selección para calcular el Sueldo de la primera forma si el número de horas trabajadas es menor o igual que 40, o de la segunda en el otro caso.

La estructura de selección tiene tres formas: **simple**, **doble** y **múltiple**, de acuerdo con el número de alternativas de acción; es decir, si hay una, dos o más de dos, respectivamente. Por razones didácticas, primero se estudiará la **doble**, después la **simple** y por último la **múltiple**.

## 4.1 La selección doble (if-then-else)

Esta estructura de selección permite controlar la ejecución de acciones cuando se tienen dos opciones alternativas de acción, y por la naturaleza de éstas, se debe ejecutar una o la otra, pero no ambas a la vez, es decir, son mutuamente excluyentes.

*Formato:*

```
if condición then
    Acción(es)
else
    Acción(es)
endif
```

*En donde:*

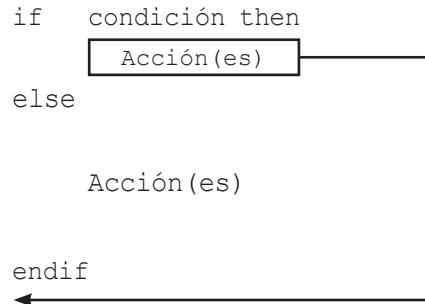
if (Si) Identifica la estructura de selección.

condición	Es una expresión lógica que denota la situación específica mediante la comparación de dos operandos para dar un resultado booleano (falso, verdadero); es decir, si se cumple o no se cumple. En el punto 4.1.2 Expresiones lógicas se explica con detalle.
then (Entonces)	Indica el curso de acción si se cumple la condición.
Acción(es)	Es la acción o conjunto de acciones en pseudocódigo que se ejecutarán en el bloque correspondiente.
else (si no; caso contrario)	Indica el curso de acción cuando no se cumple la condición.
endif	Indica el fin de la estructura de selección (del if).

*Funcionamiento:*

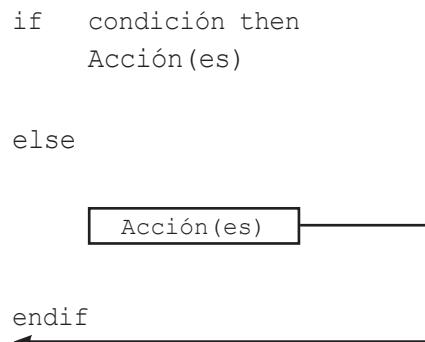
Al llegar al `if` se evalúa la condición:

- a) Opción verdadera (`then`).  
 Si se cumple, se ejecuta(n)  
 la(s) acción(es) del `then` y  
 luego salta hasta la siguiente  
 después del `endif` (fin del `if`).



**Nota:** En este caso ejecuta la opción del `then` e ignora la del `else`.

- b) Opción falsa (`else`).  
 De lo contrario, salta hacia  
 el `else`, ejecuta la(s)  
 acción(es), y después salta  
 a la siguiente acción  
 después del `endif` (fin del `if`).



**Nota:** En este caso ejecuta la opción del `else` e ignora la del `then`.

Sugerencia: En este momento se sugiere ir al punto 4.1.2 y estudiar la parte correspondiente a Expresiones lógicas simples y regresar a este punto.

Aplicando lo anterior al ejemplo de cálculo de sueldo, queda:

```
if horasTrab <= 40 then
    sueldo = horasTrab * cuotaHora
else
    sueldo = (40*cuotaHora)+( (horasTrab-40) * (cuotaHora*2) )
endif
```

La condición se plantea comparando `horasTrab` (horas trabajadas) con 40, mediante el operador relacional `<=` “menor o igual que” [esto se explica con todo detalle dos puntos más adelante (4.1.2 Expresiones lógicas)]. En caso de cumplirse la condición, se ejecutará la acción o acciones de la opción `then` y, de no cumplirse, se ejecutará la acción o acciones de la opción `else`.

A continuación se presenta el algoritmo completo para calcular el sueldo del empleado, considerando el doble para el excedente de 40 horas trabajadas.

Algoritmo CÁLCULO SUELDO DOBLE

1. Declarar
 

Variables

  - nombreEmp: Cadena
  - horasTrab: Entero
  - cuotaHora, sueldo: Real
2. Solicitar Nombre del empleado,  
número de horas trabajadas y cuota por hora
3. Leer nombreEmp, horasTrab, cuotaHora
4. if horasTrab <= 40 then
  - a. sueldo = horasTrab \* cuotaHora
5. else
  - a. sueldo = (40\*cuotaHora)+  
((horasTrab-40) \* (cuotaHora\*2))
6. endif
7. Imprimir nombreEmp, sueldo
8. Fin

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C401.C y Programa en Java: Empleado2.java



#### *Explicación:*

1. Se declaran las variables que ya conocemos
2. Se solicita el Nombre del empleado, número de horas trabajadas y cuota por hora



Observe que tanto en el `then` como en el `else`, la acción de calcular el sueldo está enumerada con el inciso a; en el siguiente punto se explica por qué.

3. Se leen los datos en `nombreEmp`, `horasTrab` y `cuotaHora`
4. Se compara si `horasTrab <= 40` si se cumple, entonces
  - a. Se calcula el Sueldo de la forma simple
5. Si no se cumple (en caso contrario)
  - a. Se calcula el Sueldo considerando horas dobles
6. Finaliza el `if (endif)`
7. Se imprime el `nombreEmp` y el sueldo
8. Fin del algoritmo

#### 4.1.1 Sangrado (indentación) y etiquetas

Las instrucciones o acciones de los algoritmos pueden etiquetarse con números y letras, alternativamente, para identificar más fácilmente su orden en los diversos niveles de subordinación.

Si vemos el ejemplo que se da más adelante, la primera acción se denota con el 1, la segunda con el 2, la tercera con el 3 y así sucesivamente. En las acciones 1, 3 y 4 existen subacciones o acciones subordinadas, éstas se encuentran en un siguiente nivel de subordinación, mismas que se jerarquizan dejando una sangría y se etiquetan con una letra. La acción 4A, tiene a su vez otras dos acciones subordinadas, y así, al haber otro nivel de subordinación, las acciones se vuelven a etiquetar con números, para usar alternativamente números, letras, números, letras, etc. Con base en lo anterior, la estructura que se tendría sería la siguiente:

##### Algoritmo EJEMPLO SANGRADO (INDENTACIÓN) Y ETIQUETAS

1. Acción 1
  - a. Acción 1A
  - b. Acción 1B
2. Acción 2
3. Acción 3
  - a. Acción 3A
4. Acción 4
  - a. Acción 4A
    1. Acción 4A1
    2. Acción 4A2
  - b. Acción 4B
    1. Acción 4B1
  - c. Acción 4C
5. Acción 5
6. Fin

Como podemos ver, la jerarquía y subordinación se denota dejando sangrados y etiquetados con números y letras, alternativamente, en los diferentes niveles.

## 4.1.2 Expresiones lógicas

En otros dos puntos se hace la sugerencia que venga a este punto, es decir, se le va a enviar a estudiar este punto en dos ocasiones, para que estudie una parte en una ocasión y la otra en la siguiente; la idea es que después de estudiar la parte correspondiente, debe regresar.

Las expresiones lógicas sirven para plantear condiciones mediante la comparación de dos o más operandos que dan como resultado un valor booleano verdadero o falso; es decir, se cumple o no se cumple la condición. Se pueden clasificar en simples y complejas.

### Expresiones lógicas simples

Se forman relacionando operandos, variables y/o constantes mediante operadores relacionales, de la forma siguiente:

Expresión lógica simple =

**Operando1 Operador relacional Operando2**

*En donde:*

**Operando1, Operando2** Son variables o valores constantes de algún tipo de datos Entero, Real, Cadena, etc. En una expresión, ambos operandos deben ser del mismo tipo de dato.

#### Operador relacional

Cualquiera de los siguientes:

Operadores relacionales	Significado
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a o igual que
!=	Diferente de o no igual a

Ejemplos:

- x == 1 significa ¿x es igual a 1?
- n != z significa ¿n es no igual a z?
- n >= 5 significa ¿n es mayor o igual a 5?
- z <= 5 significa ¿n es menor o igual a 5?
- y == z significa ¿y es igual a z?
- nombre == "Socorro Román Maldonado" significa  
¿nombre es igual a Socorro Román Maldonado?

Como ya mencionamos, los posibles resultados para cada una de las expresiones lógicas son verdadero (V –True–) o falso (F –False–), es decir, Sí se cumple o NO se cumple.



A algunos maestros no les gusta usar etiquetas para enumerar los pasos de los algoritmos, yo considero que es muy bueno para personas que recién se están formando, porque da una mayor disciplina a la persona y un orden más entendible a los algoritmos. Una vez que la persona sabe programar, podrá o no usar etiquetas. En el presente libro se recomienda el uso de las etiquetas, pero si desea, puede omitirlas. En el Apéndice A (Algoritmos sin usar etiquetas), se presentan algunos algoritmos sin usar etiquetas. Sin embargo, estimado maestro, si Usted está utilizando este libro como texto, es conveniente que utilice los ejemplos como están planteados en el libro, es decir, con etiquetas, porque si Usted explica los algoritmos sin etiquetas, y luego cuando los alumnos estudien el libro y vean los algoritmos con etiquetas, seguramente les causará confusión, y en consecuencia, la utilidad del libro podría ser limitada.



Observe que el doble signo de igual == se utiliza para plantear comparaciones, es decir, expresiones lógicas; y un signo de igual = se utiliza para plantear cálculos a través de expresiones aritméticas o fórmulas.

### Expresiones lógicas complejas

Se forman relacionando operandos booleanos mediante operadores lógicos, como sigue:

Expresión lógica compleja =

**Operando booleano1 operador lógico Operando booleano2**

En donde:

**Operando booleano1**, Son expresiones lógicas que

**Operando booleano2** proporcionan un valor verdadero (V) o falso (F).

**operador lógico** Cualquiera de los siguientes: AND, OR, XOR, NOT.



Para un mejor tratamiento del tema, se ha decidido incluir en las tablas de valor V para los valores verdaderos, sin embargo, los lenguajes de programación no emiten este valor como tal, sino una T, que es la sigla de la palabra inglesa *True*, que en español significa verdad.

### AND

Relaciona dos operandos booleanos. Proporciona un valor verdadero (V), si los dos son verdaderos (V); en caso contrario da un resultado falso (F).

Si se hace la suposición de que EXP1 y EXP2 son expresiones booleanas y se plantea la expresión:

(EXP1) AND (EXP2) Proporciona las combinaciones de resultados mostrados en la siguiente tabla:

EXP1	EXP2	(EXP1) AND (EXP2)
V	V	V
V	F	F
F	V	F
F	F	F

Ejemplo:

Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones denotadas como c1 y c2. El aspirante que obtenga calificaciones de 80 ó mayores en ambos exámenes es aceptado; en caso contrario es rechazado.

```
if (c1 >= 80) AND (c2 >= 80) then
    Imprimir "Aceptado"
else
    Imprimir "Rechazado"
endif
```

### OR

Relaciona dos operandos booleanos. Proporciona un valor verdadero (V), si uno de los dos es verdadero (V); en caso contrario da un resultado falso (F). En la siguiente tabla se presentan las posibles combinaciones:

EXP1	EXP2	(EXP1) OR (EXP2)
V	V	V
V	F	V
F	V	V
F	F	F

Ejemplo:

Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones denotadas como  $c_1$  y  $c_2$ . El aspirante que obtenga una calificación mayor que 90 en cualquiera de los exámenes es aceptado; en caso contrario es rechazado.

```
if ( $c_1 > 90$ ) OR ( $c_2 > 90$ ) then
    Imprimir "Aceptado"
else
    Imprimir "Rechazado"
endif
```

## XOR

Relaciona dos operandos booleanos. Proporciona un resultado verdadero (V) si uno de los dos es verdadero (V), pero no ambos; en caso contrario da un valor falso (F). En la siguiente tabla se presentan las posibles combinaciones.

EXP1	EXP2	(EXP1) XOR (EXP2)
V	V	F
V	F	V
F	V	V
F	F	F

Ejemplo:

Una empresa aplica dos exámenes a sus aspirantes a ingresar como trabajadores, por lo que cada uno de ellos obtiene dos calificaciones denotadas como  $c_1$  y  $c_2$ . El aspirante que obtenga 100 en cualquiera de los exámenes, pero no en ambos, es aceptado; en caso contrario es rechazado.

```
if ( $c_1 == 100$ ) XOR ( $c_2 == 100$ ) then
    Imprimir "Aceptado"
else
    Imprimir "Rechazado"
endif
```



Aunque suena ilógico rechazar a quien haya obtenido 100 en ambos exámenes, supongamos que si obtiene 100 en ambos, el aspirante le queda grande al puesto; esto provocaría que no desempeñe su puesto con entusiasmo.

## NOT

Este operador relaciona sólo un operando booleano y da como resultado un valor opuesto al que tenga el operando. En la siguiente tabla se presentan las posibles combinaciones.

EXP	NOT (EXP)
V	F
F	V

Ejemplo:

Un alumno tiene una calificación final (calFin) y se desea imprimir el resultado de aprobado si la calificación es igual o mayor a 70; o bien, reprobado en caso contrario.

Solución sin usar NOT:

```
if calFin >= 70 then
    Imprimir "Aprobado"
else
    Imprimir "Reprobado"
endif
```

Solución usando NOT:

```
if NOT(calFin >= 70) then
    Imprimir "Reprobado"
else
    Imprimir "Aprobado"
endif
```

El operador lógico NOT sirve para cambiar el resultado de una expresión lógica.

Presentamos el orden de precedencia de los operadores relacionales y lógicos.

1. Paréntesis ( )
2. NOT
3. AND
4. OR, XOR
5. <, >, ==, <=, >=, !=

### 4.1.3 If's anidados

Una estructura de selección (if) puede tener anidada a otra y ésta a otra y así sucesivamente, ejemplo:

```
if condición then
    if condición then
        Acción(es)
    else
```

```
    Acción(es)
endif
else
    if condición then
        Acción(es)
    else
        Acción(es)
    endif
endif
```

Se tiene un **if** principal el cual tiene anidado en el **then** un **if**; mismo que tiene su propio **then**, **else** y **endif**. Por el **else** también hay un **if** anidado que contiene su **then**, **else** y **endif**.

Otro ejemplo de anidación sería el caso de tener una instrucción simple por el **then** y un **if** por el **else**, la estructura quedaría:

```
if condición then
    Acción(es)
else
    if condición then
        Acción(es)
    else
        Acción(es)
    endif
endif
```

Vemos que se tiene el **then** donde no lleva **if** anidado, mientras que por el **else** se tiene el **if** anidado.

Otra alternativa de anidación es el caso en que se tenga un **if** anidado por el **then**, mientras que por el **else** no se tiene **if** anidado, el esqueleto de la estructura es:

```
if condición then
    if condición then
        Acción(es)
    else
        Acción(es)
    endif
else
    Acción(es)
endif
```

**Ejercicio:**

Elaborar un algoritmo similar al anterior de CÁLCULO SUELDO DOBLE, pero ahora tomando en cuenta que se tiene otra alternativa: Las horas que exceden de 50 se pagan al triple de la cuota por hora.

A continuación se presenta el algoritmo:

*(Primero hágalo usted, después compare la solución.)*

```

Algoritmo CÁLCULO SUELDO TRIPLE
1. Declarar
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
2. Solicitar Nombre del empleado, número de horas
   trabajadas y cuota por hora
3. Leer nombreEmp, horasTrab, cuotaHora
4. if horasTrab <= 40 then
   a. sueldo = horasTrab * cuotaHora
5. else
   a. if horasTrab <= 50 then
      1. sueldo = (40*cotaHora) +
                  ((horasTrab-40) * (cotaHora*2))
   b. else
      1. sueldo = (40*cotaHora)+(10*cotaHora*2)
                  +( (horasTrab-50) * (cotaHora*3))
   c. endif
6. endif
7. Imprimir nombreEmp, sueldo
8. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C402.C y Programa en Java: Empleado3.java

**Explicación:**

1. Se declaran las variables que ya conocemos
2. Se solicita el Nombre del empleado, número de horas trabajadas y cuota por hora
3. Se leen los datos en nombreEmp, horasTrab y cuotaHora
4. Se compara si horasTrab <= 40, si es así, entonces
  - a. Se calcula el sueldo de la forma simple

5. Si no, quiere decir que horasTrab es mayor que 40
  - a. Se compara si horasTrab <= 50, si es así, entonces
    1. Se calcula el sueldo; 40 horas a la cuota por hora normal, más el excedente de 40 a la cuota por hora al doble (por 2)
  - b. Si no, quiere decir que horasTrab es mayor que 50
    1. Se calcula de la forma triple: 40 horas a la cuota por hora normal, más 10 horas a la cuota por hora al doble (por 2), más el excedente de 50 a la cuota por hora al triple (por 3)
  - c. Fin del if interno
6. Fin del if principal
7. Se imprimen los datos de salida
8. Fin del algoritmo

#### 4.1.4 Ejercicios resueltos para la selección doble (if-then-else)

A continuación se presentan ejercicios resueltos; se le recomienda que primero haga Usted el algoritmo, y después compare su solución con la del libro.

##### Ejercicio 4.1.4.1

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4, de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el Nombre, el promedio de las calificaciones y un comentario de “Aprobado” si obtiene 60 o más, o “Reprobado” en caso contrario. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4. Este problema lo vamos a solucionar de tres formas distintas.

*(Primero hágalo usted, después compare la solución.)*

##### Primer método de solución: Usando la variable observación.

```
Algoritmo CALCULA PROMEDIO DE UN ALUMNO
1. Declarar
    Variables
        nombreAlum: Cadena
        calif1, calif2, calif3, calif4, promedio: Real
        observacion: Cadena
2. Solicitar Nombre del alumno, calificación 1,2,3 y 4
3. Leer nombreAlum, calif1, calif2, calif3, calif4
4. Calcular promedio = (calif1+calif2+calif3+calif4)/4
5. if promedio >= 60 then
    a. observacion = "Aprobado"
6. else
    a. observacion = "Reprobado"
7. endif
```



En este momento se recomienda ir al siguiente punto de ejercicios resueltos para la selección doble (if-then-else) y estudiar algunos ejemplos. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el if-then-else.

- ```

8. Imprimir nombreAlum, promedio, observacion
9. Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C403.C y Programa en Java: Alumno2.java

*Explicación:*

1. Se declaran las variables
2. Se solicitan el nombre y las cuatro calificaciones
3. Se leen los datos en las variables correspondientes
4. Se calcula el promedio
5. Se compara si promedio  $\geq 60$ , si se cumple, entonces
  - a. En la variable observacion se coloca el valor "Aprobado"
6. Si no se cumple
  - a. En la variable observacion se coloca el valor "Reprobado"
7. Fin del if
8. Se imprimen los datos nombreEmp, promedio, observacion
9. Fin del algoritmo



En la zona de descarga del capítulo 4 de la Web del libro, está disponible el ejercicio resuelto.

| Ejercicio         | Descripción                                               |
|-------------------|-----------------------------------------------------------|
| Ejercicio 4.1.4.1 | Dos métodos alternativos de solución al problema anterior |

#### Ejercicio 4.1.4.2

Elaborar un algoritmo que lea dos números y que imprima el mayor. Se supone que son números diferentes, por lo tanto, no se debe averiguar si son iguales o si son diferentes.

*(Primero hágalo usted, después compare la solución.)*

```

Algoritmo MAYOR 2 NÚMEROS
1. Declarar
   Variables
   a, b: Entero
2. Solicitar número 1, número 2
3. Leer a, b
4. if a > b then
   a. Imprimir a
5. else
   a. Imprimir b
6. endif
7. Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C406.C y Programa en Java: Mayor2Numeros.java



*Explicación:*

1. Se declaran las variables
2. Solicitar los dos números
3. Se leen en las variables a y b
4. Se compara si  $a > b$ , si es así, entonces
  - a. Se imprime a como el mayor
5. Si no (en caso contrario)
  - a. Se imprime b como el mayor
6. Fin del if
7. Fin del algoritmo

#### Ejercicio 4.1.4.3

Elaborar un algoritmo que lea tres números y que imprima el mayor. Se supone que son números diferentes, por lo tanto, no se debe averiguar si son iguales o si son diferentes. Este problema lo vamos a solucionar de tres formas distintas.

**Primer método de solución:** Utilizando if-then-else y expresiones lógicas simples, es decir, sin usar AND.

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo MAYOR 3 NÚMEROS
1. Declarar
    Variables
    a, b, c: Entero
2. Solicitar número 1, número 2, número 3
3. Leer a, b, c
4. if a > b then
    a. if a > c then
        1. Imprimir a
    b. else
        1. Imprimir c
    c. endif
5. else
    a. if b > c then
        1. Imprimir b
    b. else
        1. Imprimir c
    c. endif
```

6. endif
7. Fin



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C407.C y Programa en Java: Mayor3Numeros1.java

*Explicación:*

1. Se declaran las variables
2. Se solicitan los tres números
3. Se leen en a, b y c
4. Se compara si  $a > b$ , si es así, entonces
  - a. Se compara si  $a > c$ , si es así, entonces
    1. Se imprime a como el mayor
  - b. Si no
    1. Se imprime c como el mayor
  - c. Fin del if
5. Si no
  - a. Se compara si  $b > c$  entonces
    1. Se imprime b como el mayor
  - b. Si no
    1. Se imprime c como el mayor
  - c. Fin del if
6. Fin del if
7. Fin del algoritmo

En la zona de descarga del capítulo 4 de la Web del libro, está disponible el ejercicio 4.1.4.4. Lee cuatro números e imprime el mayor

Sugerencia: En este momento se sugiere ir al punto 4.1.2 y estudiar la parte correspondiente a Expresiones lógicas complejas (si no lo ha hecho) y regresar a este punto.

Ejercicio 4.1.4.5

Elaborar un algoritmo que lea tres números y que imprima el mayor. Se supone que son números diferentes. Este problema es la segunda vez que lo vamos a solucionar.

**Segundo método de solución:** Utilizando if-then-else y AND.

(Primero hágalo usted, después compare la solución.)

**Algoritmo MAYOR 3 NÚMEROS**

```

1. Declarar
    Variables
        a, b, c: Entero
2. Solicitar número 1, número 2, número 3
3. Leer a, b, c
4. if (a > b)AND(a > c) then
    a. Imprimir a
5. else
    a. if b > c then
        1. Imprimir b
    b. else
        1. Imprimir c
    c. endif
6. endif
7. Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C409.C y Programa en Java: Mayor3Numeros2.java


*Explicación:*

1. Se declaran las variables
2. Se solicitan los tres números
3. Se leen en a, b y c
4. Se compara si (a > b) y (a > c), si se cumple, entonces
  - a. Se imprime a como el mayor
5. Si no
  - a. Se compara b > c, si se cumple, entonces
    1. Se imprime b como el mayor
  - b. Si no
    1. Se imprime c como el mayor
  - c. Fin del if
6. Fin del if
7. Fin del algoritmo

La tabla 4.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 4 de la Web del libro.

**Tabla 4.1**

| Ejercicio         | Descripción                                       |
|-------------------|---------------------------------------------------|
| Ejercicio 4.1.4.6 | Lee cuatro números e imprime el mayor             |
| Ejercicio 4.1.4.7 | Lee tres lados y dice el tipo de triángulo que es |
| Ejercicio 4.1.4.8 | Realiza cálculos con la ecuación cuadrática       |

## 4.2 La selección simple (if-then)

Esta estructura de selección permite controlar la ejecución de acciones cuando existe una sola alternativa de acción. Se utiliza cuando alguna acción o conjunto de acciones está condicionada para que se lleve a cabo su ejecución, pero no se tiene una opción alterna.

*Formato:*

```
if condición then
    Acción(es)
endif
```

*En donde:*

|                 |                                                               |
|-----------------|---------------------------------------------------------------|
| if (Si)         | Identifica la estructura de control de selección.             |
| then (Entonces) | Indica el curso de acción a seguir si se cumple la condición. |
| endif           | Indica el fin de la estructura de selección (del if).         |

*Funcionamiento:*

Al llegar al if se evalúa la condición:

- a) Si se cumple, se ejecuta(n) if condición then  
la(s) acción(es) del then y  
luego salta hasta la siguiente  
después del endif (fin del if). 
- b) Si no se cumple, salta hasta después del endif, es decir, no hace nada.

*Ejercicio:*

Siguiendo con el mismo problema de cálculo del sueldo, ahora se otorga un incentivo del 5% del sueldo, si el empleado trabajó más de 40 horas, es decir, al sueldo se le agrega el 5% del mismo sueldo.

A continuación se tiene el algoritmo:

```
Algoritmo CÁLCULO SUELDO CON INCENTIVO
```

1. Declarar
 

```
Variables
          nombreEmp: Cadena
          horasTrab: Entero
          cuotaHora, sueldo: Real
```
2. Solicitar Nombre del empleado, número de horas trabajadas y cuota por hora
3. Leer nombreEmp, horasTrab, cuotaHora
4. sueldo = horasTrab \* cuotaHora

```
5. if horasTrab > 40 then
   1. sueldo = sueldo + (sueldo * 0.05)
6. endif
7. Imprimir nombreEmp, sueldo
8. Fin
```

En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C413.C y Programa en Java: Empleado4.java



Como se puede ver, una vez calculado el sueldo se verifica si trabajó más de 40 horas, si es así, se le adiciona el 5% del mismo sueldo. ( Nótese que aquí no aparece otra alternativa.)

Práctica: En este momento se recomienda ir al siguiente punto de ejercicios resueltos para la selección simple (if-then) y estudiar algunos ejemplos. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el if-then.

#### 4.2.1 Ejercicios resueltos para la selección simple (if-then)

A continuación se presentan ejercicios resueltos; se le recomienda que primero haga Usted el algoritmo, y después compare su solución con la del libro.

##### Ejercicio 4.2.1.1

Elaborar un algoritmo que lea tres números y que imprima el mayor. Se supone que son números diferentes. Este problema es la tercera ocasión que lo solucionamos.

**Tercer método de solución:** Utilizando if-then y AND.

(Primero hágalo usted, después compare la solución.)

```
Algoritmo MAYOR 3 NÚMEROS
1. Declarar
   Variables
      a, b, c: Entero
2. Solicitar número 1, número 2, número 3
3. Leer a, b, c
4. if (a > b)AND(a > c) then
   a. Imprimir a
5. endif
6. if (b > a)AND(b > c) then
   a. Imprimir b
7. endif
8. if (c > a)AND(c > b) then
   a. Imprimir c
9. endif
10. Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C414.C y Programa en Java: Mayor3Numeros3.java

*Explicación:*

1. Se declaran las variables
2. Solicitar los tres números
3. Se leen los datos en a, b y c
4. Se compara si (a > b) y (a > c), si se cumple, entonces
  - a. Se imprime a como el mayor
5. Fin del if
6. Se compara si (b > a) y (b > c), si se cumple, entonces
  - a. Se imprime b como el mayor
7. Fin del if
8. Se compara si (c > a) y (c > b), si se cumple, entonces
  - a. Se imprime c como el mayor
9. Fin del if
10. Fin del algoritmo

**Ejercicio 4.2.1.2**

Elaborar un algoritmo que lea cinco números y que imprima el mayor. Se supone que son números diferentes. Restricciones: Usar if-then, no usar else ni AND.

*(Primero hágalo usted, después compare la solución.)*

```

Algoritmo MAYOR 5 NÚMEROS
1. Declarar
   Variables
      a, b, c, d, e, mayor: Entero
2. Solicitar número 1, número 2, número 3,
   número 4, número 5
3. Leer a, b, c, d, e
4. mayor = a
5. if b > mayor then
   a. mayor = b
6. endif
7. if c > mayor then
   a. mayor = c
8. endif
9. if d > mayor then
   a. mayor = d
10. endif
11. if e > mayor then
   a. mayor = e
  
```

```
12. endif  
13. Imprimir mayor  
14. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C415.C y Programa en Java: Mayor5Numeros1.java



*Explicación:*

1. Se declaran las variables
2. Se solicitan los cinco números
3. Se leen en a, b, c, d y e
4. Se coloca en mayor el valor de a
5. Se compara si  $b >$  mayor si se cumple, entonces
  - a. Se coloca en mayor el valor de b
6. Fin del if
7. Se compara si  $c >$  mayor, si se cumple, entonces
  - a. Se coloca en mayor el valor de c
8. Fin del if
9. Se compara si  $d >$  mayor, si se cumple, entonces
  - a. Se coloca en mayor el valor de d
10. Fin del if
11. Se compara si  $e >$  mayor, si se cumple, entonces
  - a. Se coloca en mayor el valor de e
12. Fin del if
13. Se imprime mayor que contiene el número mayor
14. Fin del algoritmo

La tabla 4.2 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 4 de la Web del libro.

**Tabla 4.2**

| Ejercicio         | Descripción                                                        |
|-------------------|--------------------------------------------------------------------|
| Ejercicio 4.2.1.3 | Lee el tamaño de un ángulo e imprime el tipo que es                |
| Ejercicio 4.2.1.4 | Lee el tamaño de dos ángulos e imprime si son iguales o diferentes |
| Ejercicio 4.2.1.5 | Realiza cálculos logarítmicos de un ángulo                         |
| Ejercicio 4.2.1.6 | Calcula equivalencias de pies a yardas y otras                     |
| Ejercicio 4.2.1.7 | Realiza cálculos con la segunda ley de Newton                      |

### 4.3 La selección múltiple (switch)

Esta estructura de selección permite controlar la ejecución de acciones cuando se tienen más de dos opciones alternativas de acción.

*Formato:*

```
switch selector
  1: Acción(es)
  2: Acción(es)
  3: Acción(es)
  4: Acción(es)
default
  Acción(es)
endswitch
```

*En donde:*

|            |                                                                                                                                                                                                     |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| switch     | Identifica la estructura de selección múltiple.                                                                                                                                                     |
| selector   | Es una variable de tipo Entero, carácter, o algún tipo de dato ordinal (que esté constituido por un conjunto ordenado y finito de valores), la que traerá un valor que indicará el caso a ejecutar. |
| 1,2,3,4    | Son las etiquetas que identifican cada caso de acuerdo con los valores que puede tomar selector.                                                                                                    |
| Acción(es) | Es una acción o conjunto de acciones en pseudocódigo.                                                                                                                                               |
| default    | Si selector no toma ninguno de los valores colocados, se va por esta opción de default.                                                                                                             |
| endswitch  | Indica el fin de la estructura switch.                                                                                                                                                              |

*Funcionamiento:*

Si selector es 1:

Se ejecuta(n) la(s) acción(es) del caso uno y luego se va hasta después del endswitch.

Si selector es 2:

Se ejecuta(n) la(s) acción(es) del caso dos y luego se va hasta después del endswitch.

Si selector es 3:

Se ejecuta(n) la(s) acción(es) del caso tres y luego se va hasta después del endswitch.

Si selector es 4:

Se ejecuta(n) la(s) acción(es) del caso cuatro y luego se va hasta después del endswitch.

Si no se cumple ninguno de los casos anteriores, se ejecuta(n) la(s) acción(es) de la opción default y luego se va después del endswitch.

**Nota:** El default es opcional; en caso de no estar presente, la estructura quedará así:

```
switch selector
    1: Acción(es)
    2: Acción(es)
    3: Acción(es)
    4: Acción(es)
endswitch
```

Si selector no toma ninguno de los posibles valores, simplemente se salta hasta después de endswitch, es decir, no hace nada.

Ejercicio:

Elaborar un algoritmo que lea el número de día (un valor entre 1 y 7); e imprima domingo si es 1, lunes si es 2,..., sábado si es 7.

```
Algoritmo DICE DÍA
1. Declarar
    Variables
        numDia: Entero
2. Solicitar Número de día
3. Leer numDia
4. switch numDia
    1: Imprimir "DOMINGO"
    2: Imprimir "LUNES"
    3: Imprimir "MARTES"
    4: Imprimir "MIERCOLES"
    5: Imprimir "JUEVES"
    6: Imprimir "VIERNES"
    7: Imprimir "SABADO"
5. default
    a. Imprimir "NO ESTA EN EL RANGO DE 1 A 7"
6. endswitch
7. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C421.C y Programa en Java: DiceDia1.java



*Explicación:*

1. Se declara la variable para leer el número de día
2. Se solicita el número de día

3. Se lee en numDia
4. Se plantea la estructura de selección switch
  - Si numDia es 1 se imprime “DOMINGO”
  - Si numDia es 2 se imprime “LUNES”
  - Si numDia es 3 se imprime “MARTES”
  - Si numDia es 4 se imprime “MIÉRCOLES”
  - Si numDia es 5 se imprime “JUEVES”
  - Si numDia es 6 se imprime “VIERNES”
  - Si numDia es 7 se imprime “SÁBADO”
5. Si no toma (default) ningún valor de los anteriores  
Imprime “NO ESTA EN EL RANGO DE 1 A 7”
6. Fin del switch
- 7 Fin del algoritmo

#### Otros aspectos que la estructura switch permite

- a) Colocar como etiquetas de casos a más de un valor, podremos hacerlo como sigue:

```

Algoritmo DICE PAR O IMPAR
1. Declarar
    Variables
        num: Entero
2. Solicitar Número
3. Leer num
4. switch num
    0,2,4,6,8: Imprimir num, "es par"
    1,3,5,7,9: Imprimir num, "es impar"
5. default
    a. Imprimir num, "No esta en el rango 0-9"
6. endswitch
7. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C422.C y Programa en Java: DiceParImpar.java

#### *Explicación:*

1. Se declara la variable para leer el número
2. Se solicita el número
3. Se lee el número
4. Se plantea la selección switch

- Si num es 0, 2, 4, 6 u 8 imprime “es par”  
 Si num es 1, 3, 5, 7 ó 9 imprime “es impar”
5. Si no (default) toma ningún valor de los anteriores  
 Imprime “No está en el rango 0-9”
  6. Fin del switch
  7. Fin del algoritmo
- b) O bien, se pueden utilizar tipos de datos carácter, podremos hacerlo como sigue:

```
Algoritmo DICE SI ES VOCAL O NO
1. Declarar
    Variables
    car: Carácter
2. Solicitar Carácter
3. Leer car
4. switch car
    'a','e','i','o','u':
        Imprimir car, "Es una vocal minúscula"
    'A','E','I','O','U':
        Imprimir car, "Es una vocal mayúscula"
5. default
    a. Imprimir car, "No es vocal"
6. endswitch
7. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C423.C y Programa en Java: DiceVocal.java



*Explicación:*

1. Se declara la variable para leer el carácter
2. Se solicita el carácter
3. Se lee el carácter
4. Se plantea la selección switch  
 Si car está entre ‘a’,‘e’,‘i’,‘o’,‘u’:  
     Imprime “Es una vocal minúscula”  
 Si car está entre ‘A’,‘E’,‘I’,‘O’,‘U’:  
     Imprime “Es una vocal mayúscula”
5. Si no (default)  
 a. Imprime “No es vocal”
6. Fin del switch
7. Fin del algoritmo

Práctica: En este momento se recomienda ir al siguiente punto de ejercicio resuelto para la selección múltiple (switch) y estudiarlo. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el switch.

### 4.3.1 Ejercicio resuelto para la selección múltiple (switch)

#### Ejercicio 4.3.1.1

Una empresa vende hojas de hielo seco, con las condiciones siguientes:

- Si el cliente es tipo 1 se le descuenta el 5%
- Si el cliente es tipo 2 se le descuenta el 8%
- Si el cliente es tipo 3 se le descuenta el 12%
- Si el cliente es tipo 4 se le descuenta el 15%

Cuando el cliente realiza una compra se generan los datos siguientes:

- Nombre del cliente
- Tipo de cliente (1,2,3,4)
- Cantidad de hojas compradas
- Precio por hoja

Elabore un algoritmo que lea estos datos, haga cálculos e imprima:

- Nombre del cliente
- Subtotal a pagar (Cantidad de hojas X Precio por hoja)
- Descuento (el porcentaje correspondiente del Subtotal a pagar)
- Neto a pagar (Subtotal – Descuento)

Utilizar switch.

*(Primero hágalo usted, después compare la solución.)*

```

Algoritmo CLIENTE HOJAS HIELO SECO
1. Declarar
    Variables
        nombreClie: Cadena
        tipoClie, cantidad: Entero
        precioUni, subTotal, descuento, netoPagar: Real
2. Solicitar Nombre, Tipo cliente, Cantidad,
    Precio unitario
3. Leer nombreClie, tipoClie, cantidad, precioUni
4. subTotal = cantidad * precioUni
5. switch tipoClie
    1: descuento = subTotal * 0.05
    2: descuento = subTotal * 0.08
    3: descuento = subTotal * 0.12
    4: descuento = subTotal * 0.15

```

6. endswitch
7. netoPagar = subtotal - descuento
8. Imprimir nombreClie, subtotal, descuento, netoPagar
9. Fin

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C424.C y Programa en Java: Cliente1.java



*Explicación:*

1. Se declaran las variables
2. Se solicitan los datos
3. Se leen en nombreClie, tipoClie, cantidad, precioUni
4. Se calcula el subtotal
5. Se plantea el switch con el selector tipoClie
  - Si tipoClie es 1 calcula descuento con el 5%
  - Si tipoClie es 2 calcula descuento con el 8%
  - Si tipoClie es 3 calcula descuento con el 12%
  - Si tipoClie es 4 calcula descuento con el 15%
6. Fin del switch
7. Se calcula el netoPagar
8. Imprime nombreClie, subtotal, descuento, netoPagar
9. Fin del algoritmo

Nota:

En caso que se permita que el cliente pueda ser de tipo diferente de 1 a 4, y sólo tienen descuento estos tipos, se agregaría el default colocándole cero a descuento, esa parte quedaría:

5. switch tipoClie
  - 1: descuento = subtotal \* 0.05
  - 2: descuento = subtotal \* 0.08
  - 3: descuento = subtotal \* 0.12
  - 4: descuento = subtotal \* 0.15
6. default
  - a. descuento = 0
7. endswitch

## 4.4 Ejercicios propuestos

1. Elaborar un algoritmo para calcular e imprimir el precio de un terreno del cual se tienen los siguientes datos: largo, ancho y precio por metro cuadrado. Si el terreno tiene más de 400 metros cuadrados se hace un descuento del 10%.

2. Igual al ejercicio anterior, sólo que si el terreno tiene más de 500 metros cuadrados el descuento es del 17% y si tiene más de 1 000 el descuento es del 25%.
3. Elabore un algoritmo para calcular e imprimir los valores de X y Y, teniendo como entrada el valor de X y calculando el de Y de acuerdo con las siguientes condiciones:  
Si  $X < 0$  entonces  $Y = 3X + 6$   
Si  $X \geq 0$  entonces  $Y = X^2 + 6$
4. Elaborar un algoritmo que imprima el costo de un pedido de un artículo del cual se tiene la descripción, la cantidad pedida y el precio unitario. Si la cantidad pedida excede de 50 unidades, se hace un descuento del 15%.
5. Un cliente ordena cierta cantidad de hojas de hielo seco, viguetas y armazones; las hojas de hielo seco tienen 20% de descuento y las viguetas 15%. Los datos que se tienen por cada tipo de artículo son: la cantidad pedida y el precio unitario. Además, si se paga de contado todo tiene un descuento del 7%. Elaborar un algoritmo que calcule e imprima el costo total de la orden, tanto para el pago de contado como para el caso de pago a crédito.
6. Igual que el ejercicio 4.3.1.1 del punto anterior, pero además: Si la cantidad de hojas de hielo seco solicitada es mayor que 50, se hace un descuento adicional del 5%; en caso de ser mayor que 100 el descuento adicional es del 10%.
7. Elaborar un algoritmo que lea los datos de un estudiante: Nombre y tres calificaciones parciales e imprimir el Nombre y la calificación final de acuerdo con lo siguiente: Para aprobar el curso, debe tener 70 o más en cada una de las tres calificaciones, la calificación final será el promedio. En caso de haber reprobado uno o más exámenes ordinarios, la calificación final será NA (NO ACREDITADO).
8. De acuerdo con la clase de sus ángulos los triángulos se clasifican en:
  - Rectángulo, tiene un ángulo recto (igual a  $90^\circ$ ).
  - Obtusángulo, tiene un ángulo obtuso (mayor a  $90^\circ$  pero menor a  $180^\circ$ ).
  - Agutángulo, los tres ángulos son agudos (menores a  $90^\circ$ ).

Elaborar un algoritmo que permita leer el tamaño de los tres ángulos (A,B,C) de un triángulo e imprima qué tipo es.

9. En un almacén de venta de trajes, si se compra uno se hace el 50% de descuento, si compra 2 el 55%, si compra 3 el 60% y si compra más de 3 el 65%. Elaborar un algoritmo que lea la cantidad de trajes y el precio unitario (todos tienen el mismo precio) e imprima el subtotal a pagar, el descuento y el total a pagar.
10. Dos triángulos son congruentes si tienen la misma forma y tamaño, es decir, sus ángulos y lados correspondientes son iguales. Elaborar un algoritmo que lea los tres ángulos y lados de dos triángulos e imprima si son congruentes.
11. Un trapecio es isósceles si sus dos ángulos de la base son iguales. Elaborar un algoritmo que lea los ángulos A y B de la base y que imprima si el trapecio es isósceles o no.

12. Elaborar un algoritmo que permita leer los datos de un empleado: Nombre, tipo de empleado, número de horas trabajadas y cuota que se le paga por hora; calcular e imprimir el Sueldo a pagar. Si el empleado es tipo 1 se le pagan las horas extras (más de 40 horas) a 1.5 de la cuota por hora, si es tipo 2 a 2, si es tipo 3 a 2.5 y si es tipo 4 a 3 veces la cuota por hora.
13. Se tiene un terreno A cuadrado que mide LADO metros por lado a un precio COSTOA por metro cuadrado, y se tiene un terreno B rectangular que mide BASE metros de base y ALTURA metros de altura a un COSTOB por metro cuadrado. Elaborar un algoritmo que lea los datos de los dos terrenos e imprima cuál es el más barato o si cuestan igual.
14. Elabore un algoritmo que lea el número de mes entre 1 y 12, y que imprima el nombre del mes correspondiente: si es 1 “Enero”, si es 2 “Febrero”,..., etcétera.
15. En el hotel Guamúchil se hace un descuento del 10% si el cliente se hospeda más de 5 días, del 15% si se hospeda más de 10 días y del 20% si se hospeda más de 15 días. Elaborar un algoritmo que lea el número de días y el precio diario de la habitación e imprima el subtotal a pagar, el descuento y el total a pagar.
16. Elaborar un algoritmo que permita hacer conversiones de temperaturas entre grados Fahrenheit, Celsius, Kelvin y Rankine. Primero debe preguntar qué tipo de grados quiere convertir. Por ejemplo; si se le indica que se desea convertir una temperatura en grados Fahrenheit, debe leer la cantidad de grados, y luego calcular e imprimir su equivalente en grados Celsius, Kelvin y Rankine, y así, debe hacer lo mismo para cada uno de los otros tipos. Para convertir a Celsius a la temperatura Fahrenheit se le resta 32 y se multiplica por 5/9. Para convertir a Kelvin, se le suma 273 a los grados Celsius. Para convertir a Rankine a los grados Fahrenheit se le suma 460.
17. Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes o en grados (debe preguntar en que lo va a leer), e imprima el seno hiperbólico, coseno hiperbólico y tangente hiperbólica. Si lo lee en grados, debe hacer la conversión a radianes. En el ejercicio 3.5.9 está cómo hacer los cálculos.
18. Elaborar un algoritmo que permita leer los datos de un aspirante a ingresar a la carrera de Ingeniería Industrial y de Sistemas de la Universidad de Sonora: Nombre del aspirante, promedio en bachillerato y tipo de bachillerato (1– Físico-matemático, 2– etc.,..., 5); y que imprima si es aceptado si tiene un promedio mayor a 90, o bien, si tiene un promedio entre 80 y 90 y trae bachillerato Físico-matemático. En caso de no ser así imprimir rechazado.
19. Elaborar un algoritmo que permita leer los datos de un automóvil: marca, origen y costo; imprimir el impuesto a pagar y el precio de venta incluido el impuesto. Si el origen es Alemania el impuesto es 20%, si es de Japón el impuesto es 30%, si es de Italia el 15% y si es de EUA el 8%.

20. Un sistema de ecuaciones lineales:

$$ax + by = c$$

$$dx + ey = f$$

se puede resolver con las fórmulas:

$$X = \frac{ce - bf}{ae - bd} \quad Y = \frac{af - cd}{ae - bd} \quad \text{si } (ae - bd) \neq 0.$$

Elaborar un algoritmo que lea los coeficientes a,b,c,d,e y f, y que calcule e imprima los valores de X y Y. Si  $(ae - bd) \neq 0$ ; debe calcular e imprimir los valores de X y Y, en caso contrario debe imprimir un mensaje que indique que no tiene solución.

21. Elaborar un algoritmo que permita leer el Nombre, tipo de empleado y Sueldo de un empleado; que imprima el incremento de Sueldo y su nuevo Sueldo de acuerdo con lo siguiente: si es tipo de empleado 1 se le aumentará el 5%, si es tipo 2 se le aumentará el 7%, si es 3 el 9%, si es 4 el 12% y si es 5 el 15%.
22. Elaborar un algoritmo que permita leer una letra e imprima si es vocal o si es consonante.
23. Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes o en grados y que imprima la tangente, secante, cotangente y cosecante. Debe preguntar en qué tiene el tamaño del ángulo, y dependiendo de si es en grados o radianes, los cálculos deben hacerse de una u otra forma según corresponda.
24. Elaborar un algoritmo que permita leer 4 números e imprima el mayor. Debe validar que sean diferentes, es decir, si hay números iguales debe enviar un mensaje de error.
25. Elaborar un algoritmo que permita hacer conversiones entre pesos, yenes, pesetas y marcos. Debe preguntar qué moneda desea convertir; por ejemplo, si indica que yenes, debe leer cuántos yenes comprará y cuánto cuesta un yen en pesos, pesetas y marcos, luego debe imprimir cuánto es en cada una de las monedas, y así lo hará para cada una de las otras monedas.
26. Una temperatura en grados Celsius (C) se puede convertir a su equivalente Fahrenheit (F) con la fórmula:

$$F = \frac{9}{5} C + 32$$

$$\text{De Fahrenheit a Celsius con la fórmula: } C = (F - 32) \frac{5}{9}$$

Elaborar un algoritmo que pregunte qué quiere convertir, si quiere convertir Celsius, que lea la temperatura en grados Celsius y calcule e imprima la temperatura Fahrenheit equivalente. Si quiere convertir Fahrenheit debe hacer lo propio.

## 4.5 Resumen de conceptos que debe dominar

- La selección doble (if-then-else)
- Expresiones lógicas
  - simples ( $>$ ,  $<$ ,  $\geq$ ,  $\leq$ ,  $\neq$ ,  $\equiv$ )
  - complejas (AND, OR, XOR, NOT)
- if's anidados
- La selección simple (if-then)
- La selección múltiple (switch)

## 4.6 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

- 4.6.1 Resumen gráfico del capítulo
- 4.6.2 Autoevaluación
- 4.6.3 Programas
- 4.6.4 Ejercicios resueltos
- 4.6.5 Power Point para el profesor (\*)

# 5

## La repetición

### Contenido

- 5.1 La repetición do...while
  - 5.1.1 Contadores y acumuladores
  - 5.1.2 Ejercicios resueltos para la repetición do...while
- 5.2 Ejercicios propuestos para la repetición do...while
- 5.3 La repetición for
  - 5.3.1 For anidados
  - 5.3.2 Ejercicios resueltos para la repetición for
    - 5.3.3 Simulación del for con do...while
- 5.4 Ejercicios propuestos para la repetición for
- 5.5 La repetición while
  - 5.5.1 Simulación del do...while con while
  - 5.5.2 Simulación del for con while
  - 5.5.3 Ejercicios resueltos para la repetición while
- 5.6 Ejercicios propuestos para la repetición while
- 5.7 Resumen de conceptos que debe dominar
- 5.8 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 5.8.1 Resumen gráfico del capítulo
  - 5.8.2 Autoevaluación
  - 5.8.3 Programas
  - 5.8.4 Ejercicios resueltos
  - 5.8.5 Power Point para el profesor (\*)

### Objetivos del capítulo

- Estudiar la estructura de control de repetición do...while.
- Estudiar la estructura de control de repetición for.
- Estudiar la estructura de control de repetición while.
- Aprender la forma de emitir la información en forma de reporte.
- Aprender la manera de cómo utilizar contadores y acumuladores, así como cómo obtener promedios o medias aritméticas.
- Aprender a obtener el mayor y el menor de un conjunto de datos y a utilizar un ciclo repetitivo anidado dentro de otro, es decir, un do...while dentro de otro do...while; utilizando todas las estructuras aprendidas hasta este momento, además de los conceptos y estructuras nuevas que se presentan en este capítulo.
- Aprender a plantear un for dentro de otro for, un do...while dentro de un for y un for dentro de un do...while.
- Estudiar cómo simular la estructura for con do...while.
- Aprender a anidar un while dentro de otro while, se combinará con el uso de las estructuras estudiadas hasta ahora, como por ejemplo anidar while con for y do...while.
- Aprender la forma de plantear ciclos for y do...while con while.

## Introducción

Con el estudio del capítulo anterior, usted ya domina la selección y cómo diseñar algoritmos usando esa estructura de control.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando la estructura de control: la repetición.

Se explica que la repetición es una estructura que permite plantear situaciones en las que algo se debe ejecutar repetidamente y que tiene tres formas: la repetición do...while, la repetición for,y la repetición while.

Se enseña que la repetición do...while sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición; en un rango de 1 a n veces.

Se expone que la repetición for sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán un número de veces conocido de antemano.

Se explica que la repetición while sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición; en un rango de cero a n veces.

También se trata la forma de emitir la información en forma de reporte y se detalla la manera de cómo utilizar contadores y acumuladores,cómo obtener promedios o medias aritméticas al procesar varios elementos, a obtener el mayor y el menor de un conjunto de datos y a utilizar un ciclo repetitivo anidado dentro de otro, es decir, un do...while dentro de otro do...while; lógicamente utilizando todas las estructuras aprendidas hasta este momento, ademásde los conceptos y estructuras nuevas que se presentan en este capítulo.

Se enseña a plantear un for dentro de otro for, un do...while dentro de un for y un for dentro de un do...while. Cómo simular la estructura for con do...while.

Asimismo se describe cómo anidar un while dentro de otro while, se combina con el uso de las estructuras estudiadas hasta ahora, como por ejemplo anidar while con for y do...while. Además se trata la forma de plantear ciclos for y do...while con while.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejerzte estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luegoverifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que, a veces, aunque haya diferencias, las dos soluciones son correctas.

En el siguiente capítulo se estudia la estructura de datos denominada arreglos, qué es, cómo se representa y su uso en seudocódigo.

## 5.1 La repetición do...while

La repetición do...while permite controlar la ejecución de acciones en forma repetitiva, mientras la condición de control del ciclo repetitivo sea verdadera.

*Formato:*

```
do
    Acción(es)
    while condición
```

*En donde:*

|            |                                                                                                                                                        |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| do         | Identifica la estructura como un ciclo repetitivo e indica el inicio del mismo.                                                                        |
| Acción(es) | Son las acciones que se ejecutan dentro del ciclo.                                                                                                     |
| while      | Indica el fin del ciclo y significa “mientras” se cumpla la condición, vuelve al inicio del ciclo do; en caso contrario, se sale del ciclo do...while. |
| condición  | Es una expresión lógica que controla la repetición del ciclo.                                                                                          |



Este tipo de repetición se ejecuta de 1-N veces, ya que las acciones del ciclo se ejecutan por lo menos una vez y se pueden ejecutar cualquier cantidad (N) de veces.

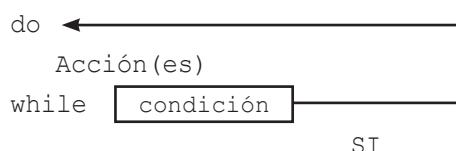
*Funcionamiento:*

1. Llega al do; entra al ciclo y ejecuta(n) la(s) acción(es).

```
do
    ↓ Acción(es)
    while condición
```

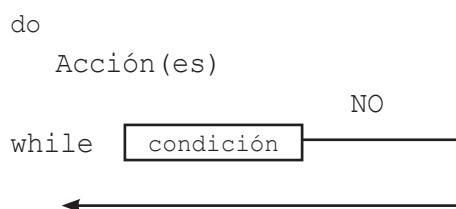
2. Llega al while, y se evalúa la condición.

- a) Si se cumple, se va hacia el inicio de la estructura do, lo que significa volver a ejecutar lo que está dentro del do.



Cabe aclarar que esta estructura originalmente se inventó como do-until; hacer lo que está dentro del ciclo, hasta que se cumpla la condición. Sin embargo, el inventor del lenguaje C la implementó como do...while, es decir, hacer lo que está dentro del ciclo, mientras se cumpla la condición, y en virtud de que el lenguaje C es la base de C++, Java y seguramente de los lenguajes que se diseñen en el futuro, es que se ha convertido en el estándar, por ello, en este libro la usaremos de esta forma.

- b) Si no se cumple, se va a la siguiente acción después del while, es decir, se sale del ciclo repetitivo.



**Ejemplo:**

Elaborar un algoritmo que calcule e imprima el sueldo de varios empleados, cada empleado se tratará en forma similar al primer problema que planteamos en el capítulo 3 cuando estudiamos la secuenciación.

A continuación se tiene el algoritmo de la solución:

```

Algoritmo CALCULA SUELDOS DE EMPLEADOS
1. Declarar
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
        desea: Carácter
2. do
    a. Solicitar Nombre, Número de horas trabajadas
       y Cuota por hora
    b. Leer nombreEmp, horasTrab, cuotaHora
    c. Calcular sueldo = horasTrab * cuotaHora
    d. Imprimir nombreEmp, sueldo
    e. Preguntar "¿Desea procesar otro empleado (S/N) ?"
    f. Leer desea
3. while desea == 'S'
4. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C501.C y Programa en Java: Empleados1.java

**Explicación:**

1. Se declaran las variables que ya conocemos nombreEmp, horasTrab, cuotaHora y sueldo, además, desea es una variable de tipo carácter que servirá para controlar al ciclo repetitivo.
2. Se inicia el planteamiento del ciclo repetitivo do
  - a. Se solicitan el nombre, número de horas trabajadas y cuota por hora
  - b. Se leen en nombreEmp, horasTrab, cuotaHora
  - c. Se calcula el sueldo
  - d. Imprime nombreEmp, sueldo
  - e. Se pregunta si "¿Desea procesar otro empleado (S/N)?", pregunta a la cual se debe contestar S para Sí o una N para NO
  - f. Se lee en desea, la respuesta que se dé a la pregunta anterior
3. Delimita el fin del ciclo repetitivo, si se cumple la condición desea == 'S', el control se transfiere hacia el do, y entra de nuevo al ciclo repetitivo. En

caso de no cumplirse, el control se transfiere hacia la siguiente acción después del while, es decir, se sale del ciclo do. 'S' se pone entre apóstrofos porque es un valor (constante) de tipo carácter

#### 4. Fin del algoritmo

##### 5.1.1 Contadores y acumuladores

En ocasiones, en los ciclos repetitivos se necesita contar el número de veces que se procesa algún elemento, o bien, hacer acumulaciones de cantidades para obtener totalizadores. Veamos el siguiente ejemplo:

En el problema anterior de sueldos de empleados, es necesario que los datos se impriman en forma de un reporte que tenga el siguiente formato:

| REPORTE DE EMPLEADOS   |            |
|------------------------|------------|
| NOMBRE                 | SUELDO     |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| -                      | -          |
| -                      | -          |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| TOTAL 999 EMPLEADOS    | 999,999.99 |

}      Encabezado  
}      Detalle  
}      Total o pie

Como puede observar, el reporte tiene 3 partes:

- a) La primera parte es el *encabezado*, que deberá aparecer una sola vez al principio; la primera línea del encabezado es el nombre del reporte, que en este caso es un REPORTE DE EMPLEADOS, la segunda línea son letreros que identifican los datos que aparecerán en el reporte, que en este caso son el NOMBRE y el SUELDO de los empleados.
- b) La segunda parte es el *detalle* de cada empleado que aparecerá tantas veces como empleados haya; el conjunto de XXXXXXXXXXXXXXXXXXXXXXX indica que va un dato cadena de caracteres, el nombre del empleado; y 99,999.99 indica que va un dato de tipo real, el sueldo del empleado.
- c) La tercera parte es una línea de *total o pie del reporte* que aparecerá una sola vez al final y tiene como propósito mostrar una estadística o resumen de los datos que aparecen en el detalle del reporte:

TOTAL 999 EMPLEADOS 999,999.99. Los 999 indican que va un dato de tipo entero –el total de empleados–, es decir, el número de empleados que están en el reporte; y los 999,999.99, indica que va un dato de tipo real, es un totalizador o acumulador con la suma de los sueldos de todos los empleados. Para el primero se requiere el uso de un contador y para el segundo un acumulador.

**¿Qué es un contador?** Una variable de tipo entero que en nuestro ejemplo podría llamarse totEmpleados que tiene como función contener el número de empleados que se procesan. El contador funciona de la forma siguiente: Al principio



La palabra varios, significa que se tiene más de un empleado y no se sabe cuántos son. Es por ello que en el paso e se pregunta si desea procesar otro empleado, en el paso f se lee la respuesta en la variable desea y en el paso 3, que cierra el ciclo, se plantea la condición desea == 'S'. De esta forma se controla que se puedan procesar varios empleados (o lo que corresponda), aun cuando no se conozca cuántos son. En este capítulo estaremos planteando problemas que tienen esta naturaleza de repetición. Posteriormente plantearemos problemas en los que se conoce de antemano cuántos elementos se procesarán.

Observe que las variables que se utilizan para manejar los datos del primer empleado, son las mismas variables que se utilizan para manejar los datos del segundo empleado, y así para todos los empleados que se procesen.

pio se inicia con cero; dentro del ciclo se incrementa en 1, para así contar a cada empleado procesado. Al final, después del fin del ciclo, se podrá imprimir el contenido del contador, que será el total de empleados procesados. A continuación se muestra el funcionamiento del contador:

```
totEmpleados = 0
do
    Procesar empleado
    --
    totEmpleados = totEmpleados + 1
    --
    --
while desea == 'S'
Imprimir totEmpleados
```

**¿Qué es un acumulador?** Es una variable de tipo numérico, que en nuestro ejemplo puede ser totSueldos, cuya función es contener la suma de un determinado conjunto de datos, que en el ejemplo está representado por los sueldos. La forma de operar del acumulador es la siguiente: Al principio se inicia con cero, dentro del ciclo se incrementa con lo que tenga la variable que contiene el dato a acumular, que en este caso es sueldo; al final, después del fin del ciclo, se podrá imprimir el contenido del acumulador, que es el total de la suma, en este caso de sueldos. A continuación se muestra el funcionamiento del acumulador:

```
totSueldos = 0
do
    Procesar empleado
    --
    totSueldos = totSueldos + sueldo
    --
    --
    --
while desea == 'S'
Imprimir totSueldos
```

A continuación se presenta el algoritmo completo:

```
Algoritmo CALCULA SUELdos DE EMPLEADOS
1. Declarar
    Variables
        nombreEmp: Cadena
        horasTrab, totEmpleados: Entero
        cuotaHora, sueldo, totSueldos: Real
        desea: Carácter
2. Imprimir encabezado
```

```
3. totEmpleados = 0
4. totSueldos = 0
5. do
    a. Solicitar Nombre, Número de horas trabajadas y
        Cuota por hora
    b. Leer nombreEmp, horasTrab, cuotaHora
    c. Calcular sueldo = horasTrab * cuotaHora
    d. Imprimir nombreEmp, sueldo
    e. totEmpleados = totEmpleados + 1
    f. totSueldos = totSueldos + sueldo
    g. Preguntar "¿Desea procesar otro empleado (S/N) ?"
    h. Leer desea
6. while desea == 'S'
7. Imprimir totEmpleados, totSueldos
8. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C502.C y Programa en Java: Empleados2.java



*Explicación:*

1. Se declaran las variables que ya conocemos nombreEmp, horasTrab, cuotaHora, sueldo y desea; además se declaran totEmpleados, que servirá para contar el total de empleados procesados y totSueldos, que se utiliza para acumular la suma de los sueldos de todos los empleados
2. Se indica que se imprime el encabezado del reporte de acuerdo con el formato del mismo. Otra forma de imprimirla sería así:  
Imprimir "REPORTE DE EMPLEADOS"  
Imprimir "NOMBRE SUELDO"  
Imprimir "\_\_\_\_\_"
3. Se inicia en cero el contador de empleados
4. Se inicia en cero el acumulador de sueldos
5. Se inicia el planteamiento del ciclo repetitivo do
  - a. Se solicitan el nombre, número de horas trabajadas y cuota por hora
  - b. Se leen en nombreEmp, horasTrab, cuotaHora
  - c. Se calcula el sueldo
  - d. Imprime nombreEmp, sueldo
  - e. Se incrementa el contador de empleados en 1
  - f. Se incrementa el acumulador de sueldos con sueldo
  - g. Se pregunta si "¿Desea procesar otro empleado (S/N)?", pregunta a la cual se debe contestar S para Sí o una N para NO
  - h. Se lee en desea, la respuesta que se dé a la pregunta anterior

6. Delimita el fin del ciclo repetitivo, si se cumple la condición desea == 'S', el control se transfiere hacia el do, y entra de nuevo al ciclo repetitivo. En caso de no cumplirse, el control se transfiere hacia la siguiente acción después del while, es decir, se sale del ciclo
7. Se imprimen el total de empleados y el total de sueldos
8. Fin del algoritmo

### Funciones Break y Continue

La mayoría de los lenguajes proporcionan las funciones Break y Continue.

#### Break

La función Break interrumpe (termina) la ejecución de un ciclo repetitivo y transfiere el control a la siguiente acción después del ciclo.

Formato:

```
Break
```

Ejemplo:

```
do  
    Break  
    .  
    while condición
```

#### Continue

La función Continue permite enviar el control al inicio del ciclo repetitivo, es decir, transfiere el control para ejecutar la siguiente repetición del ciclo.

Formato:

```
Continue
```

Ejemplo:

```
do  
    Continue  
    .  
    while condición
```

Break y Continue son aplicables en ciclos do...while, for y while. Si no están dentro de un ciclo habrá error.

En este libro no se le dará aplicación a esas funciones, porque si un programa está bien estructurado, no son necesarias, y este libro se enfoca precisamente al diseño de programas bien estructurados.

#### 5.1.2 Ejercicios resueltos para la repetición do...while

A continuación se presentan ejercicios resueltos; se le recomienda que primero haga Usted el algoritmo, y después compare su solución con la del libro.

Cabe aclarar que por la naturaleza de la repetición do...while, los problemas que se plantean en este capítulo tienen una orientación más administrativa, porque esta estructura es más natural para resolver problemas de esta índole. Sin embargo, en el punto siguiente (for), donde los problemas tienen una orientación más hacia ingeniería, utilizaremos el do...while para resolver ese tipo de problemas.

### Ejercicio 5.1.2.1

Una empresa vende hojas de hielo seco, con las condiciones siguientes:

- Si el cliente es tipo 1 se le descuenta el 5%
- Si el cliente es tipo 2 se le descuenta el 8%
- Si el cliente es tipo 3 se le descuenta el 12%
- Si el cliente es tipo 4 se le descuenta el 15%

Cuando un cliente realiza una compra se generan los datos siguientes:

- Nombre del cliente
- Tipo de cliente (1,2,3,4)
- Cantidad de hojas compradas
- Precio por hoja

Elaborar un algoritmo que permita procesar varios clientes e imprima el reporte:

#### REPORTE DE CLIENTES

| NOMBRE               | SUBTOTAL  | DESCUENTO | NETO A PAGAR |
|----------------------|-----------|-----------|--------------|
| XXXXXXXXXXXXXXXXXXXX | 99,999.99 | 99,999.99 | 99,999.99    |
| XXXXXXXXXXXXXXXXXXXX | 99,999.99 | 99,999.99 | 99,999.99    |
| -                    | -         | -         | -            |
| -                    | -         | -         | -            |
| XXXXXXXXXXXXXXXXXXXX | 99,999.99 | 99,999.99 | 99,999.99    |
| TOTAL 999 CLIENTES   | 99,999.99 | 99,999.99 | 99,999.99    |

Cálculos:

Subtotal = Cantidad de hojas X Precio por hoja

Descuento es el porcentaje correspondiente del Subtotal a pagar

Neto a pagar = Sub total – Descuento

La sumatoria de cada uno de estos datos para imprimirlos como totales.

(Primero hágalo usted, después compare la solución.)

Algoritmo CLIENTES HOJAS HIELO SECO

1. Declarar

Variables

nombreClie: Cadena

```

        tipoClie, cantidad, totClientes: Entero
        precioUni, subTotal, descuento, netoPagar,
        totSubTot, totDescuento, totNeto: Real
        desea: Carácter
    2. Imprimir encabezado
    3. totClientes = 0
        totSubTot = 0
        totDescuento = 0
        totNeto = 0
    4. do
        a. Solicitar Nombre, Tipo cliente, Cantidad,
            Precio unitario
        b. Leer nombreClie, tipoClie, cantidad, precioUni
        c. subTotal = cantidad * precioUni
        d. switch tipoClie
            1: descuento = subTotal * 0.05
            2: descuento = subTotal * 0.08
            3: descuento = subTotal * 0.12
            4: descuento = subTotal * 0.15
        e. endswitch
        f. netoPagar = subTotal - descuento
        g. Imprimir nombreClie, subTotal, descuento,
            netoPagar
        h. totClientes = totClientes + 1
            totSubTot = totSubTot + subTotal
            totDescuento = totDescuento + descuento
            totNeto = totNeto + netoPagar
        i. Preguntar "¿Desea procesar otro cliente (S/N)?""
        j. Leer desea
    5. while desea == 'S'
    6. Imprimir totClientes, totSubTot, totDescuento,
        totNeto
    7. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C503.C y Programa en Java: Clientes1.java

*Explicación:*

1. Se declaran las variables
2. Se imprime el encabezado
3. Se inician en cero los totales generales

4. Se inicia el planteamiento del ciclo repetitivo do
  - a. Se solicitan los datos
  - b. Se leen en nombreClie, tipoClie, cantidad, precioUni
  - c. Se calcula el subTotal
  - d. Se plantea el switch con el selector tipoClie
 

Si tipoClie es 1 calcula descuento con el 5%

Si tipoClie es 2 calcula descuento con el 8%

Si tipoClie es 3 calcula descuento con el 12%

Si tipoClie es 4 calcula descuento con el 15%
  - e. Fin del switch
  - f. Se calcula el netoPagar
  - g. Imprime nombreClie, subTotal, descuento, netoPagar
  - h. Se incrementan los totales
  - i. Se pregunta “¿Desea procesar otro cliente (S/N)?”
  - j. Se lee la respuesta en desea
5. Delimita el fin del ciclo repetitivo, si se cumple la condición desea == 'S', el control se transfiere hacia el do, y entra de nuevo al ciclo repetitivo. En caso de no cumplirse el control se transfiere hacia la siguiente acción después del while, es decir, se sale del ciclo.
6. Imprimie los totales totClientes, totSubTot, totDescuento, totNeto
7. Fin del algoritmo

### Ejercicio 5.1.2.2

Elaborar un algoritmo que proporcione el siguiente reporte:

ANÁLISIS DE CALIFICACIONES

| NOMBRE               | CAL.1 | CAL.2 | CAL.3 | CAL.4 | PROMEDIO |
|----------------------|-------|-------|-------|-------|----------|
| XXXXXXXXXXXXXXXXXXXX | 99.99 | 99.99 | 99.99 | 99.99 | 99.99    |
| XXXXXXXXXXXXXXXXXXXX | 99.99 | 99.99 | 99.99 | 99.99 | 99.99    |
| -                    | -     | -     | -     | -     | -        |
| -                    | -     | -     | -     | -     | -        |
| XXXXXXXXXXXXXXXXXXXX | 99.99 | 99.99 | 99.99 | 99.99 | 99.99    |
| PROMEDIOS GENERALES  | 99.99 | 99.99 | 99.99 | 99.99 | 99.99    |

A partir de que se tiene el NOMBRE, calificación 1,2,3 y 4 de varios alumnos. El promedio se obtiene sumando las cuatro calificaciones y dividiendo el resultado entre cuatro.

El promedio general de cada calificación se calcula sumando las calificaciones de todos los alumnos y dividiendo el resultado entre el número de alumnos.

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo CALIFICACIONES DE ALUMNOS
1. Declarar
    Variables
        nombreAlum: Cadena
        totAlumnos: Entero
        calif1, calif2, calif3, calif4, promedio, promCall,
        promCal2, promCal3, promCal4, promProm, totCall,
        totCal2, totCal3, totCal4, totProm: Real
        desea: Carácter
2. Imprimir encabezado
3. totAlumnos = 0
    totCall = 0; totCal2 = 0; totCal3 = 0;
    totCal4 = 0; totProm = 0;
4. do
    a. Solicitar Nombre, calificación 1, calificación 2,
       calificación 3 y calificación 4
    b. Leer nombreAlum, calif1, calif2, calif3, calif4
    c. promedio = (calif1 + calif2 + calif3 + calif4)/4
    d. Imprimir nombreAlum, calif1, calif2, calif3,
       calif4, promedio
    e. totAlumnos = totAlumnos + 1
    f. totCall = totCall + calif1
        totCal2 = totCal2 + calif2
        totCal3 = totCal3 + calif3
        totCal4 = totCal4 + calif4
        totProm = totProm + promedio
    g. Preguntar "¿Desea procesar otro alumno (S/N)?""
    h. Leer desea
5. while desea == 'S'
6. promCall = totCall / totAlumnos
    promCal2 = totCal2 / totAlumnos
    promCal3 = totCal3 / totAlumnos
    promCal4 = totCal4 / totAlumnos
    promProm = totProm / totAlumnos
7. Imprimir promCall, promCal2, promCal3, promCal4,
   promProm
8. Fin
```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C504.C y Programa en Java: Alumnos1.java

**Explicación:**

1. Se declaran las variables
2. Imprime encabezado
3. Inicia en cero los acumuladores de calificaciones; y el contador de alumnos
4. Inicia el ciclo do
  - a. Se solicitan Nombre, calificación 1,2,3 y 4
  - b. Se leen en nombreAlum, calif1, calif2, calif3, calif4
  - c. Se calcula el promedio
  - d. Se imprimen nombreAlum, calif1, calif2, calif3, calif4, promedio
  - e. Se incrementa el totAlumnos en 1
  - f. Se incrementa
   
totCal1 con calif1
   
totCal2 con calif2
   
totCal3 con calif3
   
totCal4 con calif4
   
totProm con promedio
  - g. Se pregunta “¿Desea procesar otro alumno (S/N)?”
  - h. Se lee en desea la respuesta
5. Fin ciclo while desea == 'S'; Si se cumple, vuelve al do; si no, sale del ciclo.
6. Se calcula el promCal1  
Se calcula el promCal2  
Se calcula el promCal3  
Se calcula el promCal4  
Se calcula el promProm  
  
Nota: Observe que los promedios se calculan afuera del ciclo.
7. Imprime promCal1, promCal2, promCal3, promCal4, promProm
8. Fin del algoritmo

La tabla 5.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 5 de la Web del libro.

**Tabla 5.1**

| Ejercicio         | Descripción                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------|
| Ejercicio 5.1.2.3 | Procesa las calificaciones de varios alumnos                                                      |
| Ejercicio 5.1.2.4 | Procesa artículos con inflación                                                                   |
| Ejercicio 5.1.2.5 | Procesa la producción de varios obreros con dos ciclos                                            |
| Ejercicio 5.1.2.6 | Procesa la producción de varios obreros ampliada                                                  |
| Ejercicio 5.1.2.7 | Procesa las compras de varios clientes                                                            |
| Ejercicio 5.1.2.8 | Procesa las ventas de varios vendedores; y también se plantea para cuando se tienen 15 vendedores |



### Resumiendo

La repetición do...while es útil para resolver problemas del tipo:

- Algo se repite varias veces, es decir, se debe ejecutar en más de una ocasión, pero no se sabe cuántas veces se repetirá. Se controla haciendo la pregunta y leyendo en una variable la respuesta (desea, otro), para plantear la condición de control del ciclo. Es el caso de todos los problemas planteados en este punto, con excepción del último.
- Algo se repite un número conocido de veces, es decir, se sabe cuántas veces se repetirá. Se controla utilizando un contador que va contando la cantidad de veces que entra al ciclo, el cual se utiliza para plantear la condición de control del ciclo. Es el caso del último algoritmo de este punto. Este tipo de problemas son los que se plantean en el siguiente punto, ya que son naturales para la repetición for, no obstante que también pueden resolverse utilizando do...while.

## 5.2 Ejercicios propuestos para la repetición do...while

- Elaborar un algoritmo que imprima el siguiente reporte:

| ARTÍCULO            | COSTOS DE PRODUCCIÓN   |                 |               |                        |
|---------------------|------------------------|-----------------|---------------|------------------------|
|                     | UNIDADES<br>PRODUCIDAS | FACTOR<br>COSTO | COSTO<br>FIJO | COSTO DE<br>PRODUCCIÓN |
| XXXXXXXXXXXXXXXXXX  | 9999                   | 999.99          | 99,999.99     | 999,999.99             |
| XXXXXXXXXXXXXXXXXX  | 9999                   | 999.99          | 99,999.99     | 999,999.99             |
| -                   | -                      | -               | -             | -                      |
| -                   | -                      | -               | -             | -                      |
| -                   | -                      | -               | -             | -                      |
| -                   | -                      | -               | -             | -                      |
| -                   | -                      | -               | -             | -                      |
| -                   | -                      | -               | -             | -                      |
| -                   | -                      | -               | -             | -                      |
| XXXXXXXXXXXXXXXXXX  | 9999                   | 999.99          | 99,999.99     | 999,999.99             |
| TOTAL 999 ARTÍCULOS |                        |                 |               | 99,999,999.99          |

Datos disponibles por cada artículo:

- Descripción
- Cantidad de unidades producidas
- Factor de costo de materiales
- Costo fijo

Cálculo del costo de producción =

$$\text{Número de unidades producidas} \times \text{Factor de costo de materiales} \\ + \text{Costo fijo}$$

Al final imprimir el total de artículos procesados, y el total general del costo de producción.

2. Elaborar un algoritmo que imprima el siguiente reporte:

| ARTÍCULO   | PRECIOS DE VENTA    |            |            | PRECIO DE VENTA |
|------------|---------------------|------------|------------|-----------------|
|            | COSTO DE PRODUCCIÓN | UTILIDAD   | IMUESTO    |                 |
| XXXXXXXXXX | 99,999.99           | 99,999.99  | 99,999.99  | 999,999.99      |
| XXXXXXXXXX | 99,999.99           | 99,999.99  | 99,999.99  | 999,999.99      |
| -          | -                   | -          | -          | -               |
| XXXXXXXXXX | 99,999.99           | 99,999.99  | 99,999.99  | 999,999.99      |
| XXXXXXXXXX | 99,999.99           | 99,999.99  | 99,999.99  | 999,999.99      |
| TOTAL 999  | 999,999.99          | 999,999.99 | 999,999.99 | 9, 999,999.99   |

Datos disponibles por cada artículo:

- Descripción
- Costo de producción
- Cálculos:
  - Utilidad = 120% del costo de producción
  - Impuesto = 15% (costo de producción + utilidad)
  - Precio de venta = costo de producción + utilidad + impuesto

3. Elaborar un algoritmo para imprimir una factura que contenga los datos de los artículos vendidos a un cliente, y que tenga el siguiente formato:

| FACTURA                                         |              |                 |              |
|-------------------------------------------------|--------------|-----------------|--------------|
| NOMBRE DEL CLIENTE : XXXXXXXXXXXXXXXXXXXXXXXXXX |              |                 |              |
| ARTÍCULO                                        | CANTIDAD     | PRECIO UNITARIO | PRECIO TOTAL |
| XXXXXXXXXXXXXXXXXX                              | 999          | 99,999.99       | 999,999.99   |
| XXXXXXXXXXXXXXXXXX                              | 999          | 99,999.99       | 999,999.99   |
| -                                               | -            | -               | -            |
| XXXXXXXXXXXXXXXXXX                              | 999          | 99,999.99       | 999,999.99   |
|                                                 | SUBTOTAL     | 9,999,999.99    |              |
|                                                 | IMPUESTO 15% | 9,999,999.99    |              |
|                                                 | TOTAL        | 9,999,999.99    |              |

Datos disponibles por cada artículo:

- Descripción
- Cantidad de artículos
- Precio unitario de venta
- Además se tiene el nombre del cliente

Cálculos:

- Precio total = cantidad de artículos X precio unitario
- Subtotal = la suma de los precios totales
- Impuesto = 15% del subtotal
- Total = subtotal + impuesto

4. Igual al caso anterior, pero que se puedan procesar tantas facturas como clientes se hayan atendido.
5. Elaborar un algoritmo que imprima el siguiente reporte:

| NÓMINA QUINCENAL   |            |            |            |
|--------------------|------------|------------|------------|
| NOMBRE             | SDO. BRUTO | IMPUESTO   | SDO. NETO  |
| XXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99  |
| XXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99  |
| -                  | -          | -          | -          |
| -                  | -          | -          | -          |
| XXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99  |
| TOTAL 999          | 999,999.99 | 999,999.99 | 999,999.99 |

Datos disponibles por cada empleado:

- Nombre
- Sueldo mensual
- Antigüedad

Cálculos:

- Sueldo bruto = (sueldo mensual / 2) + prima de antigüedad.

La prima de antigüedad se otorga a partir del tercer año de labores y es de 2% anual.

El impuesto se calcula usando la tabla siguiente:

| Lím. inferior | Lím. superior | Cuota fija | Porcentaje |
|---------------|---------------|------------|------------|
| 1             | 300           | 30         | 3%         |
| 301           | 700           | 50         | 8%         |
| 701           | 1100          | 100        | 11%        |
| 1101          | 1700          | 150        | 16%        |
| 1701          | adelante      | 220        | 20%        |

Cuando el sueldo bruto excede el salario mínimo, se calcula el excedente y se busca éste en la tabla para determinar en cuál rango se encuentra. El impuesto será la cuota fija más el porcentaje indicado de la diferencia del excedente sobre el límite inferior.

6. Elaborar un algoritmo que contabilice una cuenta de cheques. Al inicio se le introduce el nombre del cuentahabiente y el saldo inicial. A continuación se pueden hacer depósitos y retiros. Cuando sea depósito se incrementa al saldo y cuando sea retiro se resta. Este programa terminará cuando ya no se desee hacer movimientos. Se requiere la impresión del siguiente reporte:

## ESTADO DE CUENTA

CUENTAHABIENTE: XXXXXXXXXXXXXXXXXXXXXXX  
SALDO INICIAL: 99,999,999.99

| MOVIMIENTO | DEPÓSITO     | RETIRO       | SALDO         |
|------------|--------------|--------------|---------------|
| 1          | 999,999.99   |              | 99,999,999.99 |
| 2          |              | 999,999.99   | 99,999,999.99 |
| 3          | 999,999.99   |              | 99,999,999.99 |
| 4          |              | 999,999.99   | 99,999,999.99 |
| N          |              |              |               |
| TOTALES    | 9,999,999.99 | 9,999,999.99 | 99,999,999.99 |

7. Igual al Ejercicio 5.1.2.1, excepto en lo siguiente:
  - Si el pedido es por más de 50 hojas, se hace un 5% de descuento adicional.
  - Si el pedido es por más de 100 hojas, se hace un 10% de descuento adicional.
8. Elaborar un algoritmo que lea los datos de varios estudiantes: nombre y tres calificaciones parciales e imprimir el siguiente reporte:

| REPORTE DE CALIFICACIONES |              |
|---------------------------|--------------|
| NOMBRE                    | CALIF. FINAL |
| XXXXXXXXXXXXXXXXXXXXXX    | 999.99       |
| XXXXXXXXXXXXXXXXXXXXXX    | 999.99       |
| .                         | .            |
| XXXXXXXXXXXXXXXXXXXXXX    | 999.99       |
| TOTAL 999 ALUMNOS         |              |

Para aprobar el curso, debe tener 70 o más en cada una de las tres calificaciones, la calificación final será el promedio. En caso de haber reprobado uno o más exámenes ordinarios, la calificación final será NA (NO ACREDITADO).

9. Se tienen los datos del transporte del elevador de un edificio, por cada viaje hecho durante el día, se tienen los siguientes datos:

Viaje:

Cantidad de personas: --

Peso del viaje: --

Viaje:

Cantidad de personas: --

Peso del viaje: --

-----

-----

-----  
-----  
Viaje:

Cantidad de personas: --

Peso del viaje: --

Elaborar un algoritmo que lea los datos de los viajes del día, y al final que imprima:

|                                     |         |
|-------------------------------------|---------|
| ESTADÍSTICA DEL DÍA                 |         |
| CANTIDAD DE VIAJES:                 | 999     |
| CANTIDAD DE PERSONAS TRANSPORTADAS: | 999     |
| PESO TRANSPORTADO (KILOS):          | 9999.99 |
| PROMEDIO DE PERSONAS POR VIAJE:     | 99.99   |
| PROMEDIO DE PESO POR VIAJE:         | 9999.99 |

- 10 El equipo de futbol soccer CACHORROS requiere llevar un conteo de las acciones que realiza durante un partido, las acciones están catalogadas como sigue:

|                   |                     |                    |
|-------------------|---------------------|--------------------|
| 1 tiro a gol      | 7 amonestación      | 13 pase equivocado |
| 2 tiro desviado   | 8 gol anotado       | 0 fin de juego     |
| 3 falta recibida  | 9 gol recibido      |                    |
| 4 falta cometida  | 10 balón perdido    |                    |
| 5 fuera de juego  | 11 balón recuperado |                    |
| 6 tiro de esquina | 12 pase correcto    |                    |

Elaborar un algoritmo que permita capturar las acciones que se den en el partido, por acción se introduce el número de acción, al terminar el partido, con la acción 0, se debe imprimir la siguiente estadística:

| ACCIONES             | ESTADÍSTICA DEL JUEGO |             |
|----------------------|-----------------------|-------------|
|                      | CACHORROS             | OTRO EQUIPO |
| tiros a gol:         | ---                   | ---         |
| tiros desviados:     | ---                   | ---         |
| faltas recibidas:    | ---                   | ---         |
| faltas cometidas:    | ---                   | ---         |
| fueras de juego:     | ---                   | ---         |
| tiros de esquina:    | ---                   | ---         |
| amonestaciones:      | ---                   | ---         |
| goles anotados:      | ---                   | ---         |
| goles recibidos:     | ---                   | ---         |
| balones perdidos:    | ---                   | ---         |
| balones recuperados: | ---                   | ---         |
| pases correctos:     | ---                   | ---         |
| pases equivocados:   | ---                   | ---         |

EQUIPO GANADOR: XXXXXXXXXXXXXXXX o bien, si no hubo ganador, imprimir  
EL JUEGO FINALIZÓ EMPATADO

11. Se tienen los datos de varios automóviles importados, elaborar un algoritmo que permita leer los datos de cada automóvil: marca, origen y costo; imprimir el siguiente reporte:

| REPORTE DE AUTOMÓVILES IMPORTADOS                   |                |           |           |             |
|-----------------------------------------------------|----------------|-----------|-----------|-------------|
| MARCA                                               | ORIGEN         | COSTO     | IMPUESTO  | PRECIO VTA. |
| XXXXXXXXXXXXXX                                      | XXXXXXXXXXXXXX | 99,999.99 | 99,999.99 | 99,999.99   |
| XXXXXXXXXXXXXX                                      | XXXXXXXXXXXXXX | 99,999.99 | 99,999.99 | 99,999.99   |
| -                                                   | -              | -         | -         | -           |
| XXXXXXXXXXXXXX                                      | XXXXXXXXXXXXXX | 99,999.99 | 99,999.99 | 99,999.99   |
| TOTAL 999 AUTOMÓVILES                               |                | 99,999.99 | 99,999.99 | 99,999.99   |
| ALEMANIA: --                                        |                |           |           |             |
| JAPÓN: --                                           |                |           |           |             |
| ITALIA: --                                          |                |           |           |             |
| EUA: --                                             |                |           |           |             |
| PAÍS DEL QUE SE IMPORTARON MÁS AUTOMÓVILES: X-----X |                |           |           |             |

Cálculos:

- IMPUESTO si el origen es Alemania el impuesto es del 20%, si es de Japón el impuesto es del 30%, si es de Italia el 15% y si es de EUA el 8%.
- PRECIO VTA. se suma el costo más el impuesto.
- TOTALES se pide el total de automóviles importados, así como totales del costo, impuesto y precio de venta. Por último el total de automóviles importados de cada país.

12. En el hotel Guamúchil se tienen los datos de los huéspedes, por cada huésped se tiene: el nombre, el tipo de habitación (1,2,3,4,5) y la cantidad de días que la ocupó; se hace un descuento del 10% si el cliente se hospeda más de 5 días, del 15% si se hospeda más de 10 días y del 20% si se hospeda más de 15 días; de acuerdo con el tipo de habitación se tienen las tarifas:

| Tipo habitación | Tarifa diaria |
|-----------------|---------------|
| 1               | 120.00        |
| 2               | 155.00        |
| 3               | 210.00        |
| 4               | 285.00        |
| 5               | 400.00        |

Elaborar un algoritmo que lea los datos de los huéspedes e imprima el siguiente reporte:

| REPORTE DE HUÉSPEDES       |      |          |                 |                 |                 |
|----------------------------|------|----------|-----------------|-----------------|-----------------|
| NOMBRE DEL HUÉSPED         | DÍAS | TARIFA   | SUBTOTAL        | DESCTO.         | TOTAL           |
| XXXXXXXXXXXXXXXXXXXX       | 999  | 9,999.99 | 9,999.99        | 9,999.99        | 9,999.99        |
| XXXXXXXXXXXXXXXXXXXX       | 999  | 9,999.99 | 9,999.99        | 9,999.99        | 9,999.99        |
| -                          | -    | -        | -               | -               | -               |
| -                          | -    | -        | -               | -               | -               |
| XXXXXXXXXXXXXXXXXXXX       | 999  | 9,999.99 | 9,999.99        | 9,999.99        | 9,999.99        |
| <b>TOTAL 999 HUÉSPEDES</b> |      |          | <b>9,999.99</b> | <b>9,999.99</b> | <b>9,999.99</b> |

**TOTAL DE DÍAS DE OCUPACIÓN:**

TIPO 1: -- TIPO 2: -- TIPO 3: -- TIPO 4: -- TIPO 5: --

**TARIFA** es el precio de la habitación de acuerdo con el tipo.

**SUBTOTAL** se multiplica el número de días por la tarifa.

**DESCTO.** es el descuento que se le hace de acuerdo con la cantidad de días.

**TOTAL** se calcula SUBTOTAL menos DESCTO.

**TOTAL DE DÍAS DE OCUPACIÓN** es la sumatoria de las cantidades de días.

TIPO1 es el total de días de ocupación de habitación tipo 1, y así para todos.

13. En la carrera Ingeniería Industrial y de Sistemas de la Universidad de Sonora, se tienen varios alumnos, y por cada alumno los datos:

Nombre del alumno: X-----X

Cada alumno puede haber cursado varias materias, por cada materia que cursó, se tienen los datos:

Materia: X-----X

Calificación 1: ---

Calificación 2: ---

Calificación 3: ---

-----

-----

-----

Nombre del alumno: X-----X

Materia: X-----X

Calificación 1: ---

Calificación 2: ---

Calificación 3: ---

-----

-----

-----

-----

Elaborar un algoritmo que permita leer los datos de cada uno de los alumnos e imprima el siguiente reporte:

| REPORTE DE MATERIAS CURSADAS |                |                  |                          |
|------------------------------|----------------|------------------|--------------------------|
| NOMBRE                       | MATERIA        | CALIF. FINAL     | OBSERVACIÓN              |
| XXXXXXXXXXXXXX               | XXXXXXXXXXXXXX | 999.99           | APROBADO                 |
|                              | XXXXXXXXXXXXXX | 999.99           | REPROBADO                |
|                              | -              | -                | -                        |
|                              | XXXXXXXXXXXXXX | 999.99           | APROBADO                 |
| TOTAL ALUMNO                 | 999 MATERIAS   | PROMEDIO: 999.99 |                          |
| XXXXXXXXXXXXXX               | XXXXXXXXXXXXXX | 999.99           | APROBADO                 |
|                              | XXXXXXXXXXXXXX | 999.99           | REPROBADO                |
|                              | -              | -                | -                        |
|                              | XXXXXXXXXXXXXX | 999.99           | APROBADO                 |
| TOTAL ALUMNO                 | 999 MATERIAS   | PROMEDIO: 999.99 |                          |
| - - -                        | TOTAL ALUMNO   | 999 MATERIAS     | PROMEDIO: 999.99         |
| - - -                        |                |                  |                          |
|                              | TOTAL GENERAL  | 999 ALUMNOS      | PROMEDIO GENERAL: 999.99 |

Cálculos:

- CALIF. FINAL es el promedio de las tres calificaciones de la materia.
- OBSERVACIÓN es el comentario APROBADO o REPROBADO según sea la calificación final.
- TOTAL ALUMNO se imprime el número de materias que cursó y el promedio de las calificaciones finales de todas las materias que cursó.
- TOTAL GENERAL se imprime la cantidad de alumnos procesados y el promedio general de todos los alumnos.

14. Similar al Ejercicio 5.1.2.8, sólo que ahora se debe imprimir el siguiente reporte:

| REPORTE DE INCENTIVOS                   |                |           |
|-----------------------------------------|----------------|-----------|
| NOMBRE                                  | TOTAL VENDIDO  | INCENTIVO |
| XXXXXXXXXXXXXX                          | 99,999.99      | 99,999.99 |
| XXXXXXXXXXXXXX                          | 99,999.99      | 99,999.99 |
| -                                       | -              | -         |
| -                                       | -              | -         |
| XXXXXXXXXXXXXX                          | 99,999.99      | 99,999.99 |
| TOTAL                                   | 999 VENDEDORES | 99,999.99 |
| EL MEJOR VENDEDOR ES: XXXXXXXXXXXXXXXXX |                |           |
| CON EL TOTAL VENDIDO: 99,999.99         |                |           |

Es decir, que es el mismo reporte, sólo se añaden dos datos más al final: El nombre del mejor vendedor (el que haya vendido más), y cuánto fue el total vendido por el mismo. Vamos a suponer que no va a coincidir, que dos vendedores hayan vendido la misma cantidad.

15. En una cooperativa pesquera de Guaymas, Sonora, México, se tienen varios pescadores de camarón y cada pescador hace varios viajes de pesca. Se tienen los datos: Nombre del pescador y la cantidad de kilogramos de camarón que entregó por cada uno de los viajes que realizó. Elaborar un algoritmo que lea esos datos y que imprima el siguiente reporte:

| REPORTE DE PESCA DE CAMARÓN                               |                  |           |
|-----------------------------------------------------------|------------------|-----------|
| NOMBRE DEL PESCADOR                                       | TOTAL PESCA (kg) | SUELDO    |
| XXXXXXXXXXXXXXXXXXXXXX                                    | 999              | 99,999.99 |
| XXXXXXXXXXXXXXXXXXXXXX                                    | 999              | 99,999.99 |
| -                                                         | -                | -         |
| XXXXXXXXXXXXXXXXXXXXXX                                    | 999              | 99,999.99 |
| TOTAL 999 PESCADORES                                      | 9999             | 99,999.99 |
| NOMBRE PESCADOR CON MAYOR PESCA: XXXXXXXXXXXXXXXXXXXXXXXX |                  |           |
| PESCA QUE REALIZÓ: 999                                    |                  |           |
| NOMBRE PESCADOR CON MENOR PESCA: XXXXXXXXXXXXXXXXXXXXXXXX |                  |           |
| PESCA QUE REALIZÓ: 999                                    |                  |           |

Cálculos:

- Se lee el nombre de un pescador, luego cada una de las cantidades de kilogramos pescados por viaje de pesca, se suman estas cantidades para calcular el TOTAL PESCA (kg); en otras palabras, es la sumatoria de las cantidades pescadas de todos los días que fue de pesca.
- El sueldo se calcula a 30.00 por kilogramo.
- Al final se pide el TOTAL de pescadores, el TOTAL del TOTAL PESCA (kg), y el total de los sueldos de todos los pescadores. Además el nombre del pescador que realizó la mayor pesca y cuánto fue ésta, así como el nombre del pescador que realizó la menor pesca y cuánto fue ésta. Se supone que el TOTAL PESCA (kg) de los pescadores es diferente.

16. En un equipo de baloncesto se tienen varios jugadores y cada jugador participó en varios juegos. Se tienen los datos: Nombre del jugador y la cantidad de puntos que anotó por cada uno de los juegos en que participó. Elaborar un algoritmo que lea esos datos y que imprima el siguiente reporte:

| REPORTE DE ANOTACIONES                                      |              |                    |
|-------------------------------------------------------------|--------------|--------------------|
| NOMBRE DEL JUGADOR                                          | TOTAL PUNTOS | NIVEL DE ANOTACIÓN |
| XXXXXXXXXXXXXXXXXXXXXX                                      | 999          | BAJO               |
| XXXXXXXXXXXXXXXXXXXXXX                                      | 999          | MEDIO              |
| -                                                           | -            | -                  |
| XXXXXXXXXXXXXXXXXXXXXX                                      | 999          | ALTO               |
| TOTAL 999 JUGADORES                                         | 9999         |                    |
| NOMBRE JUGADOR MAYOR TOTAL PUNTOS: XXXXXXXXXXXXXXXXXXXXXXXX |              |                    |
| TOTAL PUNTOS QUE ANOTÓ: 999                                 |              |                    |
| NOMBRE JUGADOR MENOR TOTAL PUNTOS: XXXXXXXXXXXXXXXXXXXXXXXX |              |                    |
| TOTAL PUNTOS QUE ANOTÓ: 999                                 |              |                    |

Cálculos:

- Se lee el nombre de un jugador, luego cada una de las cantidades de puntos anotados por juego en que participó, se suman estas cantidades para calcular el TOTAL PUNTOS; en otras palabras, es la sumatoria de las cantidades anotadas de todos los juegos en que participó.
- El nivel de anotación se determina así: Imprime BAJO si anotó 100 puntos o menos; imprime MEDIO si anotó más de 100 y hasta 200 puntos; e imprime ALTO si anotó más de 200 puntos.
- Al final se pide el TOTAL de jugadores, y el TOTAL del TOTAL PUNTOS. Además el nombre del jugador que anotó el mayor TOTAL PUNTOS y cuánto fue éste, así como el nombre del jugador que anotó el menor TOTAL PUNTOS y cuánto fue éste. Se supone que el TOTAL PUNTOS de los jugadores es diferente.



Todos los ejercicios resueltos y propuestos en el siguiente punto se pueden solucionar con lo estudiado hasta este punto.

## 5.3 La repetición for

La repetición for es una estructura que permite controlar la ejecución de acciones que se repetirán un número de veces conocido de antemano. Este tipo de repetición es controlada por un contador que empieza en un valor inicial y va hasta un valor final, incrementándose o decrementándose de acuerdo con un valor, para contar la cantidad de veces que entrará al ciclo. Se dice que el for se repite N veces.

*Formato:*

```
for contador=valorInicial; condición; incremento
    Acción(es)
endfor
```

*En donde:*

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| for          | Es la palabra reservada que identifica la estructura de repetición.                                                                                                                                                                                                                                                                                                                                                                              |
| contador     | Es una variable que puede ser de tipo entero, real o carácter; la cual se utilizará como índice o contador que controlará la repetición del ciclo. El contador tomará el valorInicial, se evalúa la condición, si es verdadera, entra al ciclo for a ejecutar las acciones que están dentro del ciclo, si no es verdadera se sale del ciclo. Al llegar al endfor, éste lo regresa al for incrementando el contador de acuerdo con el incremento. |
| valorInicial | Es el valor inicial que tomará el contador, puede ser una constante, variable o expresión de acuerdo con el tipo de dato de la variable de control del ciclo. Ejemplo: i=1.                                                                                                                                                                                                                                                                      |

|             |                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| condición   | Es una expresión lógica mediante la que se establece la condición de ejecución del ciclo, es decir, si se cumple entra al ciclo; si no se cumple se sale del ciclo.<br>Ejemplo: <code>i&lt;=10.</code> |
| Acción (es) | Es una acción o grupo de acciones en pseudocódigo que se ejecutarán adentro del ciclo.                                                                                                                 |
| incremento  | Es una expresión aritmética mediante la cual se lleva a cabo el incremento del contador del ciclo. Ejemplos:                                                                                           |

```
i=i+1 o i++
m=m-1 o m--
x=x+2
z=z+2.5
```

**Nota:** En los lenguajes C, C++, Java y derivados existe el operador aritmético `++` que significa incremento de 1, y el operador `--` que significa decremento de 1.  
**endfor** Delimita el fin del ciclo.

#### Funcionamiento:

1. Se inicia el contador con el valorInicial y se evalúa la condición; si se cumple, entra al ciclo y ejecuta la(s) acción(es).

```
for [ contador=valorInicial; condición; incremento ]
    ↓ Acción(es)
endfor
```

2. Al llegar al `endfor`, remite el control al inicio del ciclo, actualizando el valor del contador de acuerdo con el incremento (o decremento).

```
→ for contador=valorInicial; condición; incremento
    Acción(es)
endfor
```

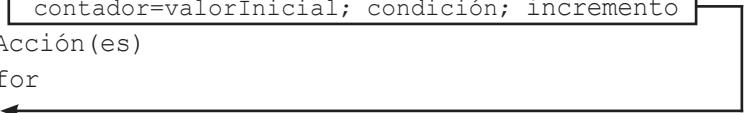
3. Al volver el control al inicio del ciclo, se evalúa la condición:  
 a) Si se cumple, entra al ciclo a ejecutar la(s) acción(es).

```
for [ contador=valorInicial; condición; incremento ]
    ↓ Acción(es)
endfor
```

Después de lo anterior, llega al `endfor`, el cual remite el control al inicio del `for`, actualizando el valor del contador de acuerdo con el incremento (o decremento).

- b) Si no se cumple, se sale del ciclo, dirigiéndose a la siguiente acción después del `endfor`, es decir, se sale del ciclo.

```
for [contador=valorInicial; condición; incremento]
    Acción(es)
endfor
```



Ejemplo:

```
for i=1; i<=10; i++
    Imprimir i
endfor
```

Se trata de un ciclo repetitivo en el que la acción (`Imprimir i`) se ejecutará diez veces, ya que el contador `i` tomará el valor inicial de 1, luego de 2 y así sucesivamente hasta llegar a 10, con incrementos de uno; donde `i` es una variable de tipo entero que debe ser declarada antes de iniciar el `for`; en declaraciones de variables.

Al iniciar `i` toma el valor de 1. Se evalúa la condición: `¿ i<=10 ?` Si se cumple, ejecuta lo que está dentro del ciclo (`Imprimir i`) la primera vez `i` tendrá valor de 1, por lo tanto, la condición se cumple y se imprime `i`.

Después de lo anterior llega al `endfor`, el cual remite el control hacia el encabezado del `for`; en este momento se aumenta el contador `i` en 1. Al llegar otra vez al `for`, evalúa de nuevo la condición; si se cumple, como es nuestro caso, entrará de nuevo al ciclo y así sucesivamente. En el ejemplo:

`i` toma el valor de 1, entra al ciclo e imprimirá: 1  
`i` toma el valor de 2, entra al ciclo e imprimirá: 2  
`i` toma el valor de 3, entra al ciclo e imprimirá: 3  
`i` toma el valor de 4, entra al ciclo e imprimirá: 4  
`i` toma el valor de 5, entra al ciclo e imprimirá: 5  
`i` toma el valor de 6, entra al ciclo e imprimirá: 6  
`i` toma el valor de 7, entra al ciclo e imprimirá: 7  
`i` toma el valor de 8, entra al ciclo e imprimirá: 8  
`i` toma el valor de 9, entra al ciclo e imprimirá: 9  
`i` toma el valor de 10, entra al ciclo e imprimirá: 10

**Nota:** La condición `i<=10` quiere decir que cuando el valor de `i` sea hasta 10, va a entrar al ciclo; cuando `i` tenga el valor 11, no se cumplirá la condición y saldrá del ciclo.

Si deseamos que entre al ciclo 30 veces, la condición será `i<=30` y saldrá del ciclo cuando `i` tenga 31. Si deseamos que entre al ciclo 50 veces, la condición será `i<=50` y saldrá del ciclo cuando `i` tenga 51.

El ciclo puede plantearse:

```
for i=10; i>=1; i--
    Imprimir i
endfor
```

En este caso `i` tomará primero el valor de 10, luego el 9, y así sucesivamente hasta llegar a 1, debido a que se está planteando un ciclo con decremento, el valor inicial debe ser mayor que el valor de la condición (`i>=1`); en este caso entrará al ciclo hasta que `i` tome el valor de 1, saldrá del ciclo cuando `i` tenga 0 (cero).

En este ejemplo se imprimirá: 10 9 8 7 6 5 4 3 2 1

Ejemplo:

Elaborar un algoritmo que calcule e imprima la suma de los números del 1 hasta el 100.

A continuación se tiene el algoritmo de la solución:

```
Algoritmo SUMA NÚMEROS 1-100
1. Declarar
    Variables
        indice, sumatoria: Entero
2. sumatoria = 0
3. for indice=1; indice<=100; indice++
    a. sumatoria = sumatoria + indice
4. endfor
5. Imprimir sumatoria
6. Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C512.C y Programa en Java: SumaNumeros1.java

*Explicación:*

1. Se declaran las variables
  - `indice` para manejar el contador del ciclo
  - `sumatoria` para calcular la sumatoria de los números del 1 al 100
2. Se inicia el acumulador en cero
3. Ciclo for desde `indice = 1` hasta 100 con incrementos de 1
  - a. Se incrementa el acumulador `sumatoria` con `indice`

4. Fin del ciclo for
5. Se imprime la sumatoria
6. Fin del algoritmo

### 5.3.1 for anidados

Al igual que todas las estructuras de control, es posible que un ciclo for contenga anidado otro ciclo y éste a otro; veamos, por ejemplo, el siguiente ciclo:

```
for i=1; i<=10; i++  
    Imprimir i  
    for j=1; j<=10; j++  
        Imprimir j  
    endfor  
endfor
```

Se trata de un ciclo controlado por *i*, dentro del cual se imprime el valor de *i*; además, contiene anidado un ciclo for controlado por *j*, donde se imprime el valor de *j*.

Por cada una de las veces que entre en el primer ciclo for (el más externo), entrará diez veces al ciclo más interno; esto significa que por las diez veces que entrará en *i*, lo hará 100 veces en *j*.

### 5.3.2 Ejercicios resueltos para la repetición for

#### Ejercicio 5.3.2.1

Elaborar un algoritmo que calcule e imprima la suma de los números pares del 2 hasta el 160.

(Primero hágalo usted, después compare la solución.)

Algoritmo SUMA NÚMEROS 2-160

1. Declarar  
 Variables  
 i, suma: Entero
2. suma = 0
3. for i=2; i<=160; i=i+2  
 a. suma = suma + i
4. endfor
5. Imprimir suma
6. Fin

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C513.C y Programa en Java: SumaNumeros2.java



*Explicación:*

1. Se declaran las variables
  - i para manejar el contador del ciclo
  - suma para calcular la sumatoria de los números del 2 al 160
2. Se inicia el acumulador en cero
3. Ciclo for desde i = 2 hasta 160 con incrementos de 2
  - a. Se incrementa el acumulador suma con i
4. Fin del ciclo for
5. Se imprime la suma
6. Fin del algoritmo

**Ejercicio 5.3.2.2**

Elaborar un algoritmo que calcule e imprima la suma  $1 + 1/2 + 1/3 + 1/4 \dots + 1/50$ .

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo SUMATORIA
1. Declarar
   Variables
      i: Entero
      valor, suma: Real
2. suma = 0
3. for i=1; i<=50; i++
   a. valor = 1/i
   b. suma = suma + valor
4. endfor
5. Imprimir suma
6. Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C514.C y Programa en Java: SumaNumeros3.java

*Explicación:*

1. Se declaran las variables
  - i para controlar el ciclo
  - valor para calcular cada uno de los valores a sumar
  - suma para calcular la sumatoria
2. Se inicia el acumulador suma en cero
3. Ciclo for desde i = 1 hasta 50 con incrementos de 1
  - a. Se calcula el valor = 1 / i
  - b. Se incrementa suma con valor

4. Fin del for
5. Se imprime la suma
6. Fin del algoritmo

#### Ejercicio 5.3.2.3

Elaborar un algoritmo que lea 20 números y que calcule e imprima el promedio de dichos números.

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo PROMEDIO DE 20 NÚMEROS
1. Declarar
    Variables
        i, numero, sumatoria: Entero
        promedio: Real
2. sumatoria = 0
3. for i=1; i<=20; i++
    a. Solicitar Número
    b. Leer numero
    c. sumatoria = sumatoria + número
4. endfor
5. promedio = sumatoria / 20
6. Imprimir promedio
7. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C515.C y Programa en Java: PromedioNumeros1.java



#### Explicación:

1. Se declaran las variables
2. Se inicia la sumatoria en cero
3. Ciclo for desde  $i = 1$  hasta 20 con incrementos de 1
  - a. Se solicita el número
  - b. Se lee el número
  - c. Se incrementa sumatoria con número
4. Fin del for
5. Se calcula el promedio dividiendo sumatoria entre 20
6. Se imprime el promedio
7. Fin del algoritmo

**Ejercicio 5.3.2.4**

Elaborar un algoritmo que solicite la cantidad de números a procesar y lea la respuesta en N; luego que lea los N números y calcule e imprima el promedio de dichos números.

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo PROMEDIO DE N NÚMEROS
1. Declarar
    Variables
        i, n, numero, sumatoria: Entero
        promedio: Real
2. Solicitar cantidad de números a procesar
3. Leer n
4. sumatoria = 0
5. for i=1; i<=n; i++
    a. Solicitar Número
    b. Leer numero
    c. sumatoria = sumatoria + numero
6. endfor
7. promedio = sumatoria / n
8. Imprimir promedio
9. Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C516.C y Programa en Java: PromedioNumeros2.java

*Explicación:*

1. Se declaran las variables
2. Se solicita la cantidad de números
3. Se lee en n
4. Se inicia sumatoria en cero
5. Ciclo for desde i = 1 hasta n
  - a. Se solicita el número
  - b. Se lee en numero
  - c. Se incrementa sumatoria con numero
6. Fin del for
7. Se calcula el promedio dividiendo sumatoria entre n
8. Se imprime el promedio
9. Fin del algoritmo

**Ejercicio 5.3.2.5**

Elaborar un algoritmo que imprima el seno, coseno y arco tangente de X; para valores de X desde -1 hasta 1 con intervalos de 0.2. Debe imprimir una tabla:

| X     | Seno X | Coseno X | Arco tangente X |
|-------|--------|----------|-----------------|
| - 1.0 | 99.99  | 99.99    | 99.99           |
| - 0.8 | 99.99  | 99.99    | 99.99           |
| -     | -      | -        | -               |
| 1.0   | 99.99  | 99.99    | 99.99           |

(Primero hágalo usted, después compare la solución.)

Algoritmo SENO COSENO ARCO TANGENTE DE -1 HASTA 1

1. Declarar
  - Variables
    - x, senoX, cosenoX, arcoTanX: Real
2. Imprimir encabezado
3. for x=-1; x<=1; x=x+0.2
  - a. senoX = Seno(x)
  - b. cosenoX = Coseno(x)
  - c. arcoTanX = ArcTan(x)
  - d. Imprimir x, senoX, cosenoX, arcoTanX
4. endfor
5. Fin

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C517.C y Programa en Java: Logaritmos1.java

**Explicación:**

1. Se declaran las variables
2. Se imprime el encabezado
3. Inicia ciclo for desde x = -1 hasta 1 con incrementos de 0.2
  - a. Se calcula el senoX
  - b. Se calcula cosenoX
  - c. Se calcula arcoTanX
  - d. Se imprimen x, senoX, cosenoX, arcoTanX
4. Fin del ciclo for
5. Fin del algoritmo

**Ejercicio 5.3.2.6**

Una temperatura en grados Fahrenheit (F) se convierte a grados Celsius (C), con la fórmula:  $C = \frac{5}{9} * (F-32)$ . Elaborar un algoritmo que imprima una tabla desde 1 hasta 65 (con intervalos de 1) grados Fahrenheit con sus equivalencias en grados Celsius:

| Fahrenheit | Celsius |
|------------|---------|
| 1          | 99.99   |
| 2          | 99.99   |
| -          |         |
| -          |         |
| 65         | 99.99   |

(Primero hágalo usted, después compare la solución.)

```
Algoritmo EQUIVALENCIAS FAHRENHEIT CELSIUS
1. Declarar
    Variables
        fahrenheit, celsius: Real
2. Imprimir encabezado
3. for fahrenheit=1; fahrenheit<=65; fahrenheit++
    a. celsius = 5/9 * (fahrenheit-32)
    b. Imprimir fahrenheit, celsius
4. endfor
5. Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C518.C y Programa en Java: EquivalenciasFahr1.java

*Explicación:*

1. Se declaran las variables
2. Imprime el encabezado
3. Inicia ciclo for desde fahrenheit = 1 hasta 65 con incrementos de 1
  - a. Calcula celsius
  - b. Imprime fahrenheit, celsius
4. Fin del for
5. Fin del algoritmo

#### Ejercicio 5.3.2.7

La escuela FORD No. 8 de Guamúchil, Sinaloa, tiene actualmente 750 alumnos, se espera tener un crecimiento anual del 12%. Elaborar un algoritmo que calcule e imprima la población estudiantil que se espera tener en el año 2035.

(Primero hágalo usted, después compare la solución.)

```
Algoritmo ESTIMAR POBLACIÓN
1. Declarar
    Variables
        n: Entero
        pobFinal: Real
```

```
2. pobFinal = 750
3. for n=2010; n<=2035; n++
   a. pobFinal = pobFinal + (pobFinal * 0.12)
4. endfor
5. Imprimir pobFinal
6. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C519.C y Programa en Java: PoblacionEstudiantil.java



*Explicación:*

1. Se declaran las variables
2. Se inicia pobFinal en 750
3. Ciclo for desde n = 2010 hasta 2035 con incrementos de 1
  - a. Se calcula pobFinal añadiéndole el 12%
4. Fin ciclo for
5. Se imprime la pobFinal
6. Fin del algoritmo

#### Ejercicio 5.3.2.8

Elaborar un algoritmo que lea un valor N, entero y positivo, y que le calcule e imprima su factorial. Por ejemplo, si se lee el 5, su factorial es el producto de  $5 \times 4 \times 3 \times 2 \times 1$ . El factorial de 0 es 1.

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo FACTORIAL
1. Declarar
   Variables
      num, i, fact: Entero
2. Solicitar Número
3. Leer num
4. if num == 0 then
   a. fact = 1
5. else
   a. fact = 1
   b. for i=num; i>=1; i--
      1. fact = fact * i
   c. endfor
6. endif
7. Imprimir fact
8. Fin
```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C520.C y Programa en Java: Factorial1.java

*Explicación:*

1. Se declaran las variables
2. Se solicita el número
3. Se lee en num
4. Se compara si num == 0, si se cumple, entonces
  - a. Se calcula fact = 1
5. Si no (else)
  - a. Se inicia fact = 1
  - b. Ciclo for desde i = num hasta 1 con decrementos de -1
    1. Se calcula fact = fact \* i
  - c. Fin del for
6. Fin del if
7. Se imprime el factorial fact
8. Fin del algoritmo

La tabla 5.2 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 5 de la Web del libro.

**Tabla 5.2**

| Ejercicio          | Descripción                                             |
|--------------------|---------------------------------------------------------|
| Ejercicio 5.2.2.9  | Calcula el factorial a N números                        |
| Ejercicio 5.2.2.10 | Calcula números aplicando la secuencia Fibonacci        |
| Ejercicio 5.2.2.11 | Calcula las potencias de los números del 1 al 8         |
| Ejercicio 5.2.2.12 | Realiza cálculos con la ecuación cuadrática             |
| Ejercicio 5.2.2.13 | Calcula intereses de una inversión fija                 |
| Ejercicio 5.2.2.14 | Calcula intereses de una inversión leyendo datos        |
| Ejercicio 5.2.2.15 | Procesa la producción de 15 trabajadores con dos ciclos |
| Ejercicio 5.2.2.16 | Procesa las ventas de varios vendedores                 |
| Ejercicio 5.2.2.17 | Procesa la producción de 15 obreros                     |
| Ejercicio 5.2.2.18 | Procesa la producción de 15 obreros (otra forma)        |
| Ejercicio 5.2.2.19 | Procesa la producción de varios obreros                 |

### 5.3.3 Simulación del for con do...while

Como ya se explicó anteriormente, el for es controlado por un contador. También se ha descrito la forma de cómo manejar contadores con do...while. El ejemplo:

```
for i=1; i<=10; i++  
    Imprimir i  
endfor
```

Lo podemos implementar con do...while de la siguiente forma:

```
i=0  
do  
    i=i+1  
    Imprimir i  
while i < 10
```

*Explicación:*

- Antes de entrar al ciclo se inicia el contador *i* en cero.
- Dentro del ciclo se incrementa *i* en 1 y se imprime.
- La condición se plantea mientras el contador *i* sea menor que 10, vuelve al do.

Es decir, que con el do...while se puede implementar soluciones a problemas que son naturales para el for.

A continuación se plantearán algunos algoritmos que se resolvieron durante este capítulo con for, pero ahora utilizando do...while, y, como podrá observar, la diferencia será la forma de manejar el contador, misma que se describió anteriormente.

#### Ejercicio 5.3.3.1

Elaborar un algoritmo similar al Ejercicio 5.2.2.5, utilizando do...while en lugar de for.

```
Algoritmo SENO COSENO ARCO TANGENTE DE -1 HASTA 1  
1. Declarar  
    Variables  
        x, senoX, cosenoX, arcoTanX: Real  
2. Imprimir encabezado  
3. x = -1.2  
4. do  
    a. x = x + 0.2  
    b. senoX = Seno(x)  
    c. cosenoX = Coseno(x)  
    d. arcoTanX = ArcTan(x)  
    e. Imprimir x, senoX, cosenoX, arcoTanX
```

- 5. while  $x < 1$
- 6. Fin



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C532.C y Programa en Java: Logaritmos2.java

La tabla 5.3 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 5 de la Web del libro.

**Tabla 5.3**

| Ejercicio         | Descripción                                 |
|-------------------|---------------------------------------------|
| Ejercicio 5.2.3.2 | Calcula equivalencias fahrenheit celsius    |
| Ejercicio 5.2.3.3 | Calcula factoriales a N números             |
| Ejercicio 5.2.3.4 | Realiza cálculos con la ecuación cuadrática |

## 5.4 Ejercicios propuestos para la repetición for

1. Elaborar un algoritmo que lea N números diferentes y calcule e imprima el mayor y el menor.
2. Elaborar un algoritmo que lea un valor N y que imprima un triángulo de asteriscos, como se muestra a continuación, si el valor leído es 5 imprimir:  
 \*  
 \* \*  
 \* \* \*  
 \* \* \* \*  
 \* \* \* \* \*
3. Elaborar un algoritmo que permita leer un número N par, y calcule e imprima la suma de los números pares del 2 hasta el número leído. Si el número leído es menor a 2 debe imprimir un mensaje de error.
4. Elaborar un algoritmo que lea un valor N; luego que lea N números de entrada e imprimir el total, el promedio, el mayor y el menor.
5. Elaborar un algoritmo que permita leer un número e imprima una tabla con las potencias de los números desde 1 hasta el número leído. La potencia de 1, es 1 elevado a la potencia 1. La potencia de 2, es 2 elevado a la potencia 2, y así sucesivamente, hasta la potencia del número leído.

| Número | Potencia |
|--------|----------|
| 1      | 9999     |
| 2      | 9999     |
| -      |          |
| -      |          |
| 8      | 9999     |

6. Elaborar un algoritmo para calcular e imprimir el cuadrado de los números impares del 1 al 15.
7. Elaborar un algoritmo que permita leer el valor inicial y el valor final en grados Fahrenheit; e imprima una tabla con equivalencias en grados Celsius, desde el valor inicial hasta el valor final de 1 en 1.

| Fahrenheit    | Celsius |
|---------------|---------|
| Valor inicial | 99.99   |
| -             | -       |
| -             | -       |
| -             | -       |
| -             | -       |
| Valor final   | 99.99   |

8. Elaborar un algoritmo similar al anterior, sólo que permita leer el valor inicial y el valor final en grados Celsius, e imprima la tabla con equivalencias en grados Fahrenheit, desde el valor inicial hasta el valor final de 1 en 1.
9. Elaborar un algoritmo que imprima la secante, cosecante y tangente de X; para valores de X desde -1 hasta 1 con intervalos de 0.1. Debe imprimir una tabla:

| X     | Secante X | Cosecante X | Tangente X |
|-------|-----------|-------------|------------|
| - 1.0 | 99.99     | 99.99       | 99.99      |
| - 0.8 | 99.99     | 99.99       | 99.99      |
| -     |           |             |            |
| -     |           |             |            |
| 1.0   | 99.99     | 99.99       | 99.99      |

10. Elaborar un algoritmo que permita que los valores de los coeficientes a, b y c, se comporten así: B debe ir de 1 hasta 7. C debe ir de 7 a 1. A debe tomar cada vez la diferencia de B-C, y que imprima para cada juego de valores de a, b, c, si tiene raíz única, raíces complejas o raíces reales, similar al Ejercicio 5.2.2.12.
11. Cada equipo de beisbol de la Liga Mexicana del Pacífico tiene un cuadro de 30 jugadores. Supóngase que cada equipo de la liga prepara un listado donde por cada jugador, aparecen los datos siguientes: nombre del jugador, peso, edad.

Elaborar un algoritmo para leer estos datos y que emita el siguiente reporte:

| ESTADÍSTICA DE JUGADORES |               |               |
|--------------------------|---------------|---------------|
| EQUIPO                   | PROMEDIO PESO | PROMEDIO EDAD |
| 1                        | 999.99        | 99.9          |
| 2                        | 999.99        | 99.9          |
| -                        |               |               |
| 10                       | 999.99        | 99.9          |

PROMEDIO GENERAL PESO: 999.99

PROMEDIO GENERAL EDAD: 99.9

12. Elaborar un algoritmo para determinar e imprimir una tabla de amortización de un préstamo; para ello se tienen como datos el saldo por amortizar, la tasa de interés anual y el número de meses que se tienen de plazo. Imprimir el reporte siguiente:

TABLA DE AMORTIZACIÓN

Saldo: 99,999,999.99

Interés anual: 999.99

Número de meses: 99

| MES     | SALDO INSOLUTO | CUOTA FIJA | INTERÉS   | MENSUALIDAD  |
|---------|----------------|------------|-----------|--------------|
| 1       | 999,999.99     | 99,999.99  | 9,999.99  | 999,999.99   |
| 2       | 999,999.99     | 99,999.99  | 9,999.99  | 999,999.99   |
| -       | -              | -          | -         | -            |
| N       | 999,999.99     | 99,999.99  | 9,999.99  | 999,999.99   |
| TOTALES | 9,999,999.99   | 999,999.99 | 99,999.99 | 9,999,999.99 |

Datos disponibles:

- Saldo
- Interés anual
- Número de meses (plazo)

El proceso a seguir para obtener la información es el siguiente:

- Como datos de entrada se tienen el saldo, interés anual y número de meses.
- El mes es un dato derivable, de 1 a n meses, según sea el plazo.
- El saldo insoluto es el capital que se debe en el mes correspondiente.
- La cuota fija se determina dividiendo el saldo entre el número de meses.
- El interés se determina mediante la aplicación del interés mensual sobre el saldo insoluto.
- La mensualidad se establece sumando la cuota fija más el interés.
- Los totales son el producto de la acumulación de la cuota fija, el interés y la mensualidad.

13. Elaborar un algoritmo para calcular la cantidad que se tendría ahorrada después de 10 años, si se depositan mil pesos mensualmente a una tasa de interés mensual de 3%, capitalizable cada mes, es decir, que al capital se le agregan los intereses.
14. Elaborar un algoritmo para calcular la cantidad que se tendría ahorrada después de 15 años, si se depositan quince mil pesos a una tasa de interés de 3.7% mensual, capitalizable cada mes.
15. Una empresa de teléfonos, ha decidido incrementar la tarifa de la renta mensual por uso del teléfono, en un 4% mensual. La tarifa en enero de 2011 es de 57.00. Elaborar un algoritmo que imprima el monto de la renta mensual en enero de 2012, 2013, 2014,..., 2035.

16. Elaborar un algoritmo para estimar la población estudiantil de una escuela, que se espera tener en un determinado año, los datos que se tienen son: la población actual de la escuela (número de estudiantes), el porcentaje de crecimiento anual que se espera tener, el año actual y el año al que se desea estimar el crecimiento. Todos estos datos deben ser leídos para imprimir al final lo estimado.
17. Los números Fibonacci constituyen una secuencia que empieza con 0 y 1; el número que sigue a éstos se calcula sumando los dos anteriores y así sucesivamente. Elaborar un algoritmo que lea un número N e imprima los N primeros números de la secuencia. Si N no es mayor que cero, debe imprimir un mensaje de error.
18. Elaborar un algoritmo que permita leer una medida (N) en número de metros, y que imprima una tabla de equivalencias a yardas, pulgadas y pies; desde 1 metro hasta N metros de uno en uno. Equivalencias: 1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 metro = 100 cm. Se debe imprimir la tabla siguiente:

CONVERSIONES

| METROS | YARDAS  | PULGADAS | PIES    |
|--------|---------|----------|---------|
| 1      | 9999.99 | 9999.99  | 9999.99 |
| 2      | 9999.99 | 9999.99  | 9999.99 |
| -      | -       | -        | -       |
| N      | 9999.99 | 9999.99  | 9999.99 |

19. Similar al anterior, sólo que se lee un valor inicial y un valor final: Por ejemplo, si los valores leídos son 100 y 200, las conversiones se harán desde 100 hasta 200 de uno en uno.
20. Similar al anterior, sólo que el dato a convertir es pies, es decir, es el que se leerá. También hacerlo para pulgadas y yardas.
21. Se tienen 12 escuelas; por cada escuela los datos: Nombre de la escuela, la población actual (número de estudiantes), el porcentaje de crecimiento anual que se espera tener. Estamos en el año actual. Elaborar un algoritmo que permita leer los datos de cada una de las escuelas e imprima el siguiente reporte:

REPORTE DE ESCUELAS

| NOMBRE DE ESCUELA                           | POBL. EST. ACTUAL | POBL. EST. AÑO 2035 |
|---------------------------------------------|-------------------|---------------------|
| XXXXXXXXXXXXXXXXXXXX                        | 9999.9            | 9999.9              |
| XXXXXXXXXXXXXXXXXXXX                        | 9999.9            | 9999.9              |
| -                                           | -                 | -                   |
| -                                           | -                 | -                   |
| XXXXXXXXXXXXXXXXXXXX                        | 9999.9            | 9999.9              |
| TOTAL 999 ESCUELAS                          | 9999.9            | 9999.9              |
| LA ESCUELA MAYOR SERÁ: XXXXXXXXXXXXXXXXXXXX |                   |                     |
| LA ESCUELA MENOR SERÁ: XXXXXXXXXXXXXXXXXXXX |                   |                     |

22. Elaborar un algoritmo similar al anterior, con la diferencia de que se tienen varias escuelas, es decir, no se sabe cuántas escuelas son; todo lo demás es igual al problema anterior
23. Una compañía manufacturera fabrica un solo producto, se tienen los datos de la producción de cada uno de los siete días de la semana; los datos son: cantidad de unidades producidas, costo de operación por la producción del día y costo de los materiales utilizados.

Elaborar un algoritmo que lea dichos datos e imprima el siguiente reporte:

| DIA                                | COSTOS DE PRODUCCIÓN |                  |                  |                |
|------------------------------------|----------------------|------------------|------------------|----------------|
|                                    | UNIDADES PRODUCIDAS  | COSTO PRODUCCIÓN | COSTO MATERIALES | COSTO UNITARIO |
| 99                                 | 999                  | 99,999.99        | 99,999.99        | 99,999.99      |
| 99                                 | 999                  | 99,999.99        | 99,999.99        | 99,999.99      |
| -                                  | -                    | -                | -                | -              |
| 99                                 | 999                  | 99,999.99        | 99,999.99        | 99,999.99      |
| TOTAL                              | 999                  | 99,999.99        | 99,999.99        | 99,999.99      |
| COSTO UNITARIO PROMEDIO: 99,999.99 |                      |                  |                  |                |

- COSTO UNITARIO se calcula sumando el costo de producción más el costo de materiales y dividiéndolo entre las unidades producidas.
  - TOTAL sumatoria de unidades producidas de todos los días, de los costos de producción y de los costos de materiales.
  - COSTO UNITARIO PROMEDIO la sumatoria de los costos unitarios diarios entre la cantidad de días.
24. Similar al caso anterior, excepto en lo siguiente: se tienen ocho plantas.  
Emitir el reporte:

| DIA                                        | COSTOS DE PRODUCCIÓN |                  |                  |                      |
|--------------------------------------------|----------------------|------------------|------------------|----------------------|
|                                            | UNIDADES PRODUCIDAS  | COSTO PRODUCCIÓN | COSTO MATERIALES | COSTO UNIT. PROMEDIO |
| 1                                          | 999                  | 99,999.99        | 99,999.99        | 99,999.99            |
| 2                                          | 999                  | 99,999.99        | 99,999.99        | 99,999.99            |
| -                                          | -                    | -                | -                | -                    |
| 8                                          | 999                  | 99,999.99        | 99,999.99        | 99,999.99            |
| TOTAL                                      | 999                  | 99,999.99        | 99,999.99        | 99,999.99            |
| COSTO UNITARIO PROMEDIO GENERAL: 99,999.99 |                      |                  |                  |                      |

Ahora las unidades producidas, costo de producción y costo de materiales son la sumatoria de todos los días de producción de cada planta; el costo unitario promedio es por planta, y al final el costo unitario promedio general.

25. Una compañía manufacturera de bombillas tiene 5 plantas: Hermosillo, Guamúchil, Tijuana, Culiacán y México, se tienen los datos de la producción de los días de la semana, de la siguiente forma:

Planta 1: Hermosillo

Día 1: Cantidad producida

Cantidad de defectuosas

Día 2: Cantidad producida  
 Cantidad de defectuosas  
 Día 7: Cantidad producida  
 Cantidad de defectuosas

Planta 2: Guamúchil  
 Día 1: Cantidad producida  
 Cantidad de defectuosas  
 Día 2: Cantidad producida  
 Cantidad de defectuosas  
 Día 7: Cantidad producida  
 Cantidad de defectuosas

-----  
 -----

Elaborar un algoritmo que lea dichos datos e imprima el siguiente reporte:

**REPORTE DE CONTROL DE CALIDAD**

| PLANTA     | UNIDADES PRODUCIDAS | UNIDADES DEFECTUOSAS | % DEFECTUOSAS |
|------------|---------------------|----------------------|---------------|
| HERMOSILLO | 999                 | 999                  | 999.99        |
| GUAMÚCHIL  | 999                 | 999                  | 999.99        |
| -          | -                   | -                    | -             |
| MÉXICO     | 999                 | 999                  | 999.99        |
| TOTAL      | 999                 | 999                  |               |

PORCENTAJE TOTAL DE DEFECTUOSAS: 999.99

- % DEFECTUOSAS se calcula dividiendo unidades defectuosas entre unidades producidas.
  - TOTAL sumatoria de unidades producidas y unidades defectuosas.
  - PORCENTAJE TOTAL DE DEFECTUOSAS es la sumatoria del % DEFECTUOSAS de cada planta entre el número de plantas.
26. Se tienen 5 inversiones, por cada una los datos: capital, tasa de interés anual y el plazo en número de meses. Elaborar un algoritmo que lea los datos de cada inversión e imprima el siguiente reporte:

**INVERSIÓN 1**

CAPITAL: 99,999.99

INTERÉS ANUAL: 999.99

PLAZO EN MESES: 999

| MES | CAPITAL   | INTERÉS   | SALDO     |
|-----|-----------|-----------|-----------|
| 1   | 99,999.99 | 99,999.99 | 99,999.99 |
| 2   | 99,999.99 | 99,999.99 | 99,999.99 |
| 3   | -         | -         | -         |
| -   | -         | -         | -         |
| N   | 99,999.99 | 99,999.99 | 99,999.99 |

TOTAL INTERÉS GANADO: -----

**INVERSIÓN 2**

CAPITAL: -----  
 INTERÉS ANUAL: -----  
 PLAZO EN MESES: -----

| MES                                      | CAPITAL   | INTERÉS   | SALDO     |
|------------------------------------------|-----------|-----------|-----------|
| 1                                        | 99,999.99 | 99,999.99 | 99,999.99 |
| 2                                        | 99,999.99 | 99,999.99 | 99,999.99 |
| 3                                        | -         | -         | -         |
| -                                        | -         | -         | -         |
| -                                        | -         | -         | -         |
| N                                        | 99,999.99 | 99,999.99 | 99,999.99 |
| TOTAL INTERÉS GANADO: 99,999.99          |           |           |           |
| -----                                    |           |           |           |
| TOTAL GENERAL INVERTIDO: 999,999.99      |           |           |           |
| TOTAL GENERAL INTERÉS GANADO: 999,999.99 |           |           |           |

27. Similar al Ejercicio 5.2.2.15, pero con las diferencias siguientes:

- el número de días que laboró cada trabajador puede variar respecto de los demás trabajadores, es decir, un trabajador pudo haber trabajado 5 días, otro 4, otro 6, otro 4, etcétera.
- imprimir al final del reporte:

TRABAJADOR MÁS PRODUCTIVO: XXXXXXXXXXXXXXXXXXXXXXX  
 TOTAL DE UNIDADES FABRICADAS: 9999

TRABAJADOR MENOS PRODUCTIVO: XXXXXXXXXXXXXXXXXXXXXXX  
 TOTAL DE UNIDADES FABRICADAS: 9999

Suponiendo que no habrá cantidades producidas iguales entre los trabajadores, el trabajador más productivo es el que tiene el mayor número de unidades producidas, mientras que el menos productivo, es el que tiene el menor número de unidades producidas.

28. En una empresa se tienen 15 vendedores. Por cada vendedor se tiene su nombre y la cantidad vendida en pesos por cada uno de los varios días en que laboró. Elaborar un algoritmo que lea esos datos e imprima el siguiente reporte:

| REPORTE DE SUELDOS                                 |             |           |
|----------------------------------------------------|-------------|-----------|
| NOMBRE                                             | TOTAL VENTA | SUELDO    |
| XXXXXXXXXXXXXXXXXXXX                               | 99,999.99   | 99,999.99 |
| XXXXXXXXXXXXXXXXXXXX                               | 99,999.99   | 99,999.99 |
| -                                                  | -           | -         |
| XXXXXXXXXXXXXXXXXXXX                               | 99,999.99   | 99,999.99 |
| TOTAL 999 VENDEDORES                               | 99,999.99   | 99,999.99 |
| NOMBRE DEL MEJOR VENDEDOR: XXXXXXXXXXXXXXXXXXXXXXX |             |           |
| NOMBRE DEL PEOR VENDEDOR: XXXXXXXXXXXXXXXXXXXXXXX  |             |           |

## Cálculos:

- TOTAL VENTA es la sumatoria de la venta de todos los días de trabajo.
  - SUELDO es el 4% del TOTAL VENTA del vendedor +
    - 300.00 si el TOTAL VENTA está entre 15 001.00 y 25 000.00
    - 600.00 si el TOTAL VENTA está entre 25 001.00 y 35 000.00
    - 900.00 si el TOTAL VENTA es mayor a 35 000.00
  - NOMBRE DEL MEJOR VENDEDOR es el vendedor que tenga el TOTAL VENTA mayor.
  - NOMBRE DEL PEOR VENDEDOR es el vendedor que tenga el TOTAL VENTA menor.
29. Similar al caso anterior, excepto en lo siguiente: se tienen varios vendedores y cada vendedor laboró 6 días.
30. En un equipo de baloncesto se tienen varios jugadores. Por cada jugador se tiene su nombre y la cantidad de puntos que anotó, por cada uno de los 10 partidos en que jugó. Elaborar un algoritmo que lea esos datos e imprima el siguiente reporte:

## ESTADÍSTICA DE JUGADORES

| NOMBRE                     | TOTAL PUNTOS           | NIVEL DE ANOTACIÓN |
|----------------------------|------------------------|--------------------|
| XXXXXXXXXXXXXXXXXXXXXX     | 999                    | DEFICIENTE         |
| XXXXXXXXXXXXXXXXXXXXXX     | 999                    | BUENO              |
| -                          | -                      | -                  |
| -                          | -                      | -                  |
| XXXXXXXXXXXXXXXXXXXXXX     | 999                    | EXCELENTE          |
| TOTAL 99 JUGADORES         | 999                    |                    |
| NOMBRE DEL MEJOR ANOTADOR: | XXXXXXXXXXXXXXXXXXXXXX |                    |
| NOMBRE DEL PEOR ANOTADOR:  | XXXXXXXXXXXXXXXXXXXXXX |                    |

## Cálculos:

- TOTAL PUNTOS es la sumatoria de los puntos que anotó en todos los juegos.
- NIVEL DE ANOTACIÓN es un comentario que indica
  - DEFICIENTE si TOTAL PUNTOS es menor de 40
  - BUENO si TOTAL PUNTOS está entre 40 y 90
  - EXCELENTE si TOTAL PUNTOS es mayor de 90
- TOTAL por un lado es el total de jugadores, y por el otro, es el total del total de puntos anotados por todos los jugadores.
- NOMBRE DEL MEJOR ANOTADOR es el jugador que tiene el TOTAL PUNTOS mayor.
- NOMBRE DEL PEOR ANOTADOR es el jugador que tiene el TOTAL PUNTOS menor.

31. Similar al caso anterior, excepto en lo siguiente: se tienen 12 jugadores y cada jugador participó en varios partidos.
32. Similar al ejercicio propuesto 15 del punto anterior, excepto en lo siguiente: se tienen 150 pescadores y cada pescador realizó 20 viajes.
33. Similar al ejercicio propuesto 15 del punto anterior, excepto en lo siguiente: se tienen 150 pescadores y cada pescador realizó varios viajes.
34. Similar al ejercicio propuesto 15 del punto anterior, excepto en lo siguiente: se tienen varios pescadores y cada pescador realizó 20 viajes.
35. Similar al ejercicio propuesto 16 del punto anterior, excepto en lo siguiente: se tienen 12 jugadores y cada jugador participó en 50 juegos.
36. Similar al ejercicio propuesto 16 del punto anterior, excepto en lo siguiente: se tienen 12 jugadores y cada jugador participó en varios juegos.
37. Similar al ejercicio propuesto 16 del punto anterior, excepto en lo siguiente: se tienen varios jugadores y cada jugador participó en 50 juegos.



Estos ejercicios también se pueden resolver utilizando las repeticiones do... while y while.

## 5.5 La repetición while

La repetición while es una estructura que permite controlar la ejecución de acciones que se repetirán en un rango de 0 (cero) a N veces, esto se debe a que la condición de control del ciclo se coloca al principio de la estructura, y entra al ciclo mientras la condición sea verdadera. En caso de que no se cumpla la condición, se termina el ciclo.

*Formato:*

```
while condición
    Acción(es)
endwhile
```

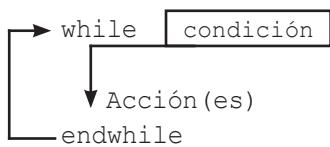
*En donde:*

|            |                                                                                    |
|------------|------------------------------------------------------------------------------------|
| while      | Identifica la estructura y su inicio como un ciclo repetitivo.                     |
| condición  | Es una expresión lógica que controla la ejecución del ciclo.                       |
| Acción(es) | Es la acción o acciones que se ejecutarán dentro del ciclo.                        |
| endwhile   | Delimita el fin del ciclo repetitivo; envía el control al inicio de la estructura. |

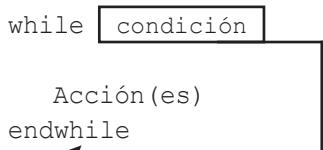
*Funcionamiento:*

Se evalúa la condición:

- a) Si se cumple, entra al ciclo, se ejecuta(n) la(s) acción(es), y al llegar al endwhile, envía el control al while, lo cual implica volver a evaluar la condición.



- b) Si no se cumple la condición, entonces se va a la siguiente acción después del endwhile, es decir, se sale del ciclo.



Al llegar al while, lo primero que se hace es evaluar la condición que controla el ciclo; si se cumple, entra al ciclo y se ejecutan las acciones especificadas dentro del mismo. Al llegar al endwhile (fin del ciclo), el control se remite al inicio de la estructura, donde se evaluará de nuevo. En caso de no cumplirse la condición, el control se traslada a la primera acción después del endwhile (fin), lo cual implica salirse del ciclo.

Por lo general la condición se establece mediante el uso de una variable que se compara con cierto valor u otra variable, debido a que la condición se encuentra al inicio del ciclo, debe hacerse una lectura adelantada o iniciar dicha variable para que tenga un valor la primera ocasión en que llega al while. Además, dentro del ciclo, debe actualizarse el valor de esa variable, por lo que es necesario incluir una lectura o asignarle un nuevo valor.

Por ejemplo, si tenemos que procesar varios empleados y no sabemos cuántos son, la estructura general quedaría:

```

Preguntar "¿Desea procesar empleado (S/N) ?"
Leer desea
while desea == 'S'
    Procesar empleado
    --
    --
Preguntar "¿Desea procesar empleado (S/N) ?"
Leer desea
endwhile

```

En la estructura anterior se utiliza la variable `desea`, que es de tipo Carácter. Para controlar el ciclo, antes del inicio del ciclo se hace la pregunta una sola vez y la lectura adelantada, a fin de que dicha variable tome un valor inicial. Después de lo anterior llega al while. Una vez procesado el empleado, para actualizar el valor de la variable, dentro del ciclo se incluyen la pregunta y lectura. Como puede observarse estas dos acciones se colocan dos veces.



Cabe aclarar que esta estructura originalmente se inventó como DOWHILE; hacer mientras se cumpla la condición. Sin embargo, el inventor del lenguaje C la implementó como while, es decir, le eliminó la palabra do. Y en virtud de que el lenguaje C es la base de C++, Java y seguramente de los lenguajes que se diseñen en el futuro, en este libro la usaremos de esta forma.



Lo primero que hace es evaluar si la condición se cumple, y por el hecho de que desde la primera vez pudiera no cumplirse, es que el while permite plantear ciclos que se repiten en un intervalo de 0 (cero) a N veces.

### 5.5.1 Simulación del do...while con while

En virtud de que con la estructura while se plantean ciclos que van en un rango de 0 hasta N veces, es que con while, es posible solucionar problemas de tipo do...while. A continuación se presenta un ejemplo que es natural para el do...while, pero resuelto con while.

Ejemplo:

Elaborar un algoritmo que permita procesar varios empleados, igual al primer ejemplo del capítulo 5 (do...while). Por cada empleado se leen los datos: Nombre del empleado, número de horas trabajadas y cuota por hora, y se imprime el nombre y sueldo.

A continuación se tiene el algoritmo de la solución:

```

Algoritmo CALCULA SUELdos DE EMPLEADOS
1. Declarar
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
        desea: Carácter
2. Preguntar "¿Desea procesar empleado (S/N) ?"
3. Leer desea
4. while desea == 'S'
    a. Solicitar Nombre, Número de horas trabajadas y
       Cuota por hora
    b. Leer nombreEmp, horasTrab, cuotaHora
    c. Calcular sueldo = horasTrab * cuotaHora
    d. Imprimir nombreEmp, sueldo
    e. Preguntar "¿Desea procesar empleado (S/N) ?"
    f. Leer desea
5. endwhile
6. Fin

```



Este es un problema en el que se procesan varios empleados y no se sabe cuántos, y como usted podrá recordar, ya lo resolvimos con el *do...while*, es decir, es un problema natural para el *do...while*. Sin embargo, aquí lo estamos resolviendo con *while* para que usted vea que puede ser resuelto con ambas estructuras, y que observe la diferencia al utilizar *while*: Antes del ciclo se pregunta "¿Desea procesar empleado (S/N)?" y luego se lee en *desea* la respuesta. Esta lectura adelantada se hace para que la primera vez que llegue al *while*, la variable contenga un valor 'S' o 'N'.



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C536.C y Programa en Java: Empleados3.java

*Explicación:*

1. Se declaran las variables que ya conocemos: nombreEmp, horasTrab, cuotaHora y sueldo, además, desea es una variable carácter que servirá para controlar al ciclo repetitivo
2. Pregunta "¿Desea procesar empleado (S/N)?"
3. Lee en *desea* la respuesta

4. Inicia ciclo while si desea == 'S' entra al ciclo
  - a. Se solicitan el nombre, número de horas trabajadas y cuota por hora
  - b. Se leen en nombreEmp, horasTrab, cuotaHora
  - c. Se calcula el sueldo
  - d. Imprime nombreEmp, sueldo
  - e. Se pregunta si “¿Desea procesar empleado (S/N)?”, pregunta a la cual se debe contestar  
S para Sí o una N para NO
  - f. Se lee en deseja, la respuesta que se dé a la pregunta anterior
5. endwhile delimita el fin del ciclo while; envía el control hacia el inicio del ciclo
6. Fin del algoritmo

### 5.5.2 Simulación del for con while

En virtud de que con la estructura while se plantean ciclos que van en un rango de 0 hasta N veces, es que con while es posible solucionar problemas de tipo for. A continuación se presenta un ejemplo que es natural para el for, pero resuelto con while y con do...while.

Ejemplo:

Elaborar un algoritmo que imprima los números del 1 al 10.

Éste es un problema natural para el for, porque se conoce cuántas veces se repetirá.

#### Solución usando for

```
Algoritmo IMPRIME 1-10
1. Declarar
    Variables
        i: Entero
2. for i=1; i<=10; i++
    a. Imprimir i
3. endfor
4. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C537.C y Programa en Java: ImprimeNumeros1.java



*Explicación:*

1. Se declara la variable i
2. Se plantea el ciclo for desde i = 1 hasta 10 con incrementos de 1; cada vez que entra al ciclo imprime el valor de i,

3. Fin del ciclo for Imprimirá: 1 2 3 4 5 6 7 8 9 10
4. Fin del algoritmo

### Solución usando while:

```

Algoritmo IMPRIME 1-10
1. Declarar
    Variables
        i: Entero
2. i = 0
3. while i < 10
    a. i = i + 1
    b. Imprimir i
4. endwhile
5. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C538.C y Programa en Java: ImprimeNumeros2.java

#### *Explicación:*

1. Se declara la variable i
2. Se inicia el contador i en 0
3. Se plantea el ciclo while pregunta si i es menor a 10 ( $i < 10$ ), si se cumple, entra al ciclo, donde:  
Incrementa i en 1  
Imprime el valor de i
4. Fin del ciclo while, que lo envía al inicio del ciclo  
Imprimirá: 1 2 3 4 5 6 7 8 9 10
5. Fin del algoritmo

En otras palabras, lo que estamos haciendo es manejando un contador, el cual se inicia en cero antes del ciclo, adentro del ciclo se incrementa en 1 y se imprime. En la condición de control del ciclo se pregunta si es menor a 10.

### Solución usando do...while:

```

Algoritmo IMPRIME 1-10
1. Declarar
    Variables
        i: Entero
2. i = 0
3. do
    a. i = i + 1
    b. Imprimir i

```

4. while i < 10
5. Fin

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C539.C y Programa en Java: ImprimeNumeros3.java



#### Explicación:

1. Se declara la variable i
2. Se inicia el contador i en 0
3. Inicia el ciclo do...while; entra al ciclo, donde:
  - a. Incrementa i en 1
  - b. Imprime el valor de i
4. Cierra el ciclo con while, pregunta si i < 10;  
si es así se regresa al do a repetir el ciclo; si no, se sale del ciclo Imprimirá:  
1 2 3 4 5 6 7 8 9 10
5. Fin del algoritmo

En otras palabras, lo que estamos haciendo es manejando un contador, el cual se inicia en cero antes del ciclo, adentro del ciclo se incrementa en 1 y se imprime. En la condición de control del ciclo se pregunta si es menor a 10.

#### Diferencia entre los tipos de repetición

Los tipos de repetición do...while, for y while se diferencian entre sí, de acuerdo al rango de repeticiones que permiten: El do...while permite un rango de repeticiones que va desde 1 hasta N veces, es decir, lo que está dentro del ciclo se deberá ejecutar al menos una vez, y mientras se cumpla la condición de ejecución del ciclo, cualquier cantidad de veces.

El for es útil para controlar ciclos en los que se conoce de antemano el número de veces que se deberán ejecutar las acciones que están dentro del ciclo. Esto es porque se controla con un contador, que toma desde un valor inicial, hasta un valor final con un incremento.

El while permite un rango de repeticiones que va desde 0 (cero) hasta N veces, porque lo primero que se hace es evaluar la condición que controla el ciclo, si ésta se cumple entra al mismo, pero si no se cumple se va a la siguiente acción después del ciclo; esto permite que, al llegar la primera vez al ciclo, si no se cumple la condición, no entra ninguna vez al ciclo y, en caso de cumplirse puede entrar una y otra vez, es decir, hasta N veces.

Así, cuando se tiene un problema que contiene repeticiones, debemos analizar el tipo de repetición que es:

- Si se conoce exactamente cuántas veces se va a repetir, es tipo for.
- Si se sabe que algo se va a repetir, no se sabe cuántas veces, pero se sabe que si va a haber al menos una ejecución, es tipo do...while.
- Si se sabe que algo se va a repetir, no se sabe cuántas veces y que puede repetirse desde 0 (cero) hasta N veces, es tipo while.

### 5.5.3 Ejercicios resueltos para la repetición while

#### Ejercicio 5.5.3.1

El sueldo que perciben los vendedores de una empresa automotriz, está integrado de la manera siguiente: El salario mínimo, más \$100.00 por cada automóvil vendido, más el 2% del valor de los automóviles vendidos.

Datos que se tienen por cada vendedor:

Nombre del Vendedor : XXXXXXXXXXXXXXXXXXXXXXXXX

    precio automóvil : 999,999.99

    precio automóvil : 999,999.99

    precio automóvil : 999,999.99

Nombre del Vendedor : XXXXXXXXXXXXXXXXXXXXXXXXX

    precio automóvil : 999,999.99

    precio automóvil : 999,999.99

    precio automóvil : 999,999.99

.....

Nombre del Vendedor : XXXXXXXXXXXXXXXXXXXXXXXXX

    precio automóvil : 999,999.99

    precio automóvil : 999,999.99

Como se puede apreciar, se tienen varios vendedores, por cada vendedor, se tiene el nombre y el precio de cada automóvil que vendió en la quincena; es posible que algunos vendedores no hayan realizado venta alguna, en tal caso, sólo se tendrá el nombre.

Elaborar un algoritmo que permita leer los datos e imprimir el siguiente reporte:

| NOMINA QUINCENAL       |            |
|------------------------|------------|
| NOMBRE                 | SUELDO     |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| ---                    | ---        |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| TOTALES 999            | 999,999.99 |

(Primero hágalo usted, después compare la solución.)

Algoritmo VENDEDORES DE AUTOMÓVILES

1. Declarar

Variables

nombreVend: Cadena

desea, otro: Carácter

totAutos, totVend: Entero

precioAuto, salMin, sueldo,

totSueldos, totVendido: Real

```
2. Solicitar el Salario mínimo
3. Leer salMin
4. Imprimir Encabezado
5. totSueldos = 0
   totVend = 0
6. do
   a. Solicitar el Nombre del vendedor
   b. Leer nombreVend
   c. totAutos = 0
      totVendido = 0
   d. Preguntar "¿Hay auto vendido (S/N) ?"
   e. Leer otro
   f. while otro == 'S'
      1. Solicitar el precio del auto
      2. Leer precioAuto
      3. totAutos = totAutos + 1
         totVendido = totVendido + precioAuto
      4. Preguntar "¿Hay otro auto vendido (S/N) ?"
      5. Leer otro
   g. endwhile
   h. sueldo = salMin+(totAutos*100)+(totVendido*0.02)
   i. Imprimir nombreVend, sueldo
   j. totVend = totVend + 1
      totSueldos = totSueldos + sueldo
   k. Preguntar "¿Hay otro vendedor (S/N) ?"
   l. Leer desea
7. while desea == 'S'
8. Imprimir totVend, totSueldos
9. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C540.C y Programa en Java: VendedoresAutos.java



*Explicación:*

1. Se declaran las variables
2. Se solicita el Salario mínimo
3. Se lee en salMin
4. Imprime encabezado
5. Inicia totSueldos en 0  
Inicia totVend en 0

6. Inicia ciclo do para procesar varios vendedores
  - a. Sigue el Nombre del vendedor
  - b. Se lee en nombreVend
  - c. Inicia totAutos en 0  
Inicia totVendido en 0
  - d. Pregunta “¿Hay auto vendido (S/N)?”
  - e. Lee en otro la respuesta
  - f. Inicia ciclo while para procesar los autos vendidos por el vendedor; compara si otro es igual a ‘S’; si es así, entra al ciclo
    1. Sigue el Precio del auto
    2. Se lee en precioAuto
    3. Incrementa totAutos en 1  
Incrementa totVendido con precioAuto
    4. Pregunta “¿Hay otro auto vendido (S/N)?”
    5. Lee en otro la respuesta
  - g. Fin del ciclo while
  - h. Calcula sueldo=salMin+(totAutos\*100)+(totVendido\*0.02)
  - i. Imprime nombreVend, sueldo
  - j. Incrementa totVend en 1  
Incrementa totSueldos con sueldo
  - k. Pregunta “¿Hay otro vendedor (S/N)?”
  - l. Lee en desea la respuesta
7. Fin ciclo (do...while) mientras desea sea igual a ‘S’ vuelve al do, si no, sale del ciclo
8. Imprime totVend, totSueldos
9. Fin del algoritmo

El primer ciclo es del tipo do...while, porque son varios vendedores, sin embargo, también podría haberse resuelto con while.

El segundo ciclo, que procesa los automóviles vendidos, sólo puede resolverse con while porque la repetición que se presenta puede ir desde 0 (cero) hasta cualquier cantidad de automóviles vendidos.

#### Ejercicio 5.5.3.2

La Comisión Nacional del Agua Delegación Sonora, lleva un registro de las lluvias que se presentan en todas las poblaciones del estado, de manera que se tienen los siguientes datos:

Población 1: X-----X

Lluvia: - - -

Lluvia: - - -

- - -

Lluvia: - - -

Población 2: X-----X

Lluvia: - - -

Lluvia: - - -

- - -

Lluvia: - - -

Población N: X-----X

Lluvia: - - -

Lluvia: - - -

- - -

Lluvia: - - -

Por cada población aparece el dato lluvia tantas veces como lluvias haya habido, este dato está en milímetros cúbicos, pudiera darse el caso de que en alguna población no traiga ningún dato de lluvia, esto quiere decir que no llovió.

Elaborar un algoritmo que lea estos datos e imprima el reporte:

| REPORTE DE LLUVIAS                     |              |
|----------------------------------------|--------------|
| POBLACIÓN                              | TOTAL LLUVIA |
| XXXXXXXXXXXXXXXXXXXXXX                 | 999.99       |
| XXXXXXXXXXXXXXXXXXXXXX                 | 999.99       |
| - - -                                  | - - -        |
| - - -                                  | - - -        |
| XXXXXXXXXXXXXXXXXXXXXX                 | 999.99       |
| TOTAL 999 POBLACIONES                  | 999.99       |
| TOTAL POBLACIONES DONDE NO LLOVIÓ: 999 |              |

Cálculos:

- TOTAL LLUVIA es el total de lluvia que se presentó en una población, se calcula sumando la cantidad de milímetros cúbicos de todas las lluvias.
- Se piden dos totales TOTAL de poblaciones procesadas y el total del total de lluvia en todas las poblaciones. También se pide la cantidad de poblaciones donde no se presentó lluvia alguna.

(Primero hágalo usted, después compare la solución.)

```
Algoritmo LLUVIAS
1. Declarar
    Variables
        poblacion: Cadena
        otro, hay: Carácter
        totPoba, totPobNoLluvia: Entero
        lluvia, totLluvia, toTotLluvia: Real
2. Imprimir encabezado
3. totPoba = 0
    totPobNoLluvia = 0
    toTotLluvia = 0
4. Preguntar "¿Hay población (S/N)?"
```

```

6. while hay == 'S'
    a. Solicitar Población
    b. Leer poblacion
    c. totLluvia = 0
    d. Preguntar "¿Hay lluvia (S/N)?""
    e. Leer otro
    f. while otro == 'S'
        1. Solicitar Lluvia en milímetros cúbicos
        2. Leer lluvia
        3. totLluvia = totLluvia + lluvia
        4. Preguntar "¿Hay otra lluvia (S/N)?""
        5. Leer otro
    g. endwhile
    h. Imprimir poblacion, totLluvia
    i. totPobla = totPobla + 1
        toTotLluvia = toTotLluvia + totLluvia
    j. IF totLluvia == 0 THEN
        1. totPobNoLluvia = totPobNoLluvia + 1
    k. ENDIF
    l. Preguntar "¿Hay otra población (S/N)?""
    m. Leer hay
    n. endwhile
    o. Imprimir totPobla, toTotLluvia, totPobNoLluvia
    p. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C541.C y Programa en Java: Lluvias.java

#### *Explicación:*

1. Se declaran las variables
2. Imprime encabezado
3. Inicia totPobla en 0; totPobNoLluvia en 0; toTotLluvia en 0
4. Pregunta "¿Hay poblacion (S/N)?"
5. Lee en hay la respuesta
6. Inicia ciclo while mientras hay es igual a 'S'; entonces entra al ciclo
  - a. Solicitud Población
  - b. Se lee en poblacion
  - c. Inicia totLluvia en 0
  - d. Pregunta "¿Hay lluvia (S/N)?"
  - e. Lee en otro la respuesta

- f. Inicia ciclo while mientras otro es igual a 'S'; entonces entra al ciclo
    1. Sigue la lluvia en milímetros cúbicos
    2. Se lee en lluvia
    3. Incrementa totLluvia con lluvia
    4. Pregunta “¿Hay otra lluvia (S/N)?”
    5. Lee en otro la respuesta
  - g. Fin del ciclo while
  - h. Imprime población, totLluvia
  - i. Incrementa totPoblación en 1
  - Incrementa toTotLluvia con totLluvia
  - j. Si totLluvia == 0 entonces
    - a. Incrementa totPobNoLluvia en 1
  - k. Fin del IF
  - l. Pregunta “¿Hay otra población (S/N)?”
  - m. Lee en hay la respuesta
7. Fin del ciclo while
8. Imprime totPoblación, toTotLluvia, totPobNoLluvia
  9. Fin del algoritmo

La tabla 5.4 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 5 de la Web del libro.

**Tabla 5.4**

| Ejercicio         | Descripción                                           |
|-------------------|-------------------------------------------------------|
| Ejercicio 5.3.3.3 | Procesa materiales requeridos para fabricar productos |
| Ejercicio 5.3.3.4 | Procesa mantenimientos de máquinas                    |
| Ejercicio 5.3.3.5 | Procesa la producción de 10 estaciones de trabajo     |
| Ejercicio 5.3.3.6 | Procesa la producción de varias estaciones de trabajo |
| Ejercicio 5.3.3.7 | Calcula valores para XYX                              |
| Ejercicio 5.3.3.8 | Obtiene pares entre 0 y 20                            |
| Ejercicio 5.3.3.9 | Determina si un número es par o impar                 |



El primer ciclo se ha resuelto con while, sin embargo, también podría resolverse con do...while; inclusive, sería más natural.

El segundo ciclo, que procesa las lluvias, sólo puede resolverse con while porque la repetición que se presenta puede ir desde 0 (cero) hasta cualquier cantidad de lluvias.



## 5.6 Ejercicios propuestos para la repetición while

1. Similar al ejercicio 5.3.3.1, sólo que ahora se tienen 20 vendedores; y al final del reporte, debe imprimirse el nombre del vendedor que tuvo el mayor total venta y del que tuvo el menor total venta. Suponiendo que no habrá iguales cantidades de total venta.
2. Similar al ejercicio 5.3.3.2, sólo que ahora se tienen 45 poblaciones; y al final del reporte, debe imprimirse el nombre de la población que tuvo el mayor total lluvia y de la que tuvo el menor total lluvia (pero habiendo llovido). Suponiendo que no habrá iguales cantidades de total lluvia.

3. Elaborar un algoritmo similar al ejercicio 5.1.2.6, con la diferencia de que ahora cada obrero trabajó varios días y puede darse el caso de no haber laborado ningún día.
4. Elaborar un algoritmo similar al ejercicio 5.1.2.6, con la diferencia de que ahora se tienen 15 obreros y cada obrero trabajó varios días y puede darse el caso de no haber laborado ningún día.
5. Elaborar un algoritmo similar al ejercicio propuesto 30 de la repetición for, con la diferencia de que ahora por cada jugador se tiene su nombre y la cantidad de puntos que anotó, por cada uno de los partidos en que anotó y puede darse el caso de que no haya anotado puntos en ningún juego.
6. Elaborar un algoritmo similar al ejercicio propuesto 30 de la repetición for, con la diferencia de que ahora se tienen 12 jugadores y por cada jugador se tiene su nombre y la cantidad de puntos que anotó, por cada uno de los partidos en que anotó y puede darse el caso de que no haya anotado puntos en ningún juego.
7. Una compañía manufacturera fabrica el producto A. Para fabricar una unidad de dicho producto se requieren los siguientes materiales:
  - Material 1: 3 unidades
  - Material 2: 4 unidades
  - Material 3: 1 unidades
  - Material 4: 2 unidades
  - Material 5: 3 unidades
  - Material 6: 2 unidades

Se tiene como datos el costo de una unidad de cada uno de los seis materiales. Elaborar un algoritmo que lea los costos de los materiales, luego que lea pedidos del producto A, en cada pedido se tiene el dato cantidad de unidades del producto A; cuando termine de leer los pedidos, imprimir:

| LISTADO DE MATERIALES REQUERIDOS |                   |                |
|----------------------------------|-------------------|----------------|
| MATERIAL                         | TOTAL DE UNIDADES | COSTO ESTIMADO |
| 1                                | 999               | 99,999.99      |
| 2                                | 999               | 99,999.99      |
| 3                                | 999               | 99,999.99      |
| 4                                | ---               | ---            |
| 5                                | ---               | ---            |
| 6                                | 999               | 99,999.99      |
| COSTO TOTAL                      |                   | 99,999.99      |

8. Similar al ejercicio anterior, sólo que ahora se fabrican el producto A, el producto B y el producto C, para fabricar el producto B se requieren los materiales:

Material 1: 2 unidades

Material 2: 5 unidades

Material 3: 2 unidades

Material 4: 1 unidades

Material 5: 2 unidades

Material 6: 4 unidades

y para fabricar el producto C se requieren los materiales:

Material 1: 7 unidades

Material 2: 1 unidades

Material 3: 5 unidades

Material 4: 4 unidades

Material 5: 2 unidades

Material 6: 3 unidades

9. Similar al ejercicio 5.3.3.4, sólo que al final se indique la máquina que tuvo la mayor cantidad de mantenimientos, y además, que se imprima cuál fue el tipo de mantenimiento que más se aplicó. Suponiendo que no habrá iguales.
10. Similar al ejercicio 5.3.3.5, sólo que al final imprima el porcentaje de estaciones de trabajo que estuvieron deficientes, el % que estuvieron bien y el % que estuvieron excelentes.
11. En la línea de producción de una compañía manufacturera se pueden presentar nueve tipos de fallas (1,2,3,4,5,6,7,8,9). Se tienen los datos de las fallas ocurridas en cada uno de los 6 días laborables de la semana: por cada día se tiene el tipo de falla, tantas veces como fallas se hayan presentado; en un día pudo no haber falla alguna. Elaborar un algoritmo que lea estos datos e imprima el siguiente reporte:

#### REPORTE SEMANAL DE FALLAS

TOTAL DE FALLAS OCURRIDAS: 999

TOTAL FALLAS TIPO 1: 999

TOTAL FALLAS TIPO 2: 999

TOTAL FALLAS TIPO 3: 999

TOTAL FALLAS TIPO 4: 999

TOTAL FALLAS TIPO 5: 999

TOTAL FALLAS TIPO 6: 999

TOTAL FALLAS TIPO 7: 999

TOTAL FALLAS TIPO 8: 999

TOTAL FALLAS TIPO 9: 999

TIPO DE FALLA QUE MÁS OCURRIÓ: 999



Como práctica, también se pueden realizar los ejercicios resueltos y los ejercicios propuestos para las repeticiones do...while y for.

## 5.7 Resumen de conceptos que debe dominar

- La estructura de repetición tipo do...while.
  - Emitir la información en forma de reporte.
  - Utilizar contadores y acumuladores, cómo obtener promedios o medias aritméticas al procesar varios elementos, a obtener el mayor y el menor de un conjunto de datos. Utilizar un ciclo repetitivo anidado dentro de otro.
- La estructura de repetición tipo for.
  - Utilizar ciclos anidados, es decir, un for dentro de otro for, un do...while dentro de un for y un for dentro de un do...while.
  - Cómo simular el for con do...while.
- La estructura de repetición tipo while.
  - Anidar ciclos repetitivos while dentro de otro while.
  - Combinar cómo anidar while con for y do...while.
  - Plantear (simular) ciclos tipo for y do...while, con while.

## 5.8 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

- 5.8.1 Resumen gráfico del capítulo**
- 5.8.2 Autoevaluación**
- 5.8.3 Programas**
- 5.8.4 Ejercicios resueltos**
- 5.8.5 Power Point para el profesor (\*)**

# 6

## Arreglos

### Contenido

- 6.1 Arreglos unidimensionales
  - 6.1.1 Ejercicios resueltos para unidimensionales
- 6.2 Arreglos bidimensionales
  - 6.2.1 Ejercicios resueltos para bidimensionales
- 6.3 Arreglos tridimensionales
  - 6.3.1 Ejercicios resueltos para tridimensionales
- 6.4 Arreglos tetradimensionales
  - 6.4.1 Ejercicios resueltos para tetradimensionales
- 6.5 Ejercicios propuestos
- 6.6 Resumen de conceptos que debe dominar
- 6.7 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 6.7.1 Resumen gráfico del capítulo
  - 6.7.2 Autoevaluación
  - 6.7.3 Programas
  - 6.7.4 Ejercicios resueltos
  - 6.7.5 Power Point para el profesor (\*)

### Objetivos del capítulo

- Estudiar la estructura de datos denominada arreglos unidimensionales, bidimensionales, tridimensionales y tetradimensionales;
- Estudiar la definición y manipulación de los elementos de los diferentes tipos de arreglos y su aplicación.

## Introducción

Con el estudio del capítulo anterior, usted ya domina la repetición y cómo diseñar algoritmos usando esa estructura de control, incluyendo el uso de todos los conceptos y estructuras de los capítulos anteriores.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando la estructura de datos denominada arreglos.

Se explica qué tipo de arreglo es un tipo de dato estructurado, formado por un conjunto de elementos de un mismo tipo de datos, y que puede almacenar más de un valor a la vez, con la condición de que todos los elementos deben ser del mismo tipo de datos, es decir, que se puede tener un arreglo de datos enteros, reales, cadenas, etcétera.

Se expone que los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así, se tienen los unidimensionales, los bidimensionales y los multidimensionales (de más de dos dimensiones); dentro de estos están los tridimensionales, tetradimensionales, etcétera. En este libro se tratan los unidimensionales, bidimensionales, tridimensionales y tetradimensionales; se abordan aspectos tales como la definición, manipulación de los elementos de los diferentes tipos de arreglos y su aplicación en pseudocódigo.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones son correctas.

En el siguiente capítulo se estudia el proceso de modularización del diseño descendente (Top down design) y su utilización en forma integrada con pseudocódigo.

El arreglo es un tipo de dato estructurado formado por un conjunto de elementos de un mismo tipo de datos. En los capítulos anteriores hemos utilizado los tipos de datos Entero, Real, Cadena, Carácter y Boolean, los cuales se consideran como datos de tipo simple, puesto que una variable que se define con alguno de estos tipos sólo puede almacenar un valor a la vez, es decir, existe una relación de uno a uno entre la variable y el número de elementos (valores) que es capaz de almacenar.

En cambio, un dato de tipo estructurado como el arreglo, puede almacenar más de un elemento (valor) a la vez, con la condición de que todos los elementos deben ser del mismo tipo de dato, es decir, que se puede tener un arreglo de datos enteros, reales, etc.

Los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así, se tienen los unidimensionales, los bidimensionales y los multidimensionales.

nales (de más de dos dimensiones); dentro de éstos están los tridimensionales, tetradiimensionales, etc. En este libro se tratarán los unidimensionales, bidimensionales, tridimensionales y tetradiimensionales.

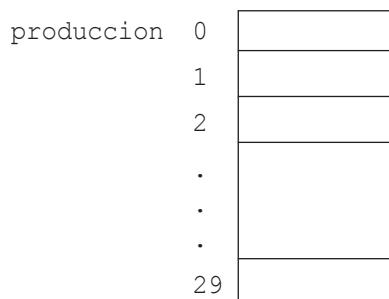
## 6.1 Arreglos unidimensionales

El arreglo unidimensional, está formado por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre y se diferencian por la posición que tiene cada elemento dentro del arreglo de datos.

Veamos el siguiente ejemplo:

Se tiene el número de unidades producidas por un obrero en cada uno de los 30 días del mes. Elaborar un algoritmo que permita leer la producción de cada uno de los 30 días, sin que se pierda la producción de ninguno de los días, esto es, se lee la producción del primer día, se lee la producción del segundo día, sin que se pierda la del primero, y así sucesivamente, al leer la producción del día 30, que no se pierda la de ninguno de los 29 días anteriores.

- Una opción sería usar 30 variables, una para cada día, de la siguiente manera:  
produccion1, produccion2, produccion3,..., produccion30
- Otra opción es usar un arreglo con una dimensión de 30 elementos, como se muestra en la siguiente figura:



*Explicación:*

En la figura tenemos un arreglo llamado `produccion` con 30 elementos, el primero de ellos se identifica con la posición 0, el segundo tiene la posición 1, el tercero la posición 2, y así sucesivamente hasta el elemento treinta que tiene la posición número 29. Así, la producción del día 1 se almacena en el elemento número 0, la producción del día 2 se almacena en el elemento 1, y así sucesivamente hasta la producción del día 30 se almacena en el elemento 29.

Sin embargo, los lenguajes C, C++, Java y derivados tienen la peculiaridad de que el primer elemento de un arreglo es el número 0 (cero), el segundo es el número 1, el tercero el 2, y así sucesivamente hasta el elemento  $N-1$ ; donde  $N$  es el número de elementos del arreglo. Por ejemplo, un arreglo de 50 elementos tendrá desde el elemento 0 hasta el elemento 49; y un arreglo de 100 elementos tendrá desde el elemento 0 hasta el elemento 99. En la metodología que se está presentando en este libro se utilizará este concepto.



Hay metodologías de la programación y lenguajes, en que los elementos de un arreglo inician con el número 1 y van hasta  $N$ ; donde  $N$  es el número de elementos del arreglo. Por ejemplo, un arreglo de 50 elementos, tendrá desde el elemento 1 hasta el elemento 50; y un arreglo de 100 elementos, tendrá desde el elemento 1 hasta el elemento 100.

### Definición del arreglo unidimensional

Cuando se define un arreglo, es necesario hacerlo como una variable. En la parte de declaraciones de variables se utiliza el siguiente formato:

```
nombreVariable: Arreglo[Tamaño] Tipo de dato
```

*En donde:*

|                |                                                                                                                                      |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------|
| nombreVariable | Es el nombre de identificación de la variable.                                                                                       |
| Arreglo        | Es la palabra reservada que indica que la variable es un arreglo.                                                                    |
| Tamaño         | Es un número entero que indica la cantidad de elementos que tendrá el arreglo, por ejemplo, 10, 20, 50, 100, 500, 1 000, etcétera.   |
| Tipo de dato   | Es el tipo de dato que tendrá el conjunto de elementos del arreglo que se está definiendo, puede ser Entero, Real, Cadena, etcétera. |

Si aplicamos los conceptos anteriores para definir un arreglo que nos sirva para almacenar la producción de los 30 días del mes, tenemos:

```
Declarar
    Variables
        produccion: Arreglo[30] Entero
```

*Explicación:*

- produccion es el nombre de la variable que se está declarando.
- Es un arreglo que contiene 30 elementos (del 0 al 29).
- Cada elemento del arreglo es un dato de tipo entero.

### Manejo de los elementos del arreglo unidimensional

Cada elemento individual de un arreglo se relaciona con el nombre de la variable y un número que indica la posición que ocupa el elemento dentro del arreglo. Dicho número se pone entre [] y se le llama subíndice, índice o suscripto. De acuerdo con lo anterior, en nuestro ejemplo tenemos que:

El elemento 1 se relaciona con produccion[0]

El elemento 2 se relaciona con produccion[1]

...

El elemento 30 se relaciona con produccion[29]

El subíndice puede ser un valor constante de tipo entero como: 0, 1, 2, ..., 29.

También puede ser una variable de tipo entero, como:

```
produccion[i]
```

O bien, puede ser una expresión algebraica que dé un resultado de tipo entero, como:

```
produccion[i+3]
produccion[(i*4)-j]
```

Como toda variable, una de tipo arreglo puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, formar parte de expresiones lógicas, etc., por ejemplo:

```
produccion[0] = 20
Leer produccion[i]
Leer produccion[10]
produccion[20] = produccion[0] + produccion[5]
Imprimir produccion[20]
```

Ejemplo:

Elaborar un algoritmo que lea la producción de un obrero en cada uno de los 30 días del mes y que lo imprima.

La lectura se podría hacer de la siguiente manera:

```
Leer produccion[0]
Leer produccion[1]
...
Leer produccion[29]
```

O bien, planteando un ciclo repetitivo se hace la lectura más eficiente:

```
for i=0; i<=29; i++
    Solicitar producción del día i+1
    Leer produccion[i]
endfor
```

*Explicación:*

Se plantea el ciclo desde que *i* tome el valor de 0 hasta 29, por cada valor de *i* entra al ciclo, donde se solicita y lee la producción del día número *i*+1, es decir, la primera vez solicita la producción del día número 1 y lo lee en el elemento *producción[i]*, es decir, en el elemento 0 del arreglo *producción*, la producción del día 2 lo lee en el elemento 1, y así, hasta llegar a leer la producción del día 30 en el elemento 29.

¿Por qué se solicita la producción del día *i*+1 y se lee en el elemento *i*? Porque se tiene la producción del día 1, del día 2, del día 3 y así hasta la producción del día 30; pero en el arreglo se tiene el elemento 0, el 1, el 2, el 3 y así hasta el elemento 29, y lo que se hace es: La producción del día 1 leerla en el elemento 0, la producción del día 2 leerla en el elemento 1 y así, hasta la producción del día 30 leerla en el elemento 29.

El algoritmo completo quedaría de la siguiente manera:

```

Algoritmo PRODUCCIÓN 30 DÍAS
1. Declarar
    Variables
        produccion: Arreglo[30] Entero
        i: Entero
2. for i=0; i<=29; i++
    a. Solicitar produccion del día i+1
    b. Leer produccion[i]
3. endfor
4. for i=0; i<=29; i++
    a. Imprimir produccion[i]
5. endfor
6. Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C601.C y Programa en Java: Produccion.java

#### *Explicación:*

1. Se declara la variable produccion como un arreglo de 30 elementos de datos de tipo entero, para almacenar en cada elemento, cada uno de los 30 días de produccion
2. Inicia ciclo for que va desde 0 hasta 29 con incrementos de 1
  - a. Solicitud la produccion del día número i+1
  - b. Lee la produccion en el elemento número i
3. Fin del ciclo for
4. Inicia ciclo for que va desde 0 hasta 29 con incrementos de 1
  - a. Imprime la produccion del día número i+1, que está en el elemento número i
5. Fin del ciclo for
6. Fin del algoritmo

#### **Forma más general de definir arreglos**

Un arreglo puede definirse de una forma más general utilizando la declaración de tipos, para definir un nuevo tipo de dato de tipo arreglo, por ejemplo:

```

Declarar
    Tipos
        DiezEnteros = Arreglo[10] Entero

```

Se ha definido un nuevo tipo de dato llamado DiezEnteros, que es un arreglo de 10 elementos de tipo Entero cada uno. Ahora, ese tipo de dato se puede utilizar en la definición de variables, por ejemplo:

```
Declarar  
    Variables  
        a: DiezEnteros  
        b: DiezEnteros  
        s: DiezEnteros
```

Otra forma de definir las variables anteriores:

```
Declarar  
    Variables  
        a, b, s: DiezEnteros
```

*en donde:*



Esto también es aplicable a los demás tipos de arreglos que se estudian en los siguientes puntos.

### 6.1.1 Ejercicios resueltos para unidimensionales

### Ejercicio 6.1.1.1

Elaborar un algoritmo que lea el nombre de un vendedor y las ventas realizadas en cada uno de los 30 días del mes, que las almacene en un arreglo y que imprima el reporte siguiente:

Nombre del vendedor: XXXXXXXXXXXXXXXXXXXXXXXXX  
Venta del día 1: 999,999.99  
Venta del día 2: 999,999.99  
.  
Venta del día 30: 999,999.99  
  
Venta total del mes: 9,999,999.99

En la Web del libro encontrará un simulador que muestra la ejecución de un programa que lleva a cabo operaciones entre vectores.

Donde la venta total del mes se calcula mediante la suma de las ventas realizadas en cada uno de los 30 días.

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo VENTAS MES
1. Declarar
    Variables
        nombreVend: Cadena
        ventas: Arreglo[30] Real
        i: Entero
        totVenta: Real
2. Solicitar nombre del vendedor
3. Leer nombreVend
4. for i=0; i<=29; i++
    a. Solicitar la venta del día i+1
```

```
b. Leer ventas[i]
5. endfor
6. totVenta = 0
7. Imprimir nombreVend
8. for i=0; i<=29; i++
    a. Imprimir ventas[i]
    b. totVenta = totVenta + ventas[i]
9. endfor
10.Imprimir totVenta
11.Fin
```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C602.C y Programa en Java: Venta.java

*Explicación:*

1. Se declaran las variables ventas como un arreglo de 30 elementos, para almacenar las ventas de cada uno de los 30 días del mes; totVenta para calcular la sumatoria de las ventas de los 30 días
2. Se solicita el nombre del vendedor
3. Se lee en nombreVend
4. Inicia ciclo for desde i=0 hasta 29 con incrementos de 1
  - a. Sigue la venta del día número i+1
  - b. Se lee en ventas[i]
5. Fin del for
6. Inicia totVenta en 0
7. Imprime nombreVend
8. Inicia ciclo for desde i=0 hasta 29 con incrementos de 1
  - a. Imprime ventas[i]
  - b. Incrementa totVenta con ventas[i]
9. Fin del for
10. Imprime totVenta
11. Fin del algoritmo

**Ejercicio 6.1.1.2**

Elaborar un algoritmo que lea los elementos de dos arreglos, cada uno con 10 números enteros. Calcular los elementos de un tercer arreglo, sumando los elementos correspondientes de los dos primeros, de la siguiente manera: Que se sume el elemento 1 del primer arreglo y el 1 del segundo y que el resultado se almacene en el 1 del tercero y así sucesivamente. Además, se requiere que al final imprima los tres arreglos de la siguiente forma:

| Arreglo 1 | + | Arreglo 2 | = | Arreglo 3 |
|-----------|---|-----------|---|-----------|
| 99        |   | 99        |   | 999       |
| 99        |   | 99        |   | 999       |
| -         |   | -         |   | -         |
| 99        |   | 99        |   | 999       |

(Primero hágalo usted, después compare la solución.)

```
Algoritmo SUMA ARREGLOS
1. Declarar
    Variables
        a, b, s: Arreglo[10] Real
        i: Entero
2. for i=0; i<=9; i++
    a. Solicitar elemento i del arreglo a
    b. Leer a[i]
    c. Solicitar elemento i del arreglo b
    d. Leer b[i]
    e. Calcular s[i] = a[i] + b[i]
3. endfor
4. Imprimir encabezado
5. for i=0; i<=9; i++
    a. Imprimir a[i], b[i], s[i]
6. endfor
7. Fin
```

En la zona de descarga de la Web del libro, están disponibles:  
 Programa en C: C603.C y Programa en Java: SumaArreglos.java



#### Explicación:

1. Se declaran las variables  
 a, b, s como arreglos de 10 elementos cada uno.  
 i : Entero
2. Inicia ciclo for desde i=0 hasta 9 con incrementos de 1
  - a. Solicitud elemento i del arreglo a
  - b. Se lee en a[i]
  - c. Solicitud elemento i del arreglo b
  - d. Se lee en b[i]
  - e. Calcula s[i]=a[i]+b[i]
3. Fin del for
4. Imprime encabezado
5. Inicia ciclo for desde i=0 hasta 9 con incrementos de 1
  - a. Imprime a[i], b[i], s[i]

6. Fin del for
7. Fin del algoritmo

#### Ejercicio 6.1.1.3

Elaborar un algoritmo que permita leer un vector de 10 números en un arreglo A de 10 elementos, lo mismo para un arreglo B; calcular e imprimir el producto de A x B. Para obtener el producto de dos vectores se multiplica el elemento 1 del vector A por el elemento 1 del vector B, el 2 de A por el 2 de B, y así sucesivamente, obteniéndose la sumatoria de los productos; el resultado no es un vector, sino un valor simple.

*(Primero hágalo usted, después compare la solución.)*

```

Algoritmo PRODUCTO DE VECTORES
1. Declarar
    Variables
        vectorA, vectorB: Arreglo[10] Entero
        r, producto: Entero
2. for r=0; r<=9; r++
    a. Solicitar vectorA[r]
    b. Leer vectorA[r]
3. endfor
4. for r=0; r<=9; r++
    a. Solicitar vectorB[r]
    b. Leer vectorB[r]
5. endfor
6. Imprimir encabezado
7. producto = 0
8. for r=0; r<=9; r++
    a. Imprimir vectorA[r], vectorB[r]
    b. producto = producto + (vectorA[r] * vectorB[r])
9. endfor
10. Imprimir producto
11. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C604.C y Programa en Java: ProductoVectores.java

#### Explicación:

1. Se declaran las variables
2. Inicia ciclo for desde r=0 hasta 9
  - a. Solicitud vectorA[r]
  - b. Se lee en vectorA[r]

3. Fin del for
4. Inicia ciclo for desde r=0 hasta 9
  - a. Sigue el elemento r del vectorB
  - b. Se lee en vectorB[r]
5. Fin del for
6. Imprime encabezado
7. Inicia producto en 0
8. Inicia ciclo for desde r=0 hasta 9
  - a. Imprime vectorA[r], vectorB[r]
  - b. Incrementa producto con (vectorA[r] \* vectorB[r])
9. Fin del for
10. Imprime producto
11. Fin del algoritmo

La tabla 6.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

**Tabla 6.1**

| Ejercicio         | Descripción                                                      |
|-------------------|------------------------------------------------------------------|
| Ejercicio 6.1.1.4 | Calcula la desviación de la media de los elementos de un arreglo |
| Ejercicio 6.1.1.5 | Realiza cálculos con la producción de un obrero                  |
| Ejercicio 6.1.1.6 | Maneja una fila de números                                       |
| Ejercicio 6.1.1.7 | Maneja 20 números en un arreglo y obtiene el mayor               |
| Ejercicio 6.1.1.8 | Maneja nombres, pesos y estaturas de personas en dos arreglos    |
| Ejercicio 6.1.1.9 | Maneja nombres, edades y sueldos de personas en dos arreglos     |



## 6.2 Arreglos bidimensionales

El arreglo bidimensional, está formado por un conjunto de elementos de un mismo tipo de dato que se almacenan bajo un mismo nombre y que al igual que en el unidimensional, se diferencian por la posición que tiene cada elemento dentro del arreglo de datos, con la aclaración de que la disposición de los elementos es en forma rectangular o cuadrada, donde la primera dimensión está dada por los renglones y la segunda por las columnas. Un arreglo de este tipo, también conocido como matriz, es de orden M x N, donde M es el número de renglones y N el número de columnas, es decir, en forma de tabla.

**Ejemplo:**

Un arreglo de orden  $4 \times 5$  tiene 4 renglones y 5 columnas, es decir, cada renglón se divide en 5 columnas, como se muestra a continuación:

| Columna   | 0 | 1 | 2 | 3 | 4 |
|-----------|---|---|---|---|---|
| Renglón 0 |   |   |   |   |   |
| Renglón 1 |   |   |   |   |   |
| Renglón 2 |   |   |   |   |   |
| Renglón 3 |   |   |   |   |   |

Para esta matriz tenemos:

|                |     |     |     |     |     |
|----------------|-----|-----|-----|-----|-----|
| Los elementos: | 0,0 | 0,1 | 0,2 | 0,3 | 0,4 |
|                | 1,0 | 1,1 | 1,2 | 1,3 | 1,4 |
|                | 2,0 | 2,1 | 2,2 | 2,3 | 2,4 |
|                | 3,0 | 3,1 | 3,2 | 3,3 | 3,4 |

Es decir, el elemento renglón 0, columna 0; el elemento renglón 0, columna 1; y así hasta el elemento renglón 3, columna 4.

**Definición del arreglo bidimensional**

Al definir un arreglo es necesario hacerlo como una variable, por lo cual en la parte de declaraciones de variables se utiliza el siguiente formato:

```
nombreArreglo: Arreglo[tamRenglones][tamColumnas] Tipo de dato
```

*En donde:*

|               |                                                                                                                                      |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------|
| nombreArreglo | Es el nombre de identificación de la variable.                                                                                       |
| Arreglo       | Es la palabra reservada que indica que la variable es un arreglo.                                                                    |
| tamRenglones  | Indica el número de renglones que tendrá el arreglo.                                                                                 |
| tamColumnas   | Indica el número de columnas que tendrá el arreglo.                                                                                  |
| Tipo de dato  | Es el tipo de dato que tiene el conjunto de elementos del arreglo que se está definiendo; pueden ser Entero, Real, Cadena, etcétera. |

Si aplicamos los conceptos del formato anterior para definir la matriz de orden  $4 \times 5$  de números enteros, tenemos:

```
Declarar
Variables
matriz: Arreglo[4][5] Entero
```

*Explicación:*

- matriz es el nombre de la variable.
- Es un arreglo que contiene 4 renglones y 5 columnas (20 elementos).
- Cada elemento del arreglo es un dato de tipo entero.

### Manejo de los elementos del arreglo bidimensional

Para relacionar cada elemento individual de una matriz se usan dos subíndices; el primero indica el renglón y el segundo la columna, como sigue:

```
matriz[renglon][columna]
```

*En donde:*

- renglon indica el número de renglón y columna indica el número de columna que ocupa el elemento relacionado.
- Los subíndices pueden ser constantes, variables o expresiones de tipo entero, como se explicó para el caso de los unidimensionales.
- Como toda variable, una de tipo matriz puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, formar parte de expresiones lógicas, etcétera.

**Ejemplos:**

```
matriz[1][1] = 20  
Leer matriz[r][c]  
Leer matriz[3][4]  
matriz[1][2] = matriz[1][2] + matriz[2][3]  
Imprimir matriz[1][2]
```

**Ejemplo:**

Elaborar un algoritmo que lea números de tipo entero para una matriz de 4 renglones por 5 columnas y además que los imprima.

Para leer los elementos de una matriz es necesario utilizar dos ciclos repetitivos anidados, el primero, y más externo, para procesar los renglones y, el segundo, que estará anidado dentro del primero, para procesar las columnas.

Si definimos:

```
Declarar  
    Variables  
        numeros: Arreglo[4][5] Entero
```

la lectura se hace así:

```
for ren=0; ren<=3; ren++  
    for col=0; col<=4; col++
```

```

    Solicitar numeros[ren][col]
    Leer numeros[ren][col]
    endfor
    endfor

```

*Explicación:*

Se utiliza un ciclo tipo for desde 0 hasta 3 renglones y, dentro del proceso de cada renglón, un ciclo de 0 a 4 para procesar las columnas. De lo anterior se tiene que: cuando ren tome el valor de 0, col tomará los valores desde 0 hasta 4, es decir, 5 veces; cuando ren tome el valor de 1, col tomará otra vez los valores desde 0 hasta 4, y así sucesivamente hasta que ren tome el valor de 3, también col tomará desde 0 hasta 4, terminando el proceso de lectura de los 20 elementos de la matriz de 4 x 5. Puede observarse que dentro del ciclo más interno se solicita el elemento ren,col, y luego se lee. Para imprimir los elementos de la matriz se hace un proceso similar, sólo que en lugar de leer se imprime.

A continuación tenemos el algoritmo completo:

```

Algoritmo MATRIZ NÚMEROS
1. Declarar
    Variables
        numeros: Arreglo[4][5] Entero
        ren, col: Entero
2. for ren=0; ren<=3; ren++
    a. for col=0; col<=4; col++
        1. Solicitar numeros[ren][col]
        2. Leer numeros[ren][col]
    b. endfor
3. endfor
4. for ren=0; ren<=3; ren++
    a. for col=0; col<=4; col++
        1. Imprimir numeros[ren][col]
    b. endfor
5. endfor
6. Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C611.C y Programa en Java: MatrizNumeros.java

*Explicación:*

1. Se declaran las variables numeros como un arreglo con 4 renglones por 5 columnas de tipo Entero  
ren, col para controlar los ciclos
2. Inicia ciclo for desde ren=0 hasta 3
  - a. Inicia ciclo for desde col=0 hasta 4

1. Solicita elemento ren,col de numeros
2. Se lee en numeros[ren][col]
- b. Fin del for
3. Fin del for
4. Inicia ciclo for desde ren=0 hasta 3
  - a. Inicia ciclo for desde col=0 hasta 4
    1. Imprime numeros[ren][col]
    - b. Fin del for
  5. Fin del for
  6. Fin del algoritmo

### 6.2.1 Ejercicios resueltos para bidimensionales

#### Ejercicio 6.2.1.1

Elaborar un algoritmo que lea números enteros para una matriz de 5 X 7, que imprima los elementos de la matriz y que al final de cada renglón imprima la suma de todos sus elementos.

*(Primero hágalo usted, después compare la solución.)*

Algoritmo SUMA POR RENGLONES

1. Declarar
 

Variables

matriz: Arreglo[5][7] Entero  
ren, col, suma: Entero
2. for ren=0; ren<=4; ren++
  - a. for col=0; col<=6; col++
    1. Solicitar matriz[ren][col]
    2. Leer matriz[ren][col]
  - b. endfor
3. endfor
4. for ren=0; ren<=4; ren++
  - a. suma = 0
  - b. for col=0; col<=6; col++
    1. Imprimir matriz[ren][col]
    2. suma = suma + matriz[ren][col]
  - c. endfor
  - d. Imprimir suma
5. endfor
6. Fin



En la Web del libro encontrará un simulador que muestra la ejecución de un programa que lleva a cabo operaciones entre matrices.

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C612.C y Programa en Java: SumaRenglones.java



*Explicación:*

1. Se declaran las variables  
matriz como arreglo de 5 renglones por 7 columnas  
suma para calcular la sumatoria por renglón
2. Inicia ciclo for desde ren=0 hasta 4
  - a. Inicia ciclo for desde col=0 hasta 6
    1. Sigue matriz[ren][col]
    2. Se lee en matriz[ren][col]
  - b. Fin del for
3. Fin del for
4. Inicia ciclo for desde ren=0 hasta 4
  - a. Inicia suma en 0
  - b. Inicia ciclo for desde col=0 hasta 6
    1. Imprime matriz[ren][col]
    2. Incrementa suma con matriz[ren][col]
  - c. Fin del for
  - d. Imprime suma
5. Fin del for
6. Fin del algoritmo

**Ejercicio 6.2.1.2**

Elaborar un algoritmo que lea números enteros para los elementos de dos matrices de 5 X 5, que calcule cada elemento de una tercera matriz sumando los elementos correspondientes de las dos anteriores. Al final imprimir las tres matrices.

*(Primero hágalo usted, después compare la solución.)*

```

Algoritmo SUMA MATRICES
1. Declarar
    Variables
        matriz1, matriz2, matriz3: Arreglo[5][5] Entero
        ren, col: Entero
2. for ren=0; ren<=4; ren++
    a. for col=0; col<=4; col++
        1. Solicitar matriz1[ren][col]
        2. Leer matriz1[ren][col]
    b. endfor
3. endfor
4. for ren=0; ren<=4; ren++
    a. for col=0; col<=4; col++
        1. Solicitar matriz2[ren][col]
        2. Leer matriz2[ren][col]
    b. endfor
  
```

```
5. endfor
6. for ren=0; ren<=4; ren++
    a. for col=0; col<=4; col++
        1. matriz3[ren][col]= matriz1[ren][col] +
            matriz2[ren][col]
    b. endfor
7. endfor
8. for ren=0; ren<=4; ren++
    a. for col=0; col<=4; col++
        1. Imprimir matriz1[ren][col]
    b. endfor
9. endfor
10. for ren=0; ren<=4; ren++
    a. for col=0; col<=4; col++
        1. Imprimir matriz2[ren][col]
    b. endfor
11. endfor
12. for ren=0; ren<=4; ren++
    a. for col=0; col<=4; col++
        1. Imprimir matriz3[ren][col]
    b. endfor
13. endfor
14. Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C613.C y Programa en Java: SumaMatrices.java



*Explicación:*

1. Se declaran las variables  
Las matrices matriz1, matriz2, matriz3 como arreglos de 5 por 5
2. Inicia ciclo for desde ren=0 hasta 4
  - a. Inicia ciclo for desde col=0 hasta 4
    1. Sigue la sentencia
    2. Se lee en matriz1[ren][col]
  - b. Fin del for
3. Fin del for
4. Inicia ciclo for desde ren=0 hasta 4
  - a. Inicia ciclo for desde col=0 hasta 4
    1. Sigue la sentencia
    2. Se lee en matriz2[ren][col]
  - b. Fin del for

5. Fin del for
6. Inicia ciclo for desde ren=0 hasta 4
  - a. Inicia ciclo for desde col=0 hasta 4
    1. Calcula matriz3[ren][col]=matriz1[ren][col]+matriz2[ren][col]
    - b. Fin del for
  7. Fin del for
  8. Inicia ciclo for desde ren=0 hasta 4
    - a. Inicia ciclo for desde col=0 hasta 4
      1. Imprime matriz1[ren][col]
      - b. Fin del for
    9. Fin del for
    10. Inicia ciclo for desde ren=0 hasta 4
      - a. Inicia ciclo for desde col=0 hasta 4
        1. Imprime matriz2[ren][col]
        - b. Fin del for
      11. Fin del for
      12. Inicia ciclo for desde ren=0 hasta 4
        - a. Inicia ciclo for desde col=0 hasta 4
          1. Imprime matriz3[ren][col]
          - b. Fin del for
        13. Fin del for
        14. Fin del algoritmo

#### Ejercicio 6.2.1.3

Se tienen los siguientes datos:

Nombre obrero 1: XXXXXXXXXXXXXXXXXXXXXXXXX

Producción mes 1: 999

Producción mes 2: 999

-

Producción mes 6: 999

Nombre obrero 2: XXXXXXXXXXXXXXXXXXXXXXXXX

Producción mes 1: 999

Producción mes 2: 999

-

Producción mes 6: 999

- - -

- - -

Nombre obrero 20: XXXXXXXXXXXXXXXXXXXXXXXXX

Producción mes 1: 999

Producción mes 2: 999

-

Producción mes 6: 999

Elaborar un algoritmo que lea estos datos en dos arreglos, uno unidimensional, para los nombres de los 20 obreros; otro, bidimensional, en el que se tendrán 20 renglones (uno para cada obrero) por 6 columnas (una para la producción de cada mes):

|    | obreros |   |   |   |   |   |
|----|---------|---|---|---|---|---|
|    | 0       | 1 | 2 | 3 | 4 | 5 |
| 0  |         |   |   |   |   |   |
| 1  |         |   |   |   |   |   |
| 2  |         |   |   |   |   |   |
| -  |         |   |   |   |   |   |
| -  |         |   |   |   |   |   |
| 19 |         |   |   |   |   |   |

Además, se requiere que imprima el reporte siguiente:

| NOMBRE DEL OBRERO | REPORTE SEMESTRAL DE PRODUCCIÓN |      |      |      |      |     | TOT.<br>MES6 | PROD. |
|-------------------|---------------------------------|------|------|------|------|-----|--------------|-------|
|                   | MES1                            | MES2 | MES3 | MES4 | MES5 |     |              |       |
| XXXXXXXXXXXXXX    | 999                             | 999  | 999  | 999  | 999  | 999 | 999          | 999   |
| XXXXXXXXXXXXXX    | 999                             | 999  | 999  | 999  | 999  | 999 | 999          | 999   |
| -                 |                                 |      |      |      |      |     |              |       |
| XXXXXXXXXXXXXX    | 999                             | 999  | 999  | 999  | 999  | 999 | 999          | 999   |
| TOTAL             |                                 |      |      |      |      |     |              |       |

(Primero hágalo usted, después compare la solución.)

```

Algoritmo PRODUCCIÓN 20 OBREROS
1. Declarar
    Variables
        obreros: Arreglo[20] Cadena
        produccion: Arreglo[20][6] Entero
        ren, col, totProd, toTotProd: Entero
    2. for ren=0; ren<=19; ren++
        a. Solicitar obreros[ren]
        b. Leer obreros[ren]
        c. for col=0; col<=5; col++
            1. Solicitar produccion[ren][col]
            2. Leer produccion[ren][col]
        d. endfor
    endfor

```

3. endfor
4. Imprimir encabezado
5. toTotProd = 0
6. for ren=0; ren<=19; ren++
  - a. Imprimir obreros[ren]
  - b. totProd = 0
  - c. for col=0; col<=5; col++
    1. Imprimir produccion[ren][col]
    2. totProd = totProd + produccion[ren][col]
  - d. endfor
  - e. Imprimir totProd
  - f. toTotProd = toTotProd + totProd
7. endfor
8. Imprimir toTotProd
9. Fin



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C614.C y Programa en Java: ObrerosMatrices.java

*Explicación:*

1. Se declaran las variables
2. Inicia ciclo for desde ren=0 hasta 19
  - a. Sigue la variable ren
  - b. Se lee en obreros[ren]
  - c. Inicia ciclo for desde col=0 hasta 5
    1. Sigue la variable produccion[ren][col]
    2. Se lee en produccion[ren][col]
  - d. Fin del for
3. Fin del for
4. Imprimir encabezado
5. Inicia toTotProd en 0
6. Inicia ciclo for desde ren=0 hasta 19
  - a. Imprime obreros[ren]
  - b. Inicia totProd en 0
  - c. Inicia ciclo for desde col=0 hasta 5
    1. Incrementa totProd con produccion[ren][col]
  - d. Fin del for
  - e. Imprime totProd
  - f. Incrementa toTotProd con totProd
7. Fin del for
8. Imprime toTotProd

### 9. Fin del algoritmo

La tabla 6.2 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

**Tabla 6.2**

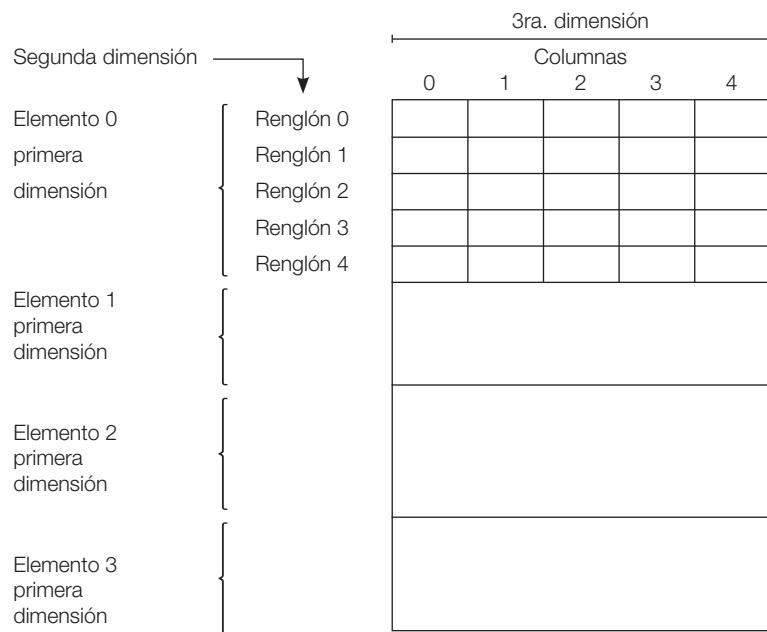
| Ejercicio          | Descripción                                          |
|--------------------|------------------------------------------------------|
| Ejercicio 6.2.1.4  | Obtiene la transpuesta de una matriz de números      |
| Ejercicio 6.2.1.5  | Indica si dos matrices de números son iguales        |
| Ejercicio 6.2.1.6  | Multiplica una matriz por vector columna             |
| Ejercicio 6.2.1.7  | Suma por renglones y columnas una matriz de números  |
| Ejercicio 6.2.1.8  | Maneja la producción de 10 artículos en 3 turnos     |
| Ejercicio 6.2.1.9  | Maneja los estados y sus capitales                   |
| Ejercicio 6.2.1.10 | Maneja una distribución de frecuencias en una matriz |
| Ejercicio 6.2.1.11 | Maneja un cuadrado mágico en una matriz              |
| Ejercicio 6.2.1.12 | Multiplica dos matrices                              |



## 6.3 Arreglos tridimensionales

El arreglo tridimensional, está formado por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre y que, al igual que en los unidimensionales y bidimensionales, se diferencian por la posición que tiene cada elemento dentro del arreglo de datos, con la aclaración de que la disposición de los elementos es una combinación del arreglo unidimensional y bidimensional. La primera dimensión se podría esquematizar como el arreglo unidimensional, un conjunto de elementos; la segunda y tercera dimensiones son un arreglo de dos dimensiones que constituye a cada elemento de la primera dimensión.

Un arreglo de tres dimensiones se podría leer como un arreglo de matrices, es decir, un arreglo compuesto por X elementos, donde cada elemento es un arreglo de MXN de dos dimensiones, esquemáticamente:



El arreglo tridimensional esquematizado, tiene 4 elementos en la primera dimensión; cada elemento de ésta, es un arreglo de 5x5, es decir, un arreglo de 2 dimensiones, se lee como un arreglo de matrices o un prisma rectangular.

Este arreglo está compuesto por los elementos:

|          |       |       |       |       |       |
|----------|-------|-------|-------|-------|-------|
| Elemento | 0,0,0 | 0,1,0 | 0,2,0 | 0,3,0 | 0,4,0 |
|          | 0,0,1 | 0,1,1 | 0,2,1 | 0,3,1 | 0,4,1 |
|          | 0,0,2 | 0,1,2 | 0,2,2 | 0,3,2 | 0,4,2 |
|          | 0,0,3 | 0,1,3 | 0,2,3 | 0,3,3 | 0,4,3 |
|          | 0,0,4 | 0,1,4 | 0,2,4 | 0,3,4 | 0,4,4 |
| Elemento | 1,0,0 | 1,1,0 | 1,2,0 | 1,3,0 | 1,4,0 |
|          | 1,0,1 | 1,1,1 | 1,2,1 | 1,3,1 | 1,4,1 |
|          | 1,0,2 | 1,1,2 | 1,2,2 | 1,3,2 | 1,4,2 |
|          | 1,0,3 | 1,1,3 | 1,2,3 | 1,3,3 | 1,4,3 |
|          | 1,0,4 | 1,1,4 | 1,2,4 | 1,3,4 | 1,4,4 |
| Elemento | 2,0,0 | 2,1,0 | 2,2,0 | 2,3,0 | 2,4,0 |
|          | 2,0,1 | 2,1,1 | 2,2,1 | 2,3,1 | 2,4,1 |
|          | 2,0,2 | 2,1,2 | 2,2,2 | 2,3,2 | 2,4,2 |
|          | 2,0,3 | 2,1,3 | 2,2,3 | 2,3,3 | 2,4,3 |
|          | 2,0,4 | 2,1,4 | 2,2,4 | 2,3,4 | 2,4,4 |

|          |       |       |       |       |       |
|----------|-------|-------|-------|-------|-------|
| Elemento | 3,0,0 | 3,1,0 | 3,2,0 | 3,3,0 | 3,4,0 |
|          | 3,0,1 | 3,1,1 | 3,2,1 | 3,3,1 | 3,4,1 |
|          | 3,0,2 | 3,1,2 | 3,2,2 | 3,3,2 | 3,4,2 |
|          | 3,0,3 | 3,1,3 | 3,2,3 | 3,3,3 | 3,4,3 |
|          | 3,0,4 | 3,1,4 | 3,2,4 | 3,3,4 | 3,4,4 |

es un arreglo de 4X5X5 el cual contiene 100 elementos.

### Definición del arreglo tridimensional

Como ya lo hemos mencionado, al definir un arreglo es necesario hacerlo como una variable, por lo cual, en la parte de declaración de variables se utiliza el siguiente formato:

```
nomVar: Arreglo[primeraDim] [segundaDim] [terceraDim] Tipo  
de dato
```

*En donde:*

- |              |                                                                      |
|--------------|----------------------------------------------------------------------|
| nomVar       | Es el nombre de la variable.                                         |
| Arreglo      | Indica que es un arreglo.                                            |
| primeraDim   | Indica la cantidad de elementos de la primera dimensión.             |
| segundaDim   | Indica la cantidad de elementos de la segunda dimensión del arreglo. |
| terceraDim   | Indica la cantidad de elementos de la tercera dimensión del arreglo. |
| Tipo de dato | Es el tipo de dato de los elementos del arreglo.                     |

*Ejemplo:*

```
numeros: Arreglo[4] [5] [5] Entero
```

numeros es un arreglo de tres dimensiones: 4 elementos de 5X5, es decir, un arreglo de 4 elementos, cada uno de los cuales es una matriz de 5X5.

### Manejo de los elementos del arreglo tridimensional

Para relacionar cada elemento individual de un arreglo de tres dimensiones se usan tres subíndices; el primero indica la primera dimensión del elemento, el segundo la segunda dimensión y el tercero la tercera dimensión, como sigue:

```
nomVar [primera] [segunda] [tercera]
```

*En donde:*

- |         |                                                       |
|---------|-------------------------------------------------------|
| primera | Indica el número de elemento en la primera dimensión. |
| segunda | Indica el número de elemento en la segunda dimensión. |
| tercera | Indica el número de elemento en la tercera dimensión. |



Los subíndices pueden ser constantes, variables o expresiones de tipo entero.

Al igual que toda variable, una de tipo arreglo tridimensional puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, etcétera.

#### Ejemplos:

```

nomVar[2][3][4] = 50
Leer nomVar[2][2][3]
Leer nomVar[1][1][3]
nomVar[1][1][4] = nomVar[2][2][3] + nomVar[1][1][3] +
nomVar[2][3][4]
Imprimir nomVar[1][1][4]

```

#### Ejemplo:

Elaborar un algoritmo que lea números de tipo entero para un arreglo como el esquematizado anteriormente, es decir, un arreglo de 4X5X5; y los imprima.

A continuación se presenta el algoritmo de la solución:

```

Algoritmo ARREGLO TRIDIMENSIONAL
1. Declarar
    Variables
        numeros: Arreglo[4][5][5] Entero
        pri, seg, ter: Entero
2. for pri=0; pri<=3; pri++
    a. for seg=0; seg<=4; seg++
        1. for ter=0; ter<=4; ter++
            a. Solicitar Elemento pri,seg,ter
            b. Leer numeros[pri][seg][ter]
        2. endfor
        b. endfor
    3. endfor
4. for pri=0; pri<=3; pri++
    a. for seg=0; seg<=4; seg++
        1. for ter=0; ter<=4; ter++
            a. Imprimir numeros[pri][seg][ter]
        2. endfor
    b. endfor
5. endfor
6. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C624.C y Programa en Java: ArregloTresDim1.java

*Explicación:*

Se utilizan tres ciclos for, el primero, y más externo, para procesar los elementos de la primera dimensión, el segundo, que está anidado dentro del primero, para procesar los renglones de la segunda dimensión, y el tercer ciclo, que está anidado dentro del segundo, para procesar las columnas. En nuestro ejemplo necesitamos un ciclo for desde 0 hasta 3 para procesar cada uno de los cuatro elementos de la primera dimensión, dentro de éste un ciclo de 0 a 4 para procesar los renglones de la segunda dimensión y, dentro del proceso de cada renglón, un ciclo de 0 a 4 para procesar las columnas (la tercera dimensión).

### 6.3.1 Ejercicios resueltos para tridimensionales

La tabla 6.3 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

**Tabla 6.3**

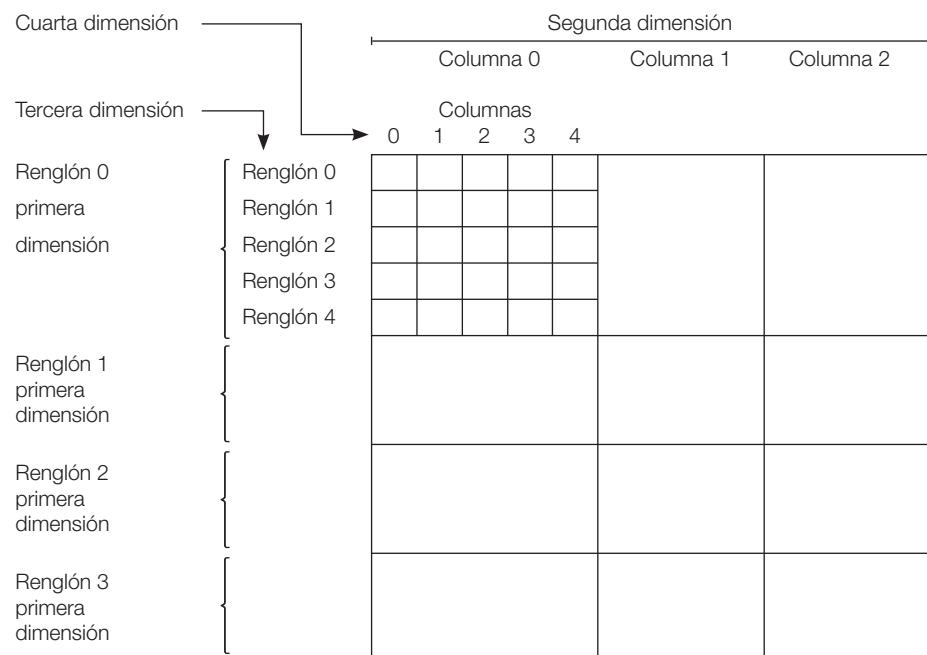
| Ejercicio         | Descripción                                                           |
|-------------------|-----------------------------------------------------------------------|
| Ejercicio 6.3.1.1 | Maneja la producción de una empresa en un arreglo de tres dimensiones |
| Ejercicio 6.3.1.2 | Similar al anterior, pero con más cálculos                            |
| Ejercicio 6.3.1.3 | Similar al anterior, pero con más cálculos                            |



## 6.4 Arreglos tetradimensionales

El arreglo tetradimensional está formado por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre y que, al igual que en los unidimensionales, bidimensionales y tridimensionales, se diferencian por la posición que tiene cada elemento dentro del arreglo de datos, con la aclaración de que la disposición de los elementos es una doble combinación del arreglo bidimensional. Las primeras dos dimensiones se podrían esquematizar como un arreglo bidimensional, y la tercera y cuarta dimensiones, esquematizan otro arreglo bidimensional dentro del primero.

Un arreglo de cuatro dimensiones se podría leer como una matriz de matrices, es decir, un arreglo de dos dimensiones MXN compuesto por elementos, donde cada elemento es un arreglo de dos dimensiones PXQ, esquemáticamente:



El arreglo tetradimensional esquematizado, tiene 4 elementos en la primera dimensión, 3 elementos en la segunda dimensión para formar un arreglo de dos dimensiones, donde cada elemento de ésta, es un arreglo de 5x5, es decir, un arreglo de 2 dimensiones (la tercera y cuarta), se lee como una matriz de matrices.

Este arreglo está compuesto por los elementos:

|          |         |         |         |         |         |
|----------|---------|---------|---------|---------|---------|
| Elemento | 0,0,0,0 | 0,0,1,0 | 0,0,2,0 | 0,0,3,0 | 0,0,4,0 |
|          | 0,0,0,1 | 0,0,1,1 | 0,0,2,1 | 0,0,3,1 | 0,0,4,1 |
|          | 0,0,0,2 | 0,0,1,2 | 0,0,2,2 | 0,0,3,2 | 0,0,4,2 |
|          | 0,0,0,3 | 0,0,1,3 | 0,0,2,3 | 0,0,3,3 | 0,0,4,3 |
|          | 0,0,0,4 | 0,0,1,4 | 0,0,2,4 | 0,0,3,4 | 0,0,4,4 |
| Elemento | 0,1,0,0 | 0,1,1,0 | 0,1,2,0 | 0,1,3,0 | 0,1,4,0 |
|          | 0,1,0,1 | 0,1,1,1 | 0,1,2,1 | 0,1,3,1 | 0,1,4,1 |
|          | 0,1,0,2 | 0,1,1,2 | 0,1,2,2 | 0,1,3,2 | 0,1,4,2 |
|          | 0,1,0,3 | 0,1,1,3 | 0,1,2,3 | 0,1,3,3 | 0,1,4,3 |
|          | 0,1,0,4 | 0,1,1,4 | 0,1,2,4 | 0,1,3,4 | 0,1,4,4 |
| Elemento | 0,2,0,0 | 0,2,1,0 | 0,2,2,0 | 0,2,3,0 | 0,2,4,0 |
|          | 0,2,0,1 | 0,2,1,1 | 0,2,2,1 | 0,2,3,1 | 0,2,4,1 |
|          | 0,2,0,2 | 0,2,1,2 | 0,2,2,2 | 0,2,3,2 | 0,2,4,2 |
|          | 0,2,0,3 | 0,2,1,3 | 0,2,2,3 | 0,2,3,3 | 0,2,4,3 |
|          | 0,2,0,4 | 0,2,1,4 | 0,2,2,4 | 0,2,3,4 | 0,2,4,4 |

|          |         |         |         |         |         |
|----------|---------|---------|---------|---------|---------|
| Elemento | 1,0,0,0 | 1,0,1,0 | 1,0,2,0 | 1,0,3,0 | 1,0,4,0 |
|          | 1,0,0,1 | 1,0,1,1 | 1,0,2,1 | 1,0,3,1 | 1,0,4,1 |
|          | 1,0,0,2 | 1,0,1,2 | 1,0,2,2 | 1,0,3,2 | 1,0,4,2 |
|          | 1,0,0,3 | 1,0,1,3 | 1,0,2,3 | 1,0,3,3 | 1,0,4,3 |
|          | 1,0,0,4 | 1,0,1,4 | 1,0,2,4 | 1,0,3,4 | 1,0,4,4 |
| Elemento | 1,1,0,0 | 1,1,1,0 | 1,1,2,0 | 1,1,3,0 | 1,1,4,0 |
|          | 1,1,0,1 | 1,1,1,1 | 1,1,2,1 | 1,1,3,1 | 1,1,4,1 |
|          | 1,1,0,2 | 1,1,1,2 | 1,1,2,2 | 1,1,3,2 | 1,1,4,2 |
|          | 1,1,0,3 | 1,1,1,3 | 1,1,2,3 | 1,1,3,3 | 1,1,4,3 |
|          | 1,1,0,4 | 1,1,1,4 | 0,1,2,4 | 1,1,3,4 | 1,1,4,4 |
| Elemento | 1,2,0,0 | 1,2,1,0 | 1,2,2,0 | 1,2,3,0 | 1,2,4,0 |
|          | 1,2,0,1 | 1,2,1,1 | 1,2,2,1 | 1,2,3,1 | 1,2,4,1 |
|          | 1,2,0,2 | 1,2,1,2 | 1,2,2,2 | 1,2,3,2 | 1,2,4,2 |
|          | 1,2,0,3 | 1,2,1,3 | 1,2,2,3 | 1,2,3,3 | 1,2,4,3 |
|          | 1,2,0,4 | 1,2,1,4 | 1,2,2,4 | 1,2,3,4 | 1,2,4,4 |
| Elemento | 2,0,0,0 | 2,0,1,0 | 2,0,2,0 | 2,0,3,0 | 2,0,4,0 |
|          | 2,0,0,1 | 2,0,1,1 | 2,0,2,1 | 2,0,3,1 | 2,0,4,1 |
|          | 2,0,0,2 | 2,0,1,2 | 2,0,2,2 | 2,0,3,2 | 2,0,4,2 |
|          | 2,0,0,3 | 2,0,1,3 | 2,0,2,3 | 2,0,3,3 | 2,0,4,3 |
|          | 2,0,0,4 | 2,0,1,4 | 2,0,2,4 | 2,0,3,4 | 2,0,4,4 |
| Elemento | 2,1,0,0 | 2,1,1,0 | 2,1,2,0 | 2,1,3,0 | 2,1,4,0 |
|          | 2,1,0,1 | 2,1,1,1 | 2,1,2,1 | 2,1,3,1 | 2,1,4,1 |
|          | 2,1,0,2 | 2,1,1,2 | 2,1,2,2 | 2,1,3,2 | 2,1,4,2 |
|          | 2,1,0,3 | 2,1,1,3 | 2,1,2,3 | 2,1,3,3 | 2,1,4,3 |
|          | 2,1,0,4 | 2,1,1,4 | 2,1,2,4 | 2,1,3,4 | 2,1,4,4 |
| Elemento | 2,2,0,0 | 2,2,1,0 | 2,2,2,0 | 2,2,3,0 | 2,2,4,0 |
|          | 2,2,0,1 | 2,2,1,1 | 2,2,2,1 | 2,2,3,1 | 2,2,4,1 |
|          | 2,2,0,2 | 2,2,1,2 | 2,2,2,2 | 2,2,3,2 | 2,2,4,2 |
|          | 2,2,0,3 | 2,2,1,3 | 2,2,2,3 | 2,2,3,3 | 2,2,4,3 |
|          | 2,2,0,4 | 2,2,1,4 | 2,2,2,4 | 2,2,3,4 | 2,2,4,4 |
| Elemento | 3,0,0,0 | 3,0,1,0 | 3,0,2,0 | 3,0,3,0 | 3,0,4,0 |
|          | 3,0,0,1 | 3,0,1,1 | 3,0,2,1 | 3,0,3,1 | 3,0,4,1 |
|          | 3,0,0,2 | 3,0,1,2 | 3,0,2,2 | 3,0,3,2 | 3,0,4,2 |
|          | 3,0,0,3 | 3,0,1,3 | 3,0,2,3 | 3,0,3,3 | 3,0,4,3 |
|          | 3,0,0,4 | 3,0,1,4 | 3,0,2,4 | 3,0,3,4 | 3,0,4,4 |

|          |         |         |         |         |         |
|----------|---------|---------|---------|---------|---------|
| Elemento | 3,1,0,0 | 3,1,1,0 | 3,1,2,0 | 3,1,3,0 | 3,1,4,0 |
|          | 3,1,0,1 | 3,1,1,1 | 3,1,2,1 | 3,1,3,1 | 3,1,4,1 |
|          | 3,1,0,2 | 3,1,1,2 | 3,1,2,2 | 3,1,3,2 | 3,1,4,2 |
|          | 3,1,0,3 | 3,1,1,3 | 3,1,2,3 | 3,1,3,3 | 3,1,4,3 |
|          | 3,1,0,4 | 3,1,1,4 | 3,1,2,4 | 3,1,3,4 | 3,1,4,4 |
| Elemento | 3,2,0,0 | 3,2,1,0 | 3,2,2,0 | 3,2,3,0 | 3,2,4,0 |
|          | 3,2,0,1 | 3,2,1,1 | 3,2,2,1 | 3,2,3,1 | 3,2,4,1 |
|          | 3,2,0,2 | 3,2,1,2 | 3,2,2,2 | 3,2,3,2 | 3,2,4,2 |
|          | 3,2,0,3 | 3,2,1,3 | 3,2,2,3 | 3,2,3,3 | 3,2,4,3 |
|          | 3,2,0,4 | 3,2,1,4 | 3,2,2,4 | 3,2,3,4 | 3,2,4,4 |

es un arreglo de 4X3X5X5 el cual contiene 300 elementos.

### Definición del arreglo tetradimensional

Como ya lo hemos mencionado, al definir un arreglo es necesario hacerlo como una variable, por lo cual, en la parte de declaración de variables se utiliza el siguiente formato:

```
nomVar: Arreglo[priDim] [segDim] [terDim] [cuarDim]
        Tipo de dato
```

*En donde:*

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| nomVar       | Es el nombre de la variable.                                                          |
| Arreglo      | Indica que es un arreglo.                                                             |
| priDim       | Indica la cantidad de elementos de la primera dimensión del arreglo (10 por ejemplo). |
| segDim       | Indica la cantidad de elementos de la segunda dimensión del arreglo.                  |
| terDim       | Indica la cantidad de elementos de la tercera dimensión del arreglo.                  |
| cuarDim      | Indica la cantidad de elementos de la cuarta dimensión del arreglo.                   |
| Tipo de dato | Es el tipo de dato de los elementos del arreglo.                                      |

*Ejemplo:*

```
numeros: Arreglo[4][3][5][5] Entero
```

numeros es un arreglo de cuatro dimensiones: 4 renglones por 3 columnas, donde cada elemento es un arreglo de 5X5; es decir, una matriz de 4X3, donde cada uno de los elementos es una matriz de 5X5.

### Manejo de los elementos del arreglo tetradimensional

Para relacionar cada elemento individual de un arreglo de cuatro dimensiones se usan cuatro subíndices; el primero indica la primera dimensión (renglón de la primera matriz) del elemento, el segundo la segunda dimensión (columna de la primera matriz), el tercero la tercera dimensión (renglón de la matriz dentro de cada elemento de la primera matriz) y el cuarto la cuarta dimensión (columna de la matriz dentro de cada elemento de la primera matriz), como sigue:

```
tetra[ren1][col1][ren2][col2]
```

*En donde:*

- ren1 Indica el número de elemento en la primera dimensión.
- col1 Indica el número de elemento en la segunda dimensión.
- ren2 Indica el número de elemento en la tercera dimensión.
- col2 Indica el número de elemento en la cuarta dimensión.



Los subíndices pueden ser constantes, variables o expresiones de tipo entero.

Al igual que toda variable, una de tipo arreglo tetradimesional puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, etcétera.

Ejemplos:

```
tetra[3][2][3][4] = 50
Leer tetra[3][2][2][3]
Leer tetra[3][1][1][3]
tetra[3][1][1][4] = tetra[3][2][2][3] +
                     tetra[3][1][1][3]+tetra[3][2][3][4]
Imprimir tetra[3][1][1][4]
```

Ejemplo:

Elaborar un algoritmo que lea números de tipo entero para un arreglo como el esquematizado anteriormente, es decir, un arreglo de 4X3X5X5, y los imprima.

```
Algoritmo ARREGLO TETRADIMENSIONAL
1. Declarar
    Variables
        numeros: Arreglo[4][3][5][5] Entero
        ren1, col1, ren2 , col2: Entero
2. for ren1=0; ren1<=3; ren1++
    a. for col1=0; col1<=2; col1++
        1. for ren2=0; ren2<=4; ren2++
            a. for col2=0; col2<=4; col2++
                1. Solicitar numeros[ren1][col1][ren2]
                    [col2]
```

```

    2. Leer numeros[ren1][col1][ren2][col2]
        b. endfor
    2. endfor
        b. endfor
    3. endfor
4. for ren1=0; ren1<=3; ren1++
    a. for col1=0; col1<=2; col1++
        1. for ren2=0; ren2<=4; ren2++
            a. for col2=0; col2<=4; col2++
                1. Imprimir numeros[ren1][col1][ren2][col2]
            b. endfor
        2. endfor
    b. endfor
5. endfor
6. Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C628.C y Programa en Java: ArregloCuatroDim1.java

#### *Explicación:*

Se utilizan cuatro ciclos for, el primero, y más externo, para procesar los elementos de la primera dimensión (renglones de la matriz más externa), el segundo, que está anidado dentro del primero, para procesar la segunda dimensión (columnas de la matriz más externa), el tercer for, para procesar la tercera dimensión (renglones de la matriz más interna), y el cuarto ciclo, que está anidado dentro del tercero, para procesar la cuarta dimensión (las columnas de la matriz más interna). En nuestro ejemplo necesitamos un ciclo for desde 0 hasta 3 para procesar cada uno de los cuatro elementos de la primera dimensión, dentro de éste un ciclo de 0 a 2 para procesar las columnas de la matriz más externa, dentro de éste un ciclo de 0 a 4 para procesar los renglones de la matriz más interna y, dentro del proceso de cada renglón de esta matriz, un ciclo de 0 a 4 para procesar las columnas (la cuarta dimensión).

#### **6.4.1 Ejercicios resueltos para tetradimensionales**

La tabla 6.4 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

**Tabla 6.4**

| Ejercicio         | Descripción                                                             |
|-------------------|-------------------------------------------------------------------------|
| Ejercicio 6.4.1.1 | Maneja la producción de una empresa en un arreglo de cuatro dimensiones |
| Ejercicio 6.4.1.2 | Similar al anterior, pero con más cálculos                              |
| Ejercicio 6.4.1.3 | Similar al anterior, pero con más cálculos                              |



## 6.5 Ejercicios propuestos

1. Elaborar un algoritmo que permita leer 20 nombres de personas en un arreglo y permita consultarlos de acuerdo con el número de posición que ocupan dentro del arreglo.
2. Elaborar un algoritmo para leer 30 días de ventas realizadas por un empleado, y que imprima el día en que tuvo la mayor y la menor venta, así como las cantidades correspondientes.
3. Elaborar un algoritmo que permita leer 30 números en un arreglo y que imprima el número mayor, el menor y la cantidad de veces que se repiten ambos.
4. Elaborar un algoritmo similar al 6.1.1.4, con la excepción de que al final debe imprimir:

TOTAL DE ELEMENTOS ARRIBA DE LA MEDIA: 999

TOTAL DE ELEMENTOS ABAJO DE LA MEDIA: 999

TOTAL DE ELEMENTOS IGUAL A LA MEDIA: 999

5. Se tienen varios obreros, por cada obrero se tienen los siguientes datos: nombre y la producción de los 30 días del mes. Elaborar un algoritmo que los lea y genere el siguiente reporte:

| REPORTE MENSUAL DE PRODUCCIÓN |           |              |                       |
|-------------------------------|-----------|--------------|-----------------------|
| NOMBRE                        | PROD. MES | PROM. DIARIO | DÍAS ARRIBA DEL PROM. |
| XXXXXXXXXXXXXXXXXXXXXX        | 999       | 999          | 999                   |
| XXXXXXXXXXXXXXXXXXXXXX        | 999       | 999          | 999                   |
| .                             | .         | .            | .                     |
| XXXXXXXXXXXXXXXXXXXXXX        | 999       | 999          | 999                   |
| TOTAL 999                     | 9999      | 9999         | 9999                  |

6. Una empresa tiene varios vendedores, por cada vendedor se tiene el nombre y la venta que realizó. Elaborar un algoritmo permita leer dichos datos y que proporcione un reporte de comisiones de ventas en el cual aparezcan todos los vendedores que tengan ventas mayores que el nivel de comisión que se calcula:

Nivel de comisión =  $3/4^*$  (promedio de ventas)

Comisión = 5% sobre el excedente de lo que vendió por arriba del nivel de comisión.

| COMISIONES DE VENDEDORES |              |            |
|--------------------------|--------------|------------|
| NOMBRE DEL VENDEDOR      | VENTAS       | COMISIÓN   |
| XXXXXXXXXXXXXXXXXXXXXX   | 999,999.99   | 99,999.99  |
| XXXXXXXXXXXXXXXXXXXXXX   | 999,999.99   | 99,999.99  |
| .                        | .            | .          |
| XXXXXXXXXXXXXXXXXXXXXX   | 999,999.99   | 99,999.99  |
| TOTAL 999 VENDEDORES     | 9,999,999.99 | 999,999.99 |

7. Elaborar un algoritmo que permita leer 15 números en un arreglo, que pregunte si se desea introducir un nuevo número en lugar de cualquiera de

los que están en el arreglo; entonces leer el número a introducir y el lugar del elemento por el que se cambiará, hacer el cambio e imprimir el arreglo antes y después del cambio.

8. Elaborar un algoritmo que permita leer 25 números en un arreglo, que imprima el arreglo y la cantidad de números que son cero, la cantidad de números que están abajo de cero y la cantidad de números arriba de cero.
9. Elaborar un algoritmo que permita leer 50 números enteros en un arreglo e imprimirlas de mayor a menor.
10. Elaborar un algoritmo que permita leer 15 nombres de personas; que los clasifique en orden descendente y los imprima.
11. Elaborar un algoritmo que lea números en una matriz de  $6 \times 6$ , que la imprima, y que coloque al final la suma por columnas.
12. Se tienen 15 estaciones de trabajo, cada una de las cuales tiene un encargado, del cual se conocen su nombre y la producción que tuvo por cada uno de los meses del año. Elaborar un algoritmo que lea los 15 nombres y los guarde en un arreglo; que haga lo mismo con los 12 meses de producción de cada una de las estaciones y que los almacene en una matriz de  $15 \times 12$ . Se requiere que imprima el siguiente reporte:

| ANÁLISIS DE PRODUCCIÓN |                  |
|------------------------|------------------|
| ESTACIÓN               | TOTAL PRODUCCIÓN |
| 99                     | 999999           |
| 99                     | 999999           |
| - - -                  |                  |
| 99                     | 999999           |
| TOTAL                  | 9999999          |

ESTACIÓN MÁS PRODUCTIVA: 99

ENCARGADO DE LA ESTACIÓN: XXXXXXXXXXXXXXXXXXXXXXXXX

CANTIDAD PRODUCIDA: 999999

13. Elaborar un algoritmo que lea el nombre de 20 trabajadores y su producción mensual por cada uno de los 12 meses del año, en dos arreglos uno para nombres y otro para producciones, en los cuales el elemento N corresponde al trabajador N. Se requiere que imprima el siguiente reporte:

| ANÁLISIS DE PRODUCCIÓN        |                  |
|-------------------------------|------------------|
| NOMBRE                        | TOTAL PRODUCCIÓN |
| XXXXXXXXXXXXXXXXXXXXXXXXXXXXX | 99999            |
| XXXXXXXXXXXXXXXXXXXXXXXXXXXXX | 99999            |
| - - -                         |                  |
| - - -                         |                  |
| XXXXXXXXXXXXXXXXXXXXXXXXXXXXX | 99999            |
| PROMEDIO DE PRODUCCIÓN:       | 99999            |

Debe imprimir sólo los que tengan producción por arriba del promedio.



14. Se tiene la producción de los 7 días de la semana de 20 plantas; elaborar un algoritmo que lea estos datos y los almacene en un arreglo de 20 renglones para las plantas, en 7 columnas, para cada día de producción; y que imprima el número de planta que tuvo la mayor producción semanal.

15. Igual al ejercicio anterior, sólo que ahora deberá imprimir el número de planta con mayor producción en un día, el día en que esto sucedió y con cuánta producción.
16. Elaborar un algoritmo que genere una matriz de  $10 \times 10$ , en la cual asigne ceros a todos los elementos, excepto a los de la diagonal principal, donde asignará unos, imprimir dicha matriz.
17. Elaborar un algoritmo que genere una matriz de  $10 \times 10$ , que asigne ceros a todos los elementos desde la diagonal principal hacia abajo; a los demás colocar unos e imprimirla.
18. Generar una matriz de  $10 \times 10$ , con ceros en los elementos desde la diagonal principal hacia arriba; a los demás colocar unos e imprimirla.
19. Elaborar un algoritmo que permita leer números en una matriz de  $6 \times 7$ , lo propio para un vector de 6 elementos. Calcular el producto de matriz por vector fila; el producto de la matriz por el vector fila, es un vector donde el elemento 1 está dado por la sumatoria de los productos del elemento 1 del vector por cada uno de los elementos de la columna 1 de la matriz, y así para el 2, 3, etc. Imprimir la matriz, el vector y el vector resultante.
20. Elaborar un algoritmo que permita leer números en una matriz de  $5 \times 6$ , que la imprima, y que imprima la sumatoria por renglones y por columnas utilizando un arreglo unidimensional para obtener la sumatoria de todos los renglones y otro arreglo de una dimensión para calcular la sumatoria de todas las columnas.
21. Una matriz es nula si todos sus elementos son iguales a cero. Elaborar un algoritmo que lea números en una matriz de  $5 \times 4$  e indique si la matriz es nula o no.
22. Elaborar un algoritmo que permita leer números para una matriz A de  $6 \times 4$ , lo propio para una matriz B, que las imprima e indique si la matriz A es mayor que la matriz B. Para que A sea mayor que B, cada elemento  $A_{ij}$  debe ser por lo menos igual que cada elemento  $B_{ij}$  correspondiente, y debe existir al menos un  $A_{ij}$  mayor que su correspondiente  $B_{ij}$ , en caso contrario A no es mayor que B.
23. Elaborar un algoritmo que lea un número, también que lea números para una matriz de  $5 \times 6$ ; calcular la matriz producto por un número, esto es, multiplicar cada elemento de la matriz por el número y colocarlo en el elemento correspondiente de la matriz resultante. Imprimir las dos matrices.
24. Una compañía manufacturera tiene 12 plantas. Elaborar un algoritmo que permita leer el nombre de cada planta y la producción que se hizo en cada uno de los siete días de la semana; utilizar un arreglo de una dimensión para leer los nombres de las plantas y un arreglo de dos dimensiones ( $12 \times 7$ ) para leer la producción de las doce plantas (uno en cada renglón) en los siete días, una columna para cada día. La idea es leer el nombre de la primera planta y luego la producción hecha en cada uno de los siete días, luego procesar la planta 2, posteriormente la 3 y así sucesivamente. Imprimir el siguiente reporte:

| PLANTA  | REPORTE SEMANAL DE PRODUCCIÓN |       |       |       |       |       |       | PROD.<br>SEMANAL |
|---------|-------------------------------|-------|-------|-------|-------|-------|-------|------------------|
|         | DÍA 1                         | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |                  |
| XXXXXX  | 999                           | 999   | 999   | 999   | 999   | 999   | 999   | 999              |
| XXXXXX  | 999                           | 999   | 999   | 999   | 999   | 999   | 999   | 999              |
| - - -   | - - -                         | - - - | - - - | - - - | - - - | - - - | - - - | - - -            |
| XXXXXX  | 999                           | 999   | 999   | 999   | 999   | 999   | 999   | 999              |
| TOTALES | 999                           | 999   | 999   | 999   | 999   | 999   | 999   | 999              |

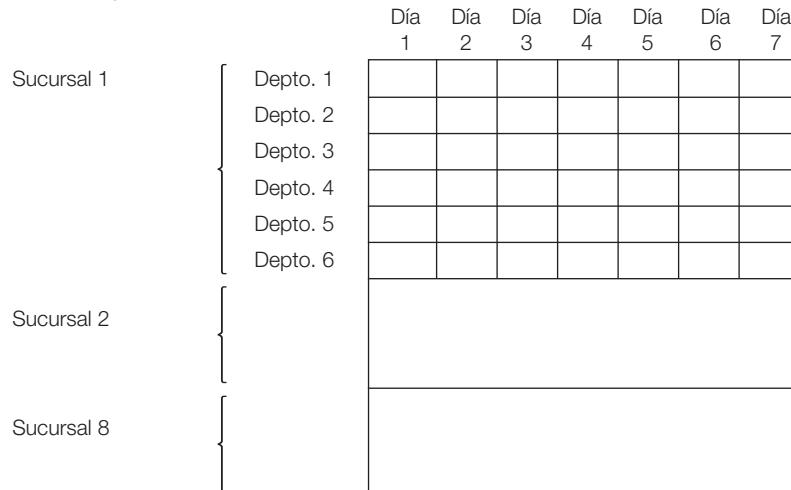
PLANTA MÁS PRODUCTIVA: XXXXXXXXXXXXXXXXXXXXXXXXX

PRODUCCIÓN DE LA PLANTA MÁS PRODUCTIVA: 999

DÍA CON MAYOR PRODUCCIÓN: 999

MAYOR PRODUCCIÓN EN UN DÍA: 999

25. Elaborar un algoritmo similar al anterior, excepto que ahora en el reporte debe agregarse un dato más por cada planta; el día más productivo, es decir, de los siete días, imprimir el número de día en el que tuvo la mayor producción.
26. Elaborar un algoritmo que permita leer números enteros para cada uno de los elementos de una matriz A de 3X4, lo mismo para una matriz B de 4X5 y calcular una matriz P de 3X5, multiplicando la matriz A por la matriz B.
27. Una compañía departamental tiene 8 sucursales, en cada sucursal tiene 6 departamentos de ventas y, por cada departamento se tiene la venta que se hizo en cada uno de los 7 días de la semana. Elaborar un algoritmo que lea los datos de las ventas en un arreglo de tres dimensiones; la primera dimensión la conforman las sucursales, la segunda dimensión los departamentos, y la tercera, los 7 días de la semana, esquemáticamente:



Una vez leídos los datos, que imprima el siguiente reporte:

**REPORTE SEMANAL DE VENTAS**

|                | SUCURSAL 1 |       |       |       |       |       |       |
|----------------|------------|-------|-------|-------|-------|-------|-------|
|                | DÍA 1      | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |
| DEPARTAMENTO-1 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-2 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-3 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-4 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-5 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-6 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |

|     | SUCURSAL 2 |       |       |       |       |       |       |
|-----|------------|-------|-------|-------|-------|-------|-------|
|     | DÍA 1      | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |
| --- | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| --- | ---        | ---   | ---   | ---   | ---   | ---   | ---   |

|                | SUCURSAL 8 |       |       |       |       |       |       |
|----------------|------------|-------|-------|-------|-------|-------|-------|
|                | DÍA 1      | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |
| DEPARTAMENTO-1 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-2 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-3 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-4 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-5 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-6 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |

28. Elaborar un algoritmo similar al anterior, pero ahora imprimir el total de ventas por departamento y por cada día para cada sucursal y el total general de ventas, esquemáticamente:

**REPORTE SEMANAL DE VENTAS**

|                | SUCURSAL 1 |       |       |       |       |       |       | TOTAL |
|----------------|------------|-------|-------|-------|-------|-------|-------|-------|
|                | DÍA 1      | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |       |
| DEPARTAMENTO-1 | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-2 | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-3 | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-4 | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-5 | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-6 | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| TOTALES        | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |

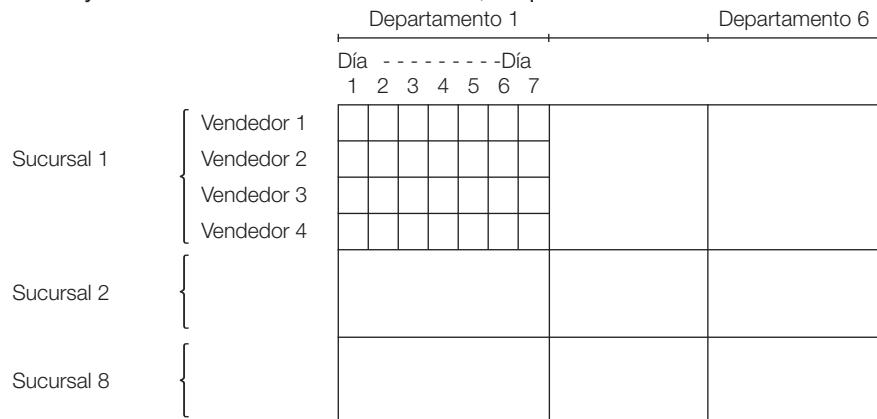
|     | SUCURSAL 2 |       |       |       |       |       |       |
|-----|------------|-------|-------|-------|-------|-------|-------|
|     | DÍA 1      | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |
| --- | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| --- | ---        | ---   | ---   | ---   | ---   | ---   | ---   |

|                | SUCURSAL 8 |       |       |       |       |       |       |
|----------------|------------|-------|-------|-------|-------|-------|-------|
|                | DÍA 1      | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |
| DEPARTAMENTO-1 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-2 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-3 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-4 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-5 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-6 | ---        | ---   | ---   | ---   | ---   | ---   | ---   |

|                         | SUCURSAL 8 |       |       |       |       |       |       |       |
|-------------------------|------------|-------|-------|-------|-------|-------|-------|-------|
|                         | DÍA 1      | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 | TOTAL |
| DEPARTAMENTO-1          | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-2          | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-3          | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-4          | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-5          | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| DEPARTAMENTO-6          | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| TOTALES                 | ---        | ---   | ---   | ---   | ---   | ---   | ---   | ---   |
| TOTAL GENERAL DE VENTAS |            |       |       |       |       |       |       | ---   |

29. Elaborar un algoritmo similar al anterior, pero además imprimir el total de ventas general por cada sucursal, el departamento que tuvo la mayor venta de cada sucursal y cuánto fue la venta; el total de ventas general, la sucursal que tuvo la mayor venta y cuánto fue ésta.
30. Una compañía departamental tiene 8 sucursales, en cada sucursal tiene 6 departamentos de ventas, por cada departamento se tienen 4 vendedores y por cada vendedor se tiene la venta que se hizo en cada uno de los 7 días de la semana. Elaborar un algoritmo que lea los datos de las ventas en un arreglo de cuatro dimensiones; la primera dimensión la conforman las sucursales, la segunda dimensión los departamentos, la tercera los vendedores y la cuarta los 7 días de la semana, esquemáticamente:



Una vez leídos los datos, que imprima el siguiente reporte:

| REPORTE SEMANAL DE VENTAS |       |       |       |       |       |       |       |
|---------------------------|-------|-------|-------|-------|-------|-------|-------|
| SUCURSAL 1                |       |       |       |       |       |       |       |
| DEPARTAMENTO 1            |       |       |       |       |       |       |       |
|                           | DÍA 1 | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |
| VENDEDOR-1                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| VENDEDOR-2                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| VENDEDOR-3                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| VENDEDOR-4                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| DEPARTAMENTO 2            |       |       |       |       |       |       |       |
| - - -                     | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| DEPARTAMENTO 6            |       |       |       |       |       |       |       |
|                           | DÍA 1 | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 |
| VENDEDOR-1                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| VENDEDOR-2                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| VENDEDOR-3                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| VENDEDOR-4                | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| SUCURSAL 8                |       |       |       |       |       |       |       |
| DEPARTAMENTO 1            |       |       |       |       |       |       |       |
| - - -                     | - - - | - - - | - - - | - - - | - - - | - - - | - - - |
| DEPARTAMENTO 6            |       |       |       |       |       |       |       |
| - - -                     | - - - | - - - | - - - | - - - | - - - | - - - | - - - |

31. Elaborar un algoritmo similar al anterior, pero que ahora imprima el siguiente reporte:

| REPORTE SEMANAL DE VENTAS   |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|-----------------------------|----------------|-------|-------|-------|-------|-------|-------|-------|--|--|--|--|--|--|--|
|                             | SUCURSAL 1     |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DEPARTAMENTO 1 |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DÍA 1          | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 | TOTAL |  |  |  |  |  |  |  |
| VENDEDOR-1                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| VENDEDOR-2                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| VENDEDOR-3                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| VENDEDOR-4                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| TOTALES                     | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DEPARTAMENTO 8 |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DÍA 1          | DÍA 2 | DÍA 3 | DÍA 4 | DÍA 5 | DÍA 6 | DÍA 7 | TOTAL |  |  |  |  |  |  |  |
| VENDEDOR-1                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| VENDEDOR-2                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| VENDEDOR-3                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| VENDEDOR-4                  | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| TOTALES                     | ---            | ---   | ---   | ---   | ---   | ---   | ---   | ---   |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | SUCURSAL 2     |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DEPARTAMENTO 1 |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DEPARTAMENTO 6 |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | SUCURSAL 8     |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DEPARTAMENTO 1 |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
|                             | DEPARTAMENTO 6 |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| TOTAL GENERAL DE PRODUCCIÓN |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |
| ---                         |                |       |       |       |       |       |       |       |  |  |  |  |  |  |  |

32. Elaborar un algoritmo similar al anterior, además, al final de cada sucursal imprimir el departamento que tuvo la mayor venta y cuánto fué ésta. Al final del reporte imprimir la venta total de toda la compañía, la sucursal que tuvo la mayor venta y cuánto fue ésta.

## 6.6 Resumen de conceptos que debe dominar

- Definición, manipulación y utilización de arreglos:
  - unidimensionales (una dimensión)
  - bidimensionales (dos dimensiones)
  - tridimensionales (tres dimensiones)
  - tetradimensionales (cuatro dimensiones)

## 6.7 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

**6.7.1 Resumen gráfico del capítulo**

**6.7.2 Autoevaluación**

**6.7.3 Programas**

**6.7.4 Ejercicios resueltos**

**6.7.5 Power Point para el profesor (\*)**

# 7

## Diseño descendente (Top Down Design)

### Contenido

- 7.1 Proceso de modularización
- 7.2 Forma de utilizar el diseño descendente con seudocódigo
- 7.3 Funciones que no regresan valor (void)
- 7.4 Variables globales, locales y parámetros
  - 7.4.1 Variables globales
  - 7.4.2 Variables locales
  - 7.4.3 Parámetros
- 7.5 Funciones estándar
  - 7.5.1 Funciones cadena de caracteres
  - 7.5.2 Validación de la entrada de datos
  - 7.5.3 Funciones especiales
- 7.6 Funciones que regresan valor
- 7.7 Ejercicios resueltos
- 7.8 Ejercicios propuestos
- 7.9 Resumen de conceptos que debe dominar
- 7.10 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 7.10.1 Resumen gráfico del capítulo
  - 7.10.2 Autoevaluación
  - 7.10.3 Programas
  - 7.10.4 Ejercicios resueltos
  - 7.10.5 Power Point para el profesor (\*)

### Objetivos del capítulo

- Estudiar el proceso de modularización del diseño descendente (Top down design) y su utilización en forma integrada con seudocódigo.
- Estudiar el uso de funciones que no regresan valor (void) y funciones que regresan valor (definidas por el usuario).
- Aprender acerca del manejo de variables globales, locales y el uso de parámetros para conectarlas.
- Estudiar algunas funciones estándar, como son: funciones para la manipulación de cadenas de caracteres y algunas funciones especiales.

## Introducción

Con el estudio del capítulo anterior, usted ya domina la estructura de datos denominada arreglos y cómo diseñar algoritmos usando esa estructura.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando el proceso de modularización del diseño descendente (Top Down Design) y su utilización en forma integrada con seudocódigo.

Se explica qué el diseño descendente es una técnica que permite diseñar la solución de un problema con base en la modularización o segmentación, dándole un enfoque de arriba hacia abajo (Top Down Design). Esta solución se divide en funciones que se estructuran e integran jerárquicamente, luego se explica cómo se diseña el algoritmo usando seudocódigo.

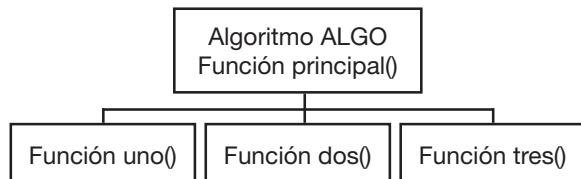
Se expone el uso de funciones que no regresan valor (`void`) y funciones que regresan valor (definidas por el usuario). También se detalla el manejo de variables globales, locales y el uso de parámetros para conectarlas. Asimismo, se estudian algunas funciones estándar, como son: funciones para la manipulación de cadenas de caracteres, validación de la entrada de datos y algunas funciones especiales.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejerzte estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones son correctas.

En el siguiente capítulo se estudian las estructuras de datos: registros y archivos.

El diseño descendente es una técnica que permite diseñar la solución de un problema con base en la modularización o segmentación, dándole un enfoque de arriba hacia abajo (Top Down Design). Esta solución se divide en funciones que se estructuran e integran jerárquicamente, como si fuera el organigrama de una empresa. Ejemplo:



En el diagrama anterior se muestra la estructura del algoritmo ALGO, que se auxilia de tres funciones subordinadas, cada una de las cuales ejecuta una tarea

específica. En su momento, la función principal() (Algoritmo ALGO) invocará o llamará a las funciones subordinadas, es decir, dirigirá su funcionamiento.

### ¿Qué es una función?

Es un módulo, método, segmento, rutina, subrutina o subalgoritmo que puede ser definido dentro de un algoritmo, con el propósito de ejecutar una tarea específica pudiendo ser llamada o invocada desde la función principal o desde otra función cuando se requiera.

### ¿Cuándo es útil la modularización?

Este enfoque de segmentación o modularización es útil en dos casos:

1. Cuando existe un grupo de instrucciones (acciones) o una tarea específica que debe ejecutarse en más de una ocasión.
2. Cuando un problema es complejo o extenso, la solución se “modulariza”, “divide” o “segmenta” en funciones que ejecutan “partes” o tareas específicas.

Dicha solución se organiza de forma similar a como lo hacen las empresas cuando se estructuran con base en las funciones para realizar sus actividades; en otras palabras, el trabajo se divide en partes que sean fácilmente manejables y que, lógicamente, puedan ser separadas; así, cada una de estas partes se dedica a ejecutar una determinada tarea, lo que redundará en una mayor concentración, entendimiento y capacidad de solución en el momento de diseñar la lógica de cada una de éstas. Dichas partes son las funciones o segmentos del algoritmo, algunas de ellas son funciones directivas o de control, que son las que se encargarán de distribuir el trabajo de las demás funciones. De esta manera se diseña un organigrama que indique la estructura general del algoritmo (programa).

En el diagrama anterior se tiene una primera función directiva, que es la función principal() del algoritmo ALGO, el cual dirige el funcionamiento de tres funciones subordinadas, que son: Función uno(), Función dos() y Función tres(); es decir, las llama o invoca cuando requiera que hagan su tarea.

## 7.1 Proceso de modularización

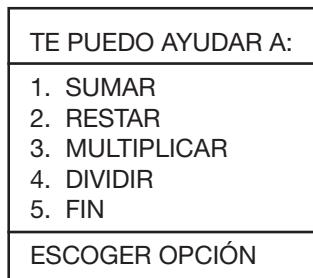
El proceso de segmentación consiste en hacer una abstracción del problema, del cual se tiene inicialmente un panorama general. Enseguida se procede a “desmembrar” o “dividir” el problema en partes pequeñas y simples, como se muestra:

1. Se forma una primera función enunciando el problema en términos de la solución a éste, y definiendo la función principal().

Ejemplo:

Elaborar un algoritmo que ayude a un niño a revisar sus tareas referentes a las operaciones aritméticas fundamentales: sumar, restar, multiplicar y dividir.

El proceso es el siguiente: Se ofrecerá un menú de opciones para escoger lo que desea hacer de acuerdo con el siguiente formato:



En caso de que el niño seleccione la opción 1, está indicando que desea revisar operaciones de sumar, enseguida se debe establecer un proceso interactivo para que el niño introduzca los dos números a sumar y su resultado, luego que la computadora le indique si la suma está correcta o incorrecta, enseguida preguntar si desea revisar otra suma, si es así, deberá repetir todo el proceso para revisar una nueva operación de sumar, algo parecido a lo siguiente:

Ayudando a revisar operaciones de sumar

Teclee primer número: 45

Teclee segundo número: 13 +

Teclee el resultado: 58

La suma está correcta

¿Desea revisar otra suma (S/N)?

Para el caso de la resta, multiplicación y división se seguirá un proceso similar, pero con las diferencias correspondientes.

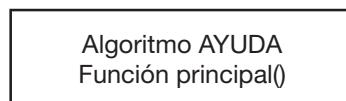
¿Qué se requiere para solucionar este problema?

El problema tiene cuatro funciones o tareas específicas:

|             |                                                                      |
|-------------|----------------------------------------------------------------------|
| sumar       | Es la parte que permite ayudar a revisar operaciones de sumar.       |
| restar      | Es la parte que permite ayudar a revisar operaciones de restar.      |
| multiplicar | Es la parte que permite ayudar a revisar operaciones de multiplicar. |
| dividir     | Es la parte que permite ayudar a revisar operaciones de dividir.     |

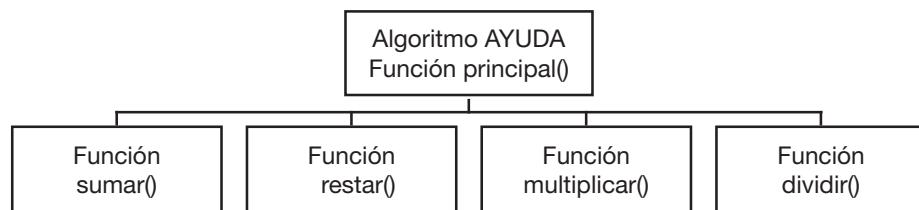
Por tanto, se requiere una función para implementar la solución de cada una de esas funciones, es decir, una función para sumar, otra función para restar, otra función para multiplicar y otra función para dividir; además de la función principal que dirigirá el funcionamiento general del algoritmo, que llamará a las funciones sumar, restar, multiplicar y dividir cuando requiera que cada una haga su tarea o función.

Aplicación: Aplicar lo enunciado en este punto; y, si se considera que se trata de un algoritmo que ayuda a un niño en la revisión de sus tareas tenemos la función principal() siguiente:



2. Se toma esta función y se busca la forma de dividirla en otras funciones más pequeñas, que ejecuten tareas o funciones específicas. Las mismas funciones que se desea que ejecute el algoritmo, nos darán la pauta para definir las funciones, y así hacer una segmentación de la solución del problema en partes más manejables.

Aplicación: Si nos referimos al ejemplo anterior, tenemos que se deben llevar a cabo cuatro tareas o funciones claramente definidas: sumar, restar, multiplicar y dividir, de ahí que necesitamos una función para cada una de dichas tareas, las cuales deberán estar subordinadas a la función principal(). La estructura que se tiene, entonces, es la siguiente:

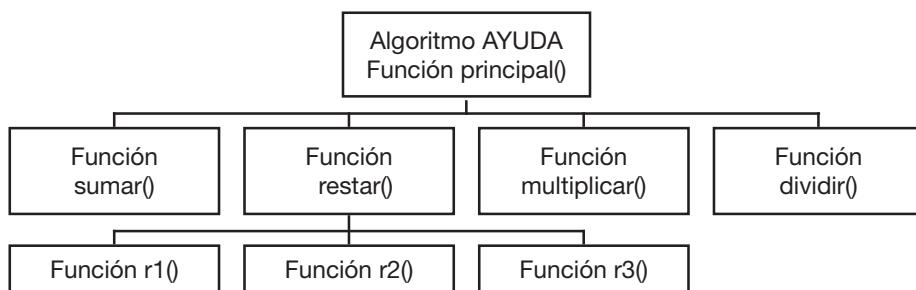


Las funciones tienen un orden, de arriba hacia abajo y de izquierda a derecha.

3. Se repite el paso 2 para cada nueva función definida, hasta llegar a un nivel de detalle adecuado, es decir, hasta hacer que cada función ejecute una tarea específica, que esté claramente definida y que el diseño y la codificación de la lógica de la misma, resulte fácil.

Aplicación: En el problema que estamos resolviendo se revisan las funciones sumar(), restar(), multiplicar() y dividir(). Se considera que estas funciones tienen un nivel de detalle adecuado, porque cada función hace una tarea muy simple, clara y específica y, en consecuencia, se pueden diseñar fácilmente.

En caso de no ser así, es decir, que la función restar(), por ejemplo, sea una función grande o compleja, entonces requerirá otras funciones subordinadas, en este caso, la estructura general del algoritmo es la siguiente:



Como se puede ver, cuando llega a la función `restar()`, ésta tiene a su vez subordinadas las funciones `r1()`, `r2()` y `r3()`, a las cuales deberá ir, antes de ir a otras funciones de su mismo nivel. En este caso, la función `restar()` se convierte a su vez, en una función directiva que controla la ejecución de sus funciones subordinadas.

## 7.2 Forma de utilizar el diseño descendente con seudocódigo

Cuando el enfoque de diseño descendente se utiliza de manera conjunta con la técnica seudocódigo, se logra una herramienta de diseño de algoritmos (programas) muy poderosa; a continuación se explica el procedimiento para hacerlo.

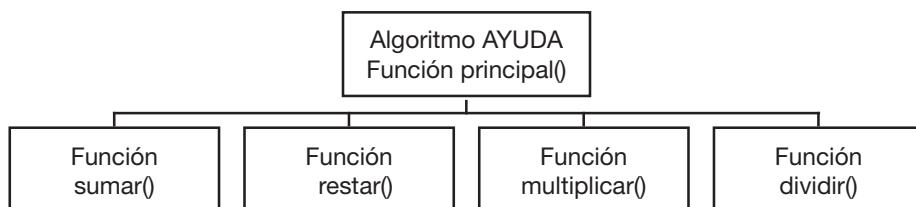
1. Se tiene un problema ya analizado.
2. Se diseña la estructura general del algoritmo utilizando el diseño descendente, es decir, se definen las funciones que integrarán el algoritmo.
3. Se toma cada función y se diseña su lógica utilizando seudocódigo.

Cómo llamar funciones en seudocódigo

`nombreFuncion()`

Significa que está llamando a la función `nombreFuncion()`, es decir, que con poner el nombre de la función significa que la está llamando o invocando.

Ejemplo: A continuación se presenta el diseño de la solución completa del problema relacionado con el diseño del algoritmo que ayuda al niño en la revisión de sus tareas de sumar, restar, multiplicar y dividir. La estructura general del algoritmo es:



El siguiente paso es diseñar la lógica de cada función utilizando seudocódigo. En el siguiente punto se explica cómo hacerlo.

### 7.3 Funciones que no regresan valor (void)

En este punto se explica el formato general y la manera de cómo se diseña un algoritmo en seudocódigo para implementar las funciones definidas en el diagrama que se diseñó con los conceptos explicados en los dos puntos anteriores. Formato de un algoritmo que utiliza funciones que no regresan valor (void):

```
Algoritmo ALGO
1. Declarar
    Variables
        nomVar1: Tipo de dato
        nomVar2: Tipo de dato
        nomVar3: Tipo de dato
2. Función principal()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Función principal
3. Función uno()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Función uno
4. Función dos()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Función dos
5. Función tres()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Función tres
Fin
```

*En donde:*

Se tiene el esquema de un algoritmo que tiene:

El paso 1, el cual es opcional, es decir, puede o no estar presente. En él

se declaran variables globales, que se pueden usar en todas las funciones del algoritmo. Si no hay que declarar nada global, entonces, el paso 1 es la función principal(), y los demás pasos se recorren.

Enseguida se tienen cuatro funciones: La función principal(), la función uno(), la función dos() y la función tres().

Las funciones uno(), dos() y tres(), cada una tiene la lógica necesaria para implementar una tarea específica para resolver el problema. Sin embargo, en la función principal() es donde inicia el funcionamiento del algoritmo, y es la que tiene la lógica general que resuelve el problema, y en su momento, deberá llamar a cada una de los otras funciones, es decir, a la función uno(), dos() y tres(). Por último se indica el fin del algoritmo.

Ejemplo:

A continuación se presenta el algoritmo de la solución, correspondiente al diagrama que en los dos puntos anteriores explicamos:

```

Algoritmo AYUDA
1. Declarar
    Variables
        num1, num2, resuNi, resuMaq: Entero
        desea: Carácter
        opcion: Entero
2. Función principal()
    a. do
        1. Imprimir el menú de opciones

```

|                                                                 |
|-----------------------------------------------------------------|
| TE PUEDO AYUDAR A:                                              |
| 1. SUMAR<br>2. RESTAR<br>3. MULTIPLICAR<br>4. DIVIDIR<br>5. FIN |
| ESCOGER OPCIÓN                                                  |

```

    2. Leer opcion
    3. switch opcion
        1: ayudaSumar()
        2: ayudaRestar()
        3: ayudaMultiplicar()
        4: ayudaDividir()
    4. endswitch
    b. while opcion != 5
    c. Fin Función principal

```

```
3. Función ayudaSumar()
    a. do
        1. Solicitar número uno, número dos y resultado
        2. Leer num1, num2, resuNi
        3. Calcular resuMaq = num1 + num2
        4. if resuMaq == resuNi then
            a. Imprimir "La suma está correcta"
        5. else
            a. Imprimir "La suma está incorrecta"
        6. endif
        7. Preguntar "¿Desea revisar otra suma (S/N) ?"
        8. Leer desea
    b. while desea == 'S'
    c. Fin Función ayudaSumar
4. Función ayudaRestar()
    a. do
        1. Solicitar número uno, número dos y resultado
        2. Leer num1, num2, resuNi
        3. Calcular resuMaq = num1 - num2
        4. if resuMaq == resuNi then
            a. Imprimir "La resta está correcta"
        5. else
            a. Imprimir "La resta está incorrecta"
        6. endif
        7. Preguntar "¿Desea revisar otra resta (S/N) ?"
        8. Leer desea
    b. while desea == 'S'
    c. Fin Función ayudaRestar
5. Función ayudaMultiplicar()
    a. do
        1. Solicitar número uno, número dos y resultado
        2. Leer num1, num2, resuNi
        3. Calcular resuMaq = num1 * num2
        4. if resuMaq == resuNi then
            a. Imprimir "La multiplicación está correcta"
        5. else
            a. Imprimir "La multiplicación está
                incorrecta"
        6. endif
        7. Preguntar "¿Desea revisar otra
                multiplicación (S/N) ?"
        8. Leer desea
    b. while desea == 'S'
    c. Fin Función ayudaMultiplicar
```

```

6. Función ayudaDividir()
a. do
    1. Solicitar número uno, número dos y resultado
    2. Leer num1, num2, resuNi
    3. Calcular resuMaq = num1 / num2
    4. if resuMaq == resuNi then
        a. Imprimir "La división está correcta"
    5. else
        a. Imprimir "La división está incorrecta"
    6. endif
    7. Preguntar "¿Desea revisar otra
       división (S/N) ?"
    8. Leer desea
    b. while desea == 'S'
    c. Fin Función ayudaDividir
Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C701.C y Programa en Java: Ayuda.java

*Explicación:*

Algoritmo AYUDA

*Es el encabezado del algoritmo*

1. Se declaran variables globales  
num1, num2, resuNi, resuMaq: Entero  
desea : Carácter  
opcion : Entero

*En esta parte se hacen las declaraciones globales del algoritmo, se pueden declarar tipos, constantes y variables; y se podrán utilizar en cualquier función del algoritmo.*

## 2. Función principal()

- a. Inicia ciclo do
  1. Imprimir el menú de opciones donde se solicita la opción
  2. Se lee en opcion la respuesta
  3. Inicia switch opcion
    - Si opcion es 1, entonces : Llama ayudaSumar
    - Si opcion es 2, entonces : Llama ayudaRestar
    - Si opcion es 3, entonces : Llama ayudaMultiplicar
    - Si opcion es 4, entonces : Llama ayudaDividir
  4. Fin del switch

- b. Fin ciclo while opcion != 5 va al do
- c. Fin Función

*Todo algoritmo diseñado con funciones, inicia su funcionamiento en la función principal(), que será la que dirigirá la operación del resto de las funciones, es decir, las llamará para que realicen la tarea que les corresponde.*

### 3. Función ayudaSumar()

- a. Inicia ciclo do
  - 1. Sigue el número uno, número dos y resultado del niño
  - 2. Lee en num1, num2, resuNi
  - 3. Calcula resultado de la máquina
  - 4. Si resuMaq == resuNi Entonces
    - a. Imprime "La suma está correcta"
  - 5. Si no
    - a. Imprime "La suma está incorrecta"
  - 6. Fin if
  - 7. Pregunta "¿Desea revisar otra suma (S/N)?"
  - 8. Lee en desea
- b. Fin ciclo while desea == 'S'; Regresa al do
- c. Fin Función

*En esta función se ayuda a revisar operaciones de sumar.*

### 4. Función ayudaRestar()

- a. Inicia ciclo do
  - 1. Sigue el número uno, número dos y resultado del niño
  - 2. Lee en num1, num2, resuNi
  - 3. Calcula resultado de la máquina
  - 4. Si resuMaq == resuNi Entonces
    - a. Imprime "La resta está correcta"
  - 5. Si no
    - a. Imprime "La resta está incorrecta"
  - 6. Fin del if
  - 7. Pregunta "¿Desea revisar otra resta (S/N)?"
  - 8. Lee en desea la respuesta
- b. Fin ciclo while desea == 'S'; Regresa al do
- c. Fin Función

*En esta función se ayuda a revisar operaciones de restar.*

5. Función ayudaMultiplicar()

- a. Inicia ciclo do
  1. Sigue la multiplicación
  2. Lee en num1, num2, resuNi
  3. Calcula resultado de la máquina
  4. Si resuMaq == resuNi Entonces
    - a. Imprime "La multiplicación está correcta"
  5. Si no
    - a. Imprime "La multiplicación está incorrecta"
  6. Fin del if
  7. Pregunta "¿Desea revisar otra multiplicación (S/N)?"
  8. Lee en desea
- b. Fin ciclo while desea == 'S'; Regresa al do
- c. Fin Función

*En esta función se ayuda a revisar operaciones de multiplicar.*

6. Función ayudaDividir()

- a. Inicia ciclo do
  1. Sigue la división
  2. Lee en num1, num2, resuNi
  3. Calcula resultado de la máquina
  4. Si resuMaq == resuNi Entonces
    - a. Imprime "La división está correcta"
  5. Si no
    - a. Imprime "La división está incorrecta"
  6. Fin del if
  7. Pregunta "¿Desea revisar otra división (S/N)?"
  8. Lee en desea
- b. Fin ciclo while desea == 'S'; Regresa al do
- c. Fin Función

*En esta función se ayuda a revisar operaciones de dividir.*

Por último se tiene el fin del algoritmo.



Cada función del algoritmo puede tener su parte de declaraciones: Tipos, constantes y variables, la cual es opcional, y lo que ahí se declare, sólo se puede utilizar en la función correspondiente, es decir, se les conoce como locales a la función donde se definen. En el siguiente punto se explican.

## 7.4 Variables globales, locales y parámetros

### 7.4.1 Variables globales

Cuando un algoritmo utiliza funciones, se pueden declarar variables tanto en el contexto global del algoritmo, como de manera local en cada función. A las variables definidas en el contexto global se les llama variables globales, mismas que pueden utilizarse en cualquier función del algoritmo. A continuación se presenta un algoritmo que muestra cómo usar una variable global:

```
Algoritmo VARIABLE GLOBAL
1. Declarar
    Variables
        x: Entero
2. Función principal()
    a. x = 0
    b. cambiar()
    c. Imprimir x
    d. Fin Función principal
3. Función cambiar()
    a. x = 1
    b. Fin Función cambiar
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C702.C y Programa en Java: VarGlobal.java



#### Explicación:

Este algoritmo VARIABLE GLOBAL tiene la función principal() y una función subordinada llamada cambiar(). Se utiliza una variable global, la cual se define en la parte de declaraciones globales, a continuación, en la función principal(), se le asigna el valor cero a dicha variable ( $x=0$ ); luego se llama o invoca la función cambiar(); la cual también utiliza la variable x, asignándole el valor 1. Al regresar a la función principal(), se imprime la variable x cuyo valor es 1.

Esto significa que se utiliza una sola variable global, a la cual se le asigna el valor cero en la función principal() y después se le asigna 1 en la función cambiar(); al regresar a la función principal() se imprime x, con el valor actual 1.



El algoritmo que elaboramos en el punto anterior (AYUDA a sumar, restar, multiplicar y dividir), utiliza este concepto de variable global.

### 7.4.2 Variables locales

Las variables locales son las que se definen en cada función, mismas que sólo pueden utilizarse en el contexto de la función en que fueron definidas. A continuación se presenta un algoritmo que muestra cómo usar una variable local:

```
Algoritmo VARIABLE LOCAL
1. Declarar
    Variables
    x: Entero
2. Función principal()
    a. x = 0
    b. cambiar()
    c. Imprimir x
    d. Fin Función principal
3. Función cambiar()
    a. Declarar
        Variables
        x: Entero
    b. x = 1
    c. Fin Función cambiar
Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C703.C y Programa en Java: VarLocal.java

#### *Explicación:*

Este algoritmo VARIABLE LOCAL tiene la función principal() y una función subordinada llamada cambiar(). Se utiliza una variable global (x), definida en la parte de declaraciones globales; a continuación, en la función principal(), se le asigna el valor cero, luego se invoca a la función cambiar() en la cual se define una variable local con el nombre de x, a la cual se le asigna el valor 1. Al regresar a la función principal(), se imprime la variable x cuyo valor es cero (0). Revise el algoritmo y vea si esto es cierto. ¿Ya lo hizo? Si es cierto, ¿por qué? Respuesta: Aunque las dos variables tienen el mismo nombre, son diferentes, ya que una se definió como global y la otra como local. Aun cuando la local definida en la función cambiar(), se llame igual que la global, para esta función tiene mayor jerarquía la local, es decir, oculta a la global porque tiene el mismo nombre. En otras palabras, en el algoritmo se define una variable global (x), a la que se le asigna el valor 0 en la función principal(), luego se llama a la función cambiar(), donde se define una variable local (x), y se le asigna el valor 1. Al regresar a la función principal() se imprime x, ¿cuál x?, obviamente la global, porque es la única de las dos x que se puede usar en la función principal(), que tiene el valor de cero (0).

**Ejercicio 7.4.2.1**

Elaborar un algoritmo que ofrezca un menú de opciones, mediante el cual se pueda escoger calcular el área de las figuras geométricas: triángulo, cuadrado, rectángulo y círculo. Una vez seleccionada la opción, que llame a una función que permita solicitar, leer los datos necesarios, hacer el cálculo correspondiente e imprimirlo.

$$\text{Área del triángulo} = \frac{\text{BASE} \times \text{ALTURA}}{2}$$

$$\text{Área del cuadrado} = \text{LADO}^2$$

$$\text{Área del círculo} = \pi r^2$$

$$\text{Área del rectángulo} = \text{BASE} \times \text{ALTURA}$$

Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

| ÁREAS FIGURAS GEOMÉTRICAS                                            |
|----------------------------------------------------------------------|
| 1. TRIÁNGULO<br>2. CUADRADO<br>3. RECTÁNGULO<br>4. CÍRCULO<br>5. FIN |
| <b>ESCOGER OPCIÓN</b>                                                |

Usar variables locales.

A continuación se tiene el algoritmo de la solución:

(Primero hágalo usted, después compare la solución.)

```

Algoritmo ÁREAS
1. Función principal()
   a. Declarar
      Variables
      opcion: Entero
   b. do
      1. Imprimir MENÚ

```

| ÁREAS FIGURAS GEOMÉTRICAS                                            |
|----------------------------------------------------------------------|
| 1. TRIÁNGULO<br>2. CUADRADO<br>3. RECTÁNGULO<br>4. CÍRCULO<br>5. FIN |
| ESCOGER OPCIÓN                                                       |

```

2. Leer opcion
3. switch opcion
  1: calcularAreaTriangulo()
  2: calcularAreaCuadrado()
  3: calcularAreaRectangulo()
  4: calcularAreaCirculo()
4. endswitch
c. while opcion != 5
d. Fin Función principal
2. Función calcularAreaTriangulo()
a. Declarar
  Variables
    base, altura, areaTria: Real
b. Solicitar Base, Altura
c. Leer base, altura
d. areaTria = (base * altura)/2
e. Imprimir areaTria
f. Fin Función calcularAreaTriangulo
3. Función calcularAreaCuadrado()
a. Declarar
  Variables
    lado, areaCuad: Real
b. Solicitar Lado
c. Leer lado
d. areaCuad = Potencia(lado,2)
e. Imprimir areaCuad
f. Fin Función calcularAreaCuadrado
4. Función calcularAreaRectangulo()
a. Declarar
  Variables
    areaRec, base, altura: Real
b. Solicitar Base, Altura
c. Leer base, altura
d. areaRec = base * altura

```

```
e. Imprimir areaRec
f. Fin Función calcularAreaRectangulo
5. Función calcularAreaCirculo()
a. Declarar
    Constantes
        PI = 3.14159265
    Variables
        areaCirc, radio: Real
b. Solicitar Radio
c. Leer radio
d. areaCirc = PI * Potencia(radio,2)
e. Imprimir areaCirc
f. Fin Función calcularAreaCirculo
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C704.C y Programa en Java: AreasFigGeometricas.java



*Explicación:*

El algoritmo tiene cinco funciones: Una es la función principal(); y las otras funciones, calculan el área de un triángulo, cuadrado, rectángulo y círculo, respectivamente:

1. Inicia la función principal()
  - a. Se declara la variable opcion para leer la opción que escoja
  - b. Inicia ciclo do
    1. Imprimir MENÚ y solicita la opción
    2. Lee en opcion
    3. switch opcion
      - Si opcion == 1: Llama Función calcularAreaTriangulo()
      - Si opcion == 2: Llama Función calcularAreaCuadrado()
      - Si opcion == 3: Llama Función calcularAreaRectangulo()
      - Si opcion == 4: Llama Función calcularAreaCirculo()
    4. Fin del switch
  - c. Fin ciclo while Si opcion != 5 vuelve al do
  - d. Termina la función
  2. Función calcularAreaTriangulo()
    - a. Se declaran las variables
    - b. Solicita Base, Altura
    - c. Lee base, altura
    - d. Calcula areaTria = (base \* altura)/2
    - e. Imprime areaTria
    - f. Fin Función

3. Función calcularAreaCuadrado()
  - a. Se declaran las variables
  - b. Solicita Lado
  - c. Lee lado
  - d. Calcula areaCuad = Potencia(lado,2)
  - e. Imprime areaCuad
  - f. Fin Función
4. Función calcularAreaRectangulo()
  - a. Se declaran las variables
  - b. Solicita Base, Altura
  - c. Lee base, altura
  - d. Calcula areaRec = base \* altura
  - e. Imprime areaRec
  - f. Fin Función
5. Función calcularAreaCirculo()
  - a. Se declaran la constante y las variables
  - b. Solicita Radio
  - c. Lee radio
  - d. Calcula areaCirc = PI \* Potencia(radio,2)
  - e. Imprime areaCirc
  - f. Fin Función

Fin del algoritmo

Como podrá observar, no se utilizaron variables globales, por ello el paso 1 es la función principal(), y en cada una de las funciones se declararon las variables requeridas en dicha función. Recordemos que las variables declaradas en una función, no pueden utilizarse en otras funciones.



### 7.4.3 Parámetros

Si analizamos los conceptos de variables globales y locales, inferimos que cuando se usan variables locales, éstas son independientes de las globales y de las de otras funciones. En ocasiones puede ser necesario conectar una variable global con una local para transmitir datos entre ambas, o bien, conectar variables locales de una función con variables locales de otra(s) función(es); esto es posible mediante el uso de parámetros, en donde las variables fungen como tales. Existen parámetros por referencia y parámetros por valor.

#### Parámetro por referencia

El parámetro por referencia (o parámetro variable, en algunos lenguajes), es una variable local de una función que se define como parámetro en el encabezado de la misma, y sirve para conectarse con otra variable de otra función, mediante el envío de su dirección, es decir, se conecta con la otra variable a través de su contenido; al llamarse o invocarse la función se establece la conexión, convirtiéndose en sinónimos, esto significa que lo que le suceda a la variable local de la función llamada, le sucederá a la variable de la función con la que fue conectada al hacer la llamada, porque utilizan la misma posición de memoria. A continuación se presenta un algoritmo que muestra el uso de un parámetro por referencia.

```

Algoritmo PARÁMETRO POR REFERENCIA
1. Función principal()
   a. Declarar
      Variables
      x: Entero
   b. x = 0
   c. Imprimir x
   d. cambiar(x)
   e. Imprimir x
   f. Fin Función principal
2. Función cambiar(Ref y: Entero)
   a. y = 1
   b. Fin Función cambiar
Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C705.C y Programa en Java: ParametroPorReferencia.java



#### *Explicación:*

En la función principal() se define la variable x, a la cual se le asigna el valor cero y se imprime; luego se llama la función cambiar(); y, dado que deseamos conectar esta variable con una variable local de la función, entonces indicamos entre paréntesis dicha variable (cambiar(x)). En el encabezado de la función cambiar(), y especificado entre paréntesis se define la variable que fungirá como enlace con la variable de la otra función; la palabra Ref indica que se está definiendo como parámetro por referencia (llamado tipo variable en algunos lenguajes como Turbo Pascal), y es el nombre de la variable que funge como parámetro (aquí se le llama parámetro formal); y Entero es el tipo de dato de la variable parámetro. Nótese que y queda definida como una variable local, independientemente de que se definan otras en la parte de declaración de variables de la función, es decir, que en una función es posible declarar variables en dos partes: una es la normal (la parte de declaración de variables), y la otra en el encabezado de la función; se definen las variables que fungirán como parámetros. Cuando se llama la función, las dos variables se conectan convirtiéndose en sinónimos, es decir, que x y y son lo mismo, y en la memoria las dos hacen referencia al mismo contenido; esto significa que lo que le sucede a y en la función, le sucederá a x, de ahí que también se le conoce como parámetro tipo variable, ya que su valor puede cambiar.

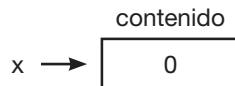
Una vez que el control esté dentro de la función, a y se le asigna el valor de 1, lo cual le sucede también a x. Al regresar a la función principal() se imprime x que tendrá el valor de 1.



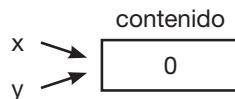
#### Esquemáticamente:

Se define la variable x en la función principal(), la cual tiene su contenido en la memoria, se le coloca 0; gráficamente:

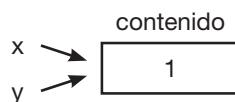
Se le llama parámetro formal al definido en la función, y parámetro actual al parámetro utilizado al hacer la llamada a la función, este concepto se aplica a todos los tipos de parámetros.



Cuando se llama la función cambiar( $x$ ), se conectan  $x$  y  $y$  vía parámetro por referencia, lo que hace que sean sinónimos, es decir, que  $y$  utilice la misma dirección de memoria que  $x$ ; gráficamente:



Cuando a  $y$  se le coloca 1 ¿qué le pasa a  $x$ ?



A  $x$  le pasa lo mismo que le sucede a  $y$ , es decir, a  $y$  se le coloca 1 e indirectamente le sucede lo mismo a  $x$ , porque manejan el mismo contenido en la memoria.

### Parámetro por valor

El **parámetro por valor** es una variable local de una función que se define como parámetro en el encabezado de la misma, y sirve para conectarse con otra variable de otra función mediante el envío de su valor, en el momento de hacer la llamada de la función, después de lo cual ya no hay relación, porque cada variable tiene su propio contenido. De ahí en adelante lo que le sucede a la variable parámetro no afectará a la otra variable, que sólo le envía su valor. A continuación se presenta un algoritmo que muestra cómo utilizar un parámetro por valor.

#### Algoritmo PARÁMETRO POR VALOR

1. Función principal()
    - a. Declarar
      - Variables
      - x: Entero
    - b. x = 0
    - c. Imprimir x
    - d. cambiar(x)
    - e. Imprimir x
    - f. Fin Función principal
  2. Función cambiar(Val y: Entero)
    - a. y = 1
    - b. Fin Función cambiar
- Fin



En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C706.C y Programa en Java: ParametroPorValor.java

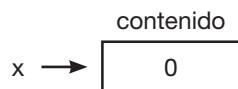
**Explicación:**

Si analizamos este algoritmo, podemos observar que la única diferencia con el anterior es que al definir la variable que funge como parámetro en la función, se indica la palabra Val, lo que significa que es un parámetro por valor. La diferencia con respecto al parámetro por referencia es que al llamar la función, el valor del parámetro actual x se le asigna al parámetro y (de la función) nada más, es decir, que de ahí en adelante lo que le suceda a y es independiente de x. En otras palabras, si x vale 0 al llamar a cambiar(x), a y se le asigna 0 (el mismo valor que x); en la memoria x tiene su propio contenido y y tiene el suyo.

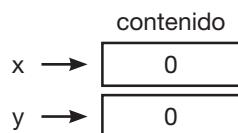
En el ejemplo anterior, a la variable x se le asigna 0; luego se llama cambiar(x), en ese momento a y se le asigna el valor de x, es decir 0. Ya en la función, a y se le asigna 1 y se regresa a la función principal() a imprimir x, la cual tiene el valor de 0, es decir, al ir a la función cambiar() no le afectó lo que se le asignó a y. Por ello este tipo de parámetro funge sólo como un valor que se envía a la función. Cabe hacer la aclaración que se ha usado un solo parámetro, pero pueden usarse más. Los parámetros indicados tanto al definir la función como al llamarla desde la función principal(), deben coincidir en número y tipo.

**Esquemáticamente:**

Se define la variable x en la función principal(), la cual tiene su contenido en la memoria, se le coloca 0; gráficamente:

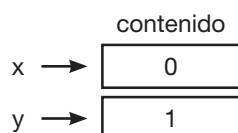


Cuando se llama la función cambiar(), se conectan x y y vía parámetro por valor, lo que hace que el valor de x se le asigne a y; de ahí en adelante x y y no tienen ninguna relación; gráficamente:



Es decir, x tiene su propio contenido en la memoria y y tiene el propio.

Cuando a y se le coloca 1 ¿qué le pasa a x?



A x no le pasa nada, es decir, a y se le coloca 1 en su contenido, y x ni cuenta se da, porque tiene su propio contenido independiente de y.

**Práctica:** En este momento se recomienda ir al punto de ejercicios resueltos y resolver y revisar el Ejercicio 7.7.1. Luego regresar para continuar con el siguiente punto.

## 7.5 Funciones estándar

Las funciones estándar son funciones proporcionadas por cualquier lenguaje de programación de alto nivel, dichas funciones se agrupan como aritméticas, cadena de caracteres o alfabéticas, archivos, grafificación, objetos, especiales, etc. En los capítulos anteriores hemos estudiado algunas funciones estándar, en este capítulo se estudiarán otras funciones, mientras que en otros capítulos se estudiarán otras funciones, por ejemplo, las funciones para archivos se presentan en el capítulo de registros y archivos.

### 7.5.1 Funciones cadena de caracteres

#### Carácter

La función **Carácter** proporciona el carácter representado por un valor entero en sistema numérico decimal, de acuerdo con el código ASCII.

*Formato:*

Carácter (x)

*En donde:*

|          |                                                                                                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Carácter | Es la palabra que identifica a la función.                                                                                                                            |
| x        | Es el parámetro que indica el valor entero en sistema numérico decimal, que se tomará como base para determinar el carácter correspondiente. Debe ser de tipo entero. |
|          | El tipo de dato del resultado es Carácter.                                                                                                                            |

*Ejemplo:*

car = Carácter (65) car toma el valor A, es decir, la letra A corresponde al entero decimal 65 en el código ASCII.

#### Ordinal

La función **Ordinal** proporciona el valor entero decimal que se utiliza para representar un carácter, de acuerdo con el código ASCII.

*Formato:*

Ordinal (x)

*En donde:*

|         |                                                                                                                                                     |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Ordinal | Es la palabra que identifica a la función.                                                                                                          |
| x       | Es el parámetro que indica el carácter al cual se determinará el valor entero decimal que se utiliza para representarlo. Debe ser de tipo Carácter. |
|         | El tipo de dato del resultado es entero.                                                                                                            |

Ejemplo:

```
num = Ordinal("A") num toma el valor 65, es decir, la letra A corresponde  
al entero decimal 65 en el código ASCII.
```

### Concatenar

La función **Concatenar** permite concatenar o “unir” cadenas de caracteres.

Ejemplo:

Si tenemos las variables:

```
nombrePila = "María"  
apellidoPaterno = "López"  
apellidoMaterno = "Acosta"
```

Podemos concatenar de la forma:

```
nombrePersona = Concatenar(nombrePila +  
    apellidoPaterno + apellidoMaterno)
```

o bien, de la forma siguiente:

```
nombrePersona = nombrePila + apellidoPaterno +  
    apellidoMaterno
```

*En donde:*

Concatenar Es una función que realiza la concatenación. Entonces NombrePersona tendrá el contenido “María López Acosta”

### Subcadena

La función **Subcadena** permite obtener una copia de una subcadena (una parte de una cadena).

*Formato:*

```
Subcadena(cadenaFuente, inicio, tamaño)
```

*En donde:*

Subcadena Identifica la función.

cadenaFuente Es la variable tipo cadena de caracteres de la cual se obtendrá la subcadena.

inicio Es un número, variable o expresión de tipo entero que indica la posición desde la cual se empezará a obtener la subcadena.

tamaño Es un número entero, variable o expresión de tipo entero que indica la cantidad de caracteres que serán copiados.

**Ejemplo:**

Si tenemos la cadena nombrePersona con el valor “María López Acosta” y se aplica lo siguiente:

```
a = Subcadena (nombrePersona, 7, 5)
```

a tomará el valor “López”. Porque se copian 5 caracteres desde el 7.

### Borrar

La función **Borrar** borra o quita una subcadena de una cadena.

*Formato:*

```
Borrar (cadenaFuente, inicio, tamaño)
```

*En donde:*

|              |                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------|
| Borrar       | Identifica la función.                                                                                     |
| cadenaFuente | Es la variable tipo cadena de caracteres de la cual se borrará.                                            |
| inicio       | Es un número, variable o expresión de tipo entero que indica la posición desde donde se empezará a borrar. |
| tamaño       | Es un número, variable o expresión de tipo entero que indica la cantidad de caracteres que serán borrados. |

**Ejemplo:**

Si tenemos:

```
comentario = "EL FÚTBOL SOCCER ES POCO INTERESANTE"
```

y le aplicamos lo siguiente:

```
Borrar (comentario, 21, 4)
```

tendremos:

```
comentario = "EL FÚTBOL SOCCER ES INTERESANTE"
```

porque se borra a partir del carácter 21 (desde la P), y se borran cuatro caracteres (POCO).

### Insertar

La función **Insertar** inserta caracteres en una cadena.

*Formato:*

```
Insertar (caracteres, cadenaObjeto, inicio)
```

*En donde:*

|              |                                                                                                                                 |
|--------------|---------------------------------------------------------------------------------------------------------------------------------|
| Insertar     | Identifica la función.                                                                                                          |
| caracteres   | Es una cadena de caracteres ya sea constante o variable.                                                                        |
| cadenaObjeto | Es la variable tipo cadena de caracteres en la cual se insertarán los caracteres.                                               |
| inicio       | Es un número, variable o expresión de tipo entero que indica la posición desde la cual empezará la inserción de los caracteres. |

*Ejemplo:*

Si tenemos:

```
comentario = "EL FÚTBOL SOCCER ES INTERESANTE"
```

y le aplicamos:

```
Insertar("POCO", comentario, 21)
```

Quedará:

```
comentario = "EL FÚTBOL SOCCER ES POCO INTERESANTE"
```

### Longitud

La función **Longitud** proporciona la longitud de una cadena.

*Formato:*

```
Longitud(cadenaFuente)
```

*En donde:*

|              |                                                                                                           |
|--------------|-----------------------------------------------------------------------------------------------------------|
| Longitud     | Identifica la función.                                                                                    |
| cadenaFuente | Es la variable tipo cadena de caracteres de la cual se proporcionará su longitud en número de caracteres. |

*Ejemplo:*

Si tenemos:

```
comentario = "EL FÚTBOL SOCCER ES INTERESANTE"
```

y le aplicamos:

```
n = Longitud(comentario)
```

n tomará el valor 31 porque comentario tiene 31 caracteres.

### Posición

La función **Posición** busca una cadena dentro de otra y proporciona la posición donde inicia.

*Formato:*

```
Posición(cadenaFuente, cadenaBuscada)
```

*En donde:*

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| Posición      | Identifica la función.                                                         |
| cadenaFuente  | Es la variable tipo cadena de caracteres en la cual se buscará la otra cadena. |
| cadenaBuscada | Es la cadena de caracteres que será buscada.                                   |

*Ejemplo:*

Si tenemos:

```
comentario = "EL FÚTBOL SOCCER ES POCO INTERESANTE"
```

y le aplicamos:

```
a = Posición(comentario, "POCO")
```

a tomará el valor de 21, donde empieza la cadena buscada.

### Cadena

La función **Cadena** convierte un valor numérico real o entero en cadena de caracteres.

*Formato:*

```
Cadena(numero, cadenaObjeto)
```

*En donde:*

|              |                                                                                       |
|--------------|---------------------------------------------------------------------------------------|
| Cadena       | Identifica la función.                                                                |
| numero       | Es una variable o número entero o real.                                               |
| cadenaObjeto | Es una variable tipo cadena de caracteres en la cual se colocará el valor convertido. |

*Ejemplo:*

Si tenemos n con el valor 278, y le aplicamos:

```
Cadena(n, n2)
```

n2 toma el valor "278", es decir, transforma el número entero en cadena de caracteres.

✓

La variable numérica tiene la opción de formatearse para tipos reales, por ejemplo, si  $n = 278.55$  se convierte de la siguiente forma:  
 $\text{Cadena}(n:10:2,n2)$   
 Donde se está formateando el contenido de la variable n, a 10 posiciones de salida con 2 decimales, entonces el valor de n2 será "278.55"

Cadena (n:10:2 , n2)

Donde se está formateando el contenido de la variable n, a 10 posiciones de salida con 2 decimales, entonces el valor de n2 será “278.55”

### Valor

La función **Valor** convierte una cadena de caracteres (string) a número entero o real.

*Formato:*

Valor(cadenaFuente, numero, estado)

*En donde:*

|              |                                                                                                                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Valor        | Identifica la función.                                                                                                                                                                           |
| cadenaFuente | Es la cadena de caracteres que se convertirá a numérico.                                                                                                                                         |
| numero       | Es una variable de tipo entero o real, en la cual se asignará el valor de la cadena de caracteres pero en formato numérico.                                                                      |
| estado       | Es una variable tipo entero que tendrá al valor cero si se hizo la conversión correctamente. En caso de no hacerse correctamente, contendrá el número de carácter donde se presentó el problema. |

Ejemplo:

Si tenemos n2 con el valor “278.55”, y la aplicamos:

Valor(n2, n, i)

n toma el valor 278.55; i toma el valor 0, porque se hizo la conversión correctamente.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar las funciones explicadas en este punto.

### 7.5.2 Validación de la entrada de datos

Las funciones del punto anterior son útiles para algunas operaciones, tal como la validación de datos. Por ejemplo, si debemos leer un número entero podríamos hacerlo de la siguiente manera:

A continuación se presenta el algoritmo de la solución:

```
Algoritmo LEER IMPRIMIR N
1. Función principal()
   a. Declarar
      Variables
      n: Entero
   b. Solicitar Número entero
   c. Leer n
```

```

d. Imprimir n
e. Fin Función principal
Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en C: C707.C y Programa en Java: LeelprimeN.java

#### *Explicación:*

Se define la variable n; se solicita un número entero; se lee en n; se imprime el número leído; fin.

También podríamos leer ese número validando la lectura (entrada de datos), ¿que significa esto? Que cuando se teclee el número, puede teclear una letra o símbolo especial; esto sería un error. La idea es que el algoritmo valide la entrada, es decir, detecte si hay error en la entrada del dato y que permita recuperarse del error, esto quiere decir, no quedarse tirado con el error. A continuación se presenta un algoritmo que permite leer el número entero, si hay error, lo vuelve a leer, y así, hasta que se teclee sin error:

```

Algoritmo LEER IMPRIMIR N VALIDANDO
1. Función principal()
   a. Declarar
      Variables
      n, i, bandera, e: Entero
      numero: Cadena
      num: Carácter
   b. do
      1. bandera = 1
      2. Solicitar Número
      3. Leer numero
      4. for i=0; i<=Longitud(numero)-1; i++
         a. num = Subcadena(numero,i,1)
         b. if NOT((num>='0') AND (num<='9'))
            OR(num=='-')) then
            1. bandera = 0
         c. endif
         d. if (num == '-') AND (i > 0) then
            1. bandera = 0
         e. endif
      5. endfor
      c. while bandera != 1
      d. n = valor(numero,n,e)
      e. Imprimir n
      f. Fin Función principal
Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C708.C y Programa en Java: LeelprimeValidando.java



#### *Explicación:*

Se entra en un ciclo repetitivo donde la bandera se “enciende” colocándole 1, enseguida se solicita y lee el número en una variable tipo Cadena, luego se entra en un ciclo que revisa cada uno de los caracteres leídos; si alguno(s) no es un dígito entre 0 y 9 o no es el signo de menos, entonces se “apaga” la bandera, colocándole 0; al llegar al while del ciclo externo, se evalúa si la bandera no está “apagada” (diferente de 1), si es así, se devuelve a entrar al ciclo para volver a leer el número; esto se repetirá mientras que la bandera no se “apague”, es decir, hasta que se haya leído correctamente un número de tipo Entero, es decir, no se le puso 0 (cero), al final se convierte a numérico y se imprime.

#### **Observación:**

La variable bandera pudo haberse definido de tipo Boolean y usar los valores True (Verdadero) y False (Falso); en lugar de 1 y 0.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar validación de la entrada de datos.

### 7.5.3 Funciones especiales

En este punto se estudiarán algunas otras funciones especiales de índole general.

#### **Random**

La función **Random** proporciona un número aleatorio, es decir, un número generado al azar; realmente es un número seudoaleatorio porque es generado por una función matemática, por lo cual no es totalmente aleatorio.

#### *Formato:*

```
Random [ (n) ]
```

#### *En donde:*

Random                      Identifica la función.

n                            Es opcional:

Cuando no está presente se genera un valor entre 0 y 1

$0 \leq \text{valor} < 1$ . El valor es tipo Real.

Cuando está presente se genera un valor entre 0 y N

$0 \leq \text{valor} < n$ . El valor es tipo Entero.

## Ejemplos:

```
num = Random(20)
```

num tomará un valor entre 0 y 20, es decir, cualquiera desde 0 al 19.

```
num = Random
```

num tomará un valor entre 0 y 1, es decir, cualquiera entre 0 y 0.9999999....

*IniciaRandom*

La función **IniciaRandom** permite iniciar la semilla del generador de números aleatorios. Los números aleatorios (seudo) se generan a partir del valor que tiene la semilla, el cual se almacena en la variable Randseed. Cada vez que se ejecuta esta función Randseed toma un nuevo valor generado al azar. Formato:

## IniciaRandom

Sonido

La función Sonido genera un sonido en el altavoz de la computadora de acuerdo con la frecuencia que se le indique hasta que se encuentre con la función NoSonido. Formato:

Sonido (frecuencia)

*En donde:*

Sonido Es el nombre de la función.

**frecuencia** Es un número entero que indica la frecuencia del sonido.

NoSonido

La función NoSonido detiene la generación de sonido en el altavoz. Formato:

NoSonido

### *Detener*

La función Detener detiene el funcionamiento de la computadora la cantidad de milisegundos que se le indique. Formato:

### Detener (tiempo)

En donde:

**tiempo** Es un número entero que indica la cantidad de milisegundos que se detendrá el funcionamiento de la computadora.

Ejemplo:

```
Sonido(250) /* Se activa el sonido con frecuencia  
250 */
```

```
Detener(500) /* Se detiene el funcionamiento  
500 milisegundos,  
antes de ir a la sig. Instrucción */  
NoSonido /* Se detiene el sonido */
```

### *SepararReal*

La función SepararReal separa un número de tipo real en su parte entera y fraccionaria. Formato:

```
parteReal = SepararReal(numero, parteEntera)
```

*En donde:*

|             |                                                                       |
|-------------|-----------------------------------------------------------------------|
| SepararReal | Es el nombre de la función.                                           |
| numero      | Es el número de tipo real que se descompondrá.                        |
| parteReal   | Es una variable de tipo real donde se colocará la parte fraccionaria. |
| parteEntera | Es una variable de tipo real donde se colocará la parte entera.       |

Ejemplo:

```
x = SepararReal(234.85, y)
```

Explicación:

En x coloca .85

En y coloca 234

## **7.6 Funciones que regresan valor**

Este tipo de funciones las puede definir el programador con el propósito de ejecutar alguna función específica, y que por lo general se usan cuando se trata de hacer algún cálculo que será requerido en varias ocasiones en la función principal() o en alguna otra función del algoritmo. La función que devuelve valor es casi igual en todos los aspectos a la función que no regresa valor (void), excepto que la función que devuelve valor, regresa un valor de un tipo de datos simple a su punto de referencia y se utiliza como si fuera una variable de un tipo simple de datos.

El nombre de la función puede estar seguido por uno o más parámetros encerrados entre paréntesis. Por lo general transfieren datos a parámetros por valor.

### Definición de funciones que devuelven valor

Para definir funciones que devuelven valor se utiliza el siguiente formato:

```
Función nomFuncion(parametro) Tipo de dato
a. Declarar
    Constantes
    Tipos
    Variables
b. Acción1
c. Acción2
d. AcciónN
e. return valor
f. Fin Función nomFuncion
```

*En donde:*

|              |                                                                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Función      | Es la palabra que indica que se está definiendo un función.                                                                                                |
| nomFuncion   | Es el nombre de la función el cual fungirá como cualquier variable de tipo simple.                                                                         |
| parámetro    | Es la variable que fungirá como parámetro. Es la que recibe el valor que servirá como base para hacer los cálculos en la función (puede haber más de uno). |
| Tipo de dato | Es el tipo de dato del resultado que regresará la función.                                                                                                 |
| return       | Indica que la función está regresando el valor obtenido o calculado.                                                                                       |
| valor        | Es el valor que regresa la función.                                                                                                                        |

### Referencia de funciones que devuelven valor

Para llamar o invocar a una función que regresa valor, se indica el nombre de la misma poniendo entre paréntesis el parámetro que se envía, utilizándose como cualquier otra variable de tipo simple. Por ejemplo:

```
x = 5
n = nomFuncion(x)
```

x toma el valor 5; luego se llama a la función nomFuncion, enviándole como parámetro x, la función utiliza el valor de x y hace algún cálculo devolviendo un valor, el cual se coloca en n.

*Ejemplo:*

Elaborar un algoritmo que evalúe la función:

$$f = \frac{X!}{Y! (X - Y)!}$$

de la cual se tienen los datos X y Y; mismos que deberán solicitarse y leerse. Factorial de X se divide entre el Factorial de Y multiplicado por el factorial de X-Y.

A continuación se presenta el algoritmo de la solución:

```
Algoritmo EVALÚA EL VALOR DE F
1. Función principal()
   a. Declarar
      Variables
      x, y: Entero
      f: Real
   b. Solicitar X, Y
   c. Leer x, y
   d. f = factorial(x) / (factorial(y) * factorial(x-y))
   e. Imprimir f
   f. Fin Función principal
2. Función factorial(Val n: Entero) Entero
   a. Declarar
      Variables
      i, fact: Entero
   b. if n == 0 then
      1. fact = 1
   c. else
      1. fact = 1
      2. for i=n; i>=1; i--
         a. fact = fact * i
      3. endfor
   d. endif
   e. return fact
   f. Fin Función factorial
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C709.C y Programa en Java: EvaluaF.java



#### Explicación:

Se definen dos funciones: Una es la función principal(); la otra función, calcula el factorial de un número, el cual es recibido en n como parámetro por valor:

1. Función principal()
  - a. Se declaran las variables de la función principal  
x, y : Entero  
f : Real

- b. Se solicitan X, Y
  - c. Se leen en x,y
  - d. Calcula  $f = \text{factorial}(x) / (\text{factorial}(y) * \text{factorial}(x-y))$   
Llama a la función factorial() tres veces:
    - factorial(x) Enviando x como parámetro
    - factorial(y) Enviando y como parámetro
    - factorial(x-y) Enviando el resultado de x-y como parámetro
 Cada vez factorial devuelve el valor correspondiente al factorial del valor del parámetro.
  - e. Imprime f
  - f. Fin de la función
2. Función factorial(Val n: Entero) Entero  
 Define n como parámetro donde recibirá el valor que se le envíe como parámetro cada vez que sea llamada desde la función principal(). Devuelve un valor de tipo entero.
- a. Se declaran las variables necesarias en la función
    - i, fact : Entero
  - b. Si  $n == 0$  Entonces
    1. Coloca 1 en fact, es decir, 1 es el factorial de 0
  - c. Si no
    1. Inicia fact en 1
    2. Inicia ciclo for desde i = n hasta 1 con decrementos de -1
      - A. Calcula  $\text{fact} = \text{fact} * i$
      - 3. Fin del for
  - d. Fin del if
  - e. Regresa el valor que calculó en fact
  - f. Fin Función
- Fin del algoritmo

## 7.7 Ejercicios resueltos

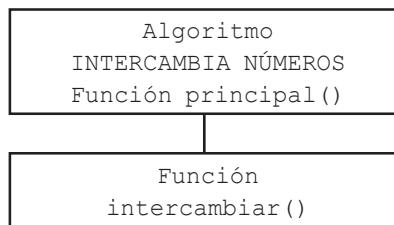
### Ejercicio 7.7.1

Elaborar un algoritmo que permita leer dos números de tipo real en la función principal, que en una función los intercambie vía parámetros por referencia, y los imprima en la función principal. Usar parámetros por referencia.

A continuación se tiene el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*

A continuación se tiene el diagrama general de la solución:



Ahora tenemos el algoritmo en pseudocódigo:

```

Algoritmo INTERCAMBIA NÚMEROS
1. Función principal()
   a. Declarar
      Variables
      a, b: Real
   b. Solicitar dos números
   c. Leer a, b
   d. Imprimir a, b
   e. intercambiar(a,b)
   f. Imprimir a, b
   g. Fin Función principal
2. Función intercambiar(Ref a1, b1: Real)
   a. Declarar
      Variables
      auxiliar: Real
   b. auxiliar = a1
   c. a1 = b1
   d. b1 = auxiliar
   e. Fin Función intercambiar
Fin
  
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C710.C y Programa en Java: IntercambiaNumeros.java



*Explicación:*

1. Función principal()
  - a. Se declaran las variables
  - b. Solicitud dos números
  - c. Lee a, b
  - d. Imprime a, b

- e. Llama intercambiar(a,b) enviando a y b como parámetros, que se conectarán con a1 y b1.
  - f. Imprime a, b
  - g. Fin Función
2. Función intercambiar(Ref a1, b1: Real)
- a1 y b1 se definen como parámetros por referencia, que se conectarán con a y b, respectivamente al ser llamada esta función.
- a. Se declara la variable auxiliar
  - b. Se coloca a1 en auxiliar
  - c. Se coloca b1 en a1
  - d. Se coloca auxiliar en b1
  - e. Fin Función
- Fin del algoritmo

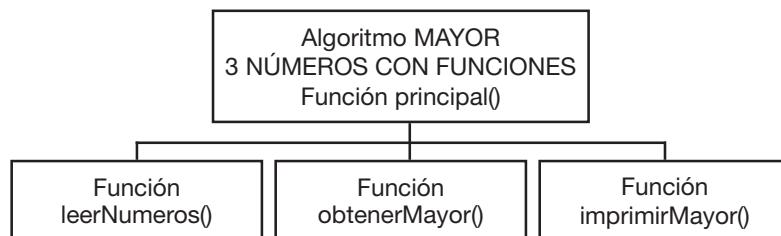
#### Ejercicio 7.7.2

Elaborar un algoritmo que permita leer tres números de tipo entero e imprima el mayor, utilizando una función para leer los números, otra función para obtener y devolver el mayor y otra función para imprimir el mayor. Usar parámetros por valor y por referencia.

A continuación se tiene el algoritmo de la solución:

*(Primero hágalo usted...después compara la solución)*

A continuación se tiene el diagrama general de la solución:



Ahora tenemos el algoritmo en seudocódigo:

```

Algoritmo MAYOR 3 NÚMEROS CON FUNCIONES
1. Función principal()
    a. Declarar
        Variables
        a, b, c, may: Entero
    b. leerNumeros(a, b, c)
    c. may = obtenerMayor(a, b, c)
    d. imprimirMayor(may)
    e. Fin Función principal
  
```

```
2. Función leerNumeros(Ref n1,n2,n3: Entero)
   a. Solicitar tres números
   b. Leer n1, n2, n3
   c. Fin Función leerNumeros
3. Función obtenerMayor(Val num1,num2,num3: Entero)
   Entero
   a. Declarar
      Variables
      mayor: Entero
   b. mayor = num1
   c. if num2 > mayor then
      1. mayor = num2
   d. endif
   e. if num3 > mayor then
      1. mayor = num3
   f. endif
   g. return mayor
   h. Fin Función obtenerMayor
4. Función imprimirMayor(Val m: Entero)
   a. Imprimir m, "ES EL MAYOR"
   b. Fin Función imprimirMayor
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en C: C711.C y Programa en Java: Mayor3Numeros4.java



*Explicación:*

1. Función principal()
  - a. Se declaran las variables
  - b. Llama leerNumeros(a, b, c) enviando a, b y c como parámetros.
  - c. Coloca en may lo que regrese obtenerMayor(a, b, c) enviando a, b y c como parámetros.
  - d. Llama imprimirMayor(may) enviando may como parámetro.
  - e. Fin Función
2. Función leerNumeros(Ref n1,n2,n3: Entero)  
n1, n2 y n3 se definen como parámetros por referencia; se conectarán con a, b y c
  - a. Solicitud tres números
  - b. Lee n1, n2, n3
  - c. Fin Función

3. Función obtenerMayor(Val num1,num2,num3: Entero) Entero  
num1, num2 y num3 se declaran como parámetros por valor, que se conectarán con a, b y c
    - a. Se declara la variable mayor
    - b. Se coloca num1 en mayor
    - c. Si num2 > mayor Entonces
      1. Se coloca num2 en mayor
    - d. Fin del if
    - e. Si num3 > mayor Entonces
      1. Se coloca num3 en mayor
    - f. Fin del if
    - g. return mayor
    - h. Fin Función
  4. Función imprimirMayor(Val m: Entero)  
m se declara como parámetro por valor, que se conecta con may
    - a. Imprime m, "ES EL MAYOR"
    - b. Fin Función
- Fin del algoritmo

**Nota:** En estos últimos algoritmos se han elaborado diversas funciones que permiten hacer validaciones; es conveniente que el programador las utilice cotidianamente en los algoritmos que desarrolle, porque son indispensables para que nuestros algoritmos tengan una mejor calidad.



La tabla 7.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 7 de la Web del libro.

**Tabla 7.1**

| Ejercicio        | Descripción                                        |
|------------------|----------------------------------------------------|
| Ejercicio 7.7.3  | Realiza cálculos con medidas estadísticas          |
| Ejercicio 7.7.4  | Eleva a una potencia en una función                |
| Ejercicio 7.7.5  | Calcula la suma de raíces cuadradas en una función |
| Ejercicio 7.7.6  | Calcula la suma de cuadrados en una función        |
| Ejercicio 7.7.7  | Calcula la media de un arreglo con funciones       |
| Ejercicio 7.7.8  | Eleva a potencias                                  |
| Ejercicio 7.7.9  | Obtiene el mayor y menor de una matriz de números  |
| Ejercicio 7.7.10 | Determina si una matriz es simétrica o no          |
| Ejercicio 7.7.11 | Obtiene el dígito verificador de un número         |
| Ejercicio 7.7.12 | Indica si una frase es un palíndromo               |
| Ejercicio 7.7.13 | Dice fecha con letras                              |
| Ejercicio 7.7.14 | Realiza validación en entrada de datos             |
| Ejercicio 7.7.15 | Indica cuántas veces se usó cada vocal             |
| Ejercicio 7.7.16 | Separa una frase en palabras                       |
| Ejercicio 7.7.17 | Separa una frase en palabras e indica la mayor     |
| Ejercicio 7.7.18 | Genera números aleatorios                          |



## 7.8 Ejercicios propuestos

1. Elaborar un algoritmo que proporcione un menú que permita escoger una de las funciones siguientes: tangente, cotangente, secante y cosecante. El cálculo de cada una es de la forma siguiente:

$$\text{Tangente}(X) = \frac{\text{Seno}(X)}{\text{Coseno}(X)}$$

$$\text{Cotangente}(X) = \frac{\text{Coseno}(X)}{\text{Seno}(X)}$$

$$\text{Secante}(X) = \frac{1}{\text{Coseno}(X)}$$

$$\text{Cosecante}(X) = \frac{1}{\text{Seno}(X)}$$

2. Elaborar un algoritmo que proporcione un menú donde ofrezca hacer conversiones entre metros, yardas, pulgadas y pies. Si escoge metros, en una función debe leer la cantidad N de metros y que imprima una tabla de equivalencias a yardas, pulgadas y pies; desde 1 metro hasta N metros de uno en uno. Equivalencias: 1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 metro = 100 cm. Se debe imprimir la tabla siguiente:

| METROS | CONVERSIONES |          |         |
|--------|--------------|----------|---------|
|        | YARDAS       | PULGADAS | PIES    |
| 1      | 9999.99      | 9999.99  | 9999.99 |
| 2      | 9999.99      | 9999.99  | 9999.99 |
| ---    | ---          | ---      | ---     |
| N      | 9999.99      | 9999.99  | 9999.99 |

Y así si escoge pies, pulgadas o yardas se hace lo propio para cada una de ellas.

3. Similar al anterior, sólo que se lee un valor inicial y un valor final. Por ejemplo, si los valores leídos son 100 y 200, las conversiones se harán desde 100 hasta 200 de uno en uno.
4. Elaborar un algoritmo que ofrezca un menú que permita escoger la impresión de las tablas de multiplicar, dividir, sumar y restar, y dentro de cada opción que permita escoger las del 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10.
5. Elaborar un algoritmo que defina tres matrices A,B y S de 7 X 7, que lea números enteros para A y B; que obtenga la S multiplicando los elementos A(1,1) y B(1,1) y lo coloque en el S(1,1), y así sucesivamente. Que al final imprima las tres matrices. Utilizar una función para leer y otra para imprimir.
6. El sistema monetario mexicano incluye entre billetes y monedas las denominaciones siguientes: 1000, 500, 200, 100, 50, 20, 10, 5, 2, 1, pesos y de 50, 20, 10, 5 centavos. Hacer un algoritmo que lea una cantidad y que

imprima cuántos billetes y monedas y de qué denominaciones se necesitan para cubrir dicha cantidad. Por ejemplo, la cantidad 577.80 generaría: 5 billetes de 100, 1 de 50, 1 de 20, 1 de 5, 1 de 2 pesos, una moneda de 50 centavos, 1 de 20, 1 de 10. Nota: En lugar de 5 de 100; podrían ser dos de 200 y uno de 100; o uno de 500.

7. Igual al problema del capítulo 5, que calcula el sueldo a empleados, y al anterior, que al final presente una distribución de efectivo donde indique cuántos billetes y monedas y de qué denominaciones se requieren para cubrir el total a pagar.

8. Según el teorema de Pitágoras el cuadrado de la hipotenusa es igual a la suma del cuadrado de los catetos ( $C^2 = A^2 + B^2$ ). Utilizando este concepto es posible conocer de qué tipo es un triángulo:

Si  $C^2 = A^2 + B^2$  es un triángulo rectángulo

Si  $C^2 < A^2 + B^2$  es un triángulo acutángulo

Si  $C^2 > A^2 + B^2$  es un triángulo obtusángulo

Elaborar un algoritmo que permita leer los dos catetos y la hipotenusa e imprima el tipo de triángulo que es. Usar una función que reciba los catetos y devuelva el tipo de triángulo que es.

9. Elaborar un algoritmo que imprima las potencias de 2 que no excedan 8500. Utilizando una función que calcule la potencia de cada número.

10. Elaborar un algoritmo que lea 20 números enteros y positivos; y que para cada uno imprima si es par o impar. Utilizar una función que determine si es par o impar.

11. Elaborar un algoritmo que lea un valor X (entero y positivo); y que imprima el valor de y.

$$Y = \frac{X^1}{1!} + \frac{X^2}{2!} + \frac{X^3}{3!} + \dots + \frac{X^x}{X!}$$

Usar una función que eleve X a la potencia y otra función que calcule el factorial en cada ocasión.

12. Elaborar un algoritmo que lea números enteros en un arreglo de 5X7, que lo imprima, e imprima la sumatoria por renglones y por columnas; utilizando un arreglo para el cálculo de las sumatorias de los 5 renglones y lo propio para las 7 columnas. Utilizando una función para hacer la lectura, otra para los cálculos y otra para imprimir, éstos dirigidos por una función de control que es la principal.

13. Elaborar un algoritmo que permita leer diez valores para A, B y C, calcular los valores de X1 y X2 para cada tripleta de valores e imprimir A, B, C, X1 y X2. Utilizar una función para hacer los cálculos. Utilizar la ecuación cuadrática:

$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

14. Elaborar un algoritmo que lea dos números y llame una función que calcule el máximo común divisor (mcd) y otra que calcule el mínimo común múltiplo (mcm) y los imprima al regresar a la función principal.
15. Similar al anterior, sólo que ahora se deberán leer ocho veces los dos números.
16. Elaborar un algoritmo que permita leer el tamaño de un ángulo y calcule e imprima el seno, coseno y tangente. Utilizar funciones.
17. Un número entero es primo si es divisible sólo por la unidad y por sí mismo. Elaborar un algoritmo que lea un valor N, y calcule e imprima los números primos entre 1 y N.
18. Elaborar un algoritmo que lea el dividendo y divisor de una división entera, que llame una función la cual envíe dichos valores mediante parámetros, calcule el cociente y el residuo de la división sin utilizar los operadores \, Mod y devuelva los resultados mediante parámetros.
19. Elaborar un algoritmo que permita leer un número entero e indique si es capicúa (es capicúa si se lee igual de izquierda a derecha que en sentido contrario). Utilizar una función que reciba como parámetro el número, y que proporcione el valor True si es capicúa o False en caso de no ser capicúa. Ejemplo: 1991.
20. Elaborar un algoritmo similar al que calcula la media, varianza y desviación estándar, pero ahora usar variables locales en las funciones y además, que se utilice una función para leer los números y funciones que regresan valor para calcular el valor de cada medida.
21. Elaborar un algoritmo que permita leer una frase u oración y que imprima la palabra más larga, la más corta y la posición donde inicia cada una de ellas. Utilizar una función que obtenga cada palabra.
22. Elaborar un algoritmo que permita leer una expresión aritmética, la cual estará formada por operadores aritméticos, operandos y paréntesis. Separar y colocar los operandos en un arreglo y los operadores en otro; por último, que imprima la frase y los dos arreglos. Los paréntesis no se colocarán en ninguno de los dos arreglos.
23. Elaborar un algoritmo que permita leer las longitudes de los tres lados de un triángulo y calcular e imprimir cada uno de los tres ángulos, utilizando una función para el cálculo de cada uno de los ángulos.
24. Elaborar un algoritmo que permita leer números enteros en una matriz de 4X5 e imprimir los elementos que al mismo tiempo sean el mayor de su renglón y el mayor de su columna.
25. Elaborar un algoritmo que permita leer una frase u oración e imprima cuántas veces se utilizó cada letra del alfabeto. Utilizar una función que reciba como parámetros la frase y la posición en la que se debe obtener el carácter, y que obtenga un carácter en cada llamada.
26. Elaborar un algoritmo que permita leer una frase u oración y que imprima la palabra más larga, contemplar la posibilidad de que haya más de una palabra con la longitud más larga, en tal caso, se puede utilizar un arreglo para guardarlas y al final imprimir todas las palabras que tuvieron la máxima longitud.

27. Elaborar un algoritmo que permita leer 10 números enteros en un arreglo, e imprimir cuántas veces se utilizó cada uno de los dígitos del 0 al 9.
28. En un triángulo rectángulo es posible que se desee calcular el cateto A, el cateto B o la hipotenusa C, de acuerdo con los datos que se tengan disponibles. Elaborar un algoritmo que permita seleccionar el cálculo a realizar, leer los datos correspondientes e imprimir el resultado. Por ejemplo, si se escoge calcular la hipotenusa, se deben leer el cateto A y el cateto B; si se selecciona calcular el cateto A, se leen el cateto B y la hipotenusa C; si se escoge calcular el cateto B, se deben leer el cateto A y la hipotenusa. Utilizar funciones para hacer cada uno de los cálculos.
29. Elaborar un algoritmo que permita leer una cantidad de tipo Real, que le inserte comas cada tres cifras y la imprima.

## 7.9 Resumen de conceptos que debe dominar

- Funciones que no regresan valor (void)
- Variables globales y locales; parámetros por referencia y por valor
- Funciones que regresan valor
- Funciones cadenas de caracteres
- Validación de la entrada de datos
- Funciones especiales

## 7.10 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

### 7.10.1 Resumen gráfico del capítulo

### 7.10.2 Autoevaluación

### 7.10.3 Programas

### 7.10.4 Ejercicios resueltos

### 7.10.5 Power Point para el profesor (\*)

# 8

## Registros y archivos

### Contenido

- 8.1 Organización de archivos
- 8.2 Manejo de registros en seudocódigo
- 8.3 Operaciones para el manejo de archivos en seudocódigo
- 8.4 Proceso de un archivo secuencial
- 8.5 Proceso de un archivo directo
- 8.6 Ejercicios resueltos
- 8.7 Ejercicios propuestos
- 8.8 Resumen de conceptos que debe dominar
- 8.9 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*)  
sólo está disponible para docentes.
  - 8.9.1 Resumen gráfico del capítulo
  - 8.9.2 Autoevaluación
  - 8.9.3 Programas
  - 8.9.4 Ejercicios resueltos
  - 8.9.5 Power Point para el profesor (\*)

### Objetivos del capítulo

- Estudiar los conceptos de registros y archivos.
- Estudiar la organización de archivos.
- Estudiar cómo manejar registros y archivos en seudocódigo.
- Estudiar el proceso de archivos con organización secuencial y organización directa.
- Aprender a desarrollar algoritmos para programas grandes conformando sistemas de información.

## Introducción

Con el estudio del capítulo anterior, usted ya domina el proceso de modularización del diseño descendente (Top down design) y su utilización en forma integrada con seudocódigo.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando las estructuras de datos: registros y archivos.

Se explica que un dato describe un atributo o característica de un objeto o sujeto de proceso de datos; que el registro es un tipo de dato formado por un conjunto de datos que describen a un objeto o sujeto de proceso de datos; y que el archivo es un tipo de dato formado por un conjunto de registros que contienen datos acerca de un mismo objeto o sujeto de proceso de datos.

Se puntualiza que los archivos se almacenan en medios externos, como son los discos magnéticos duros o flexibles, dispositivos ópticos, cintas magnéticas, entre otros, mismos que permiten el almacenamiento permanente de datos para usos futuros, y constituyen un elemento esencial en el uso de las computadoras.

Se define que la organización de archivos es el procedimiento que sirve para relacionar la forma de registrar los datos con la forma de procesarlos, que existen diversas formas de organización como son la secuencial, directa o relativa, secuencial indexado, indexado no secuencial, entre otras. Se expone el manejo de registros en seudocódigo, las operaciones para el manejo de archivos en seudocódigo, características de los archivos, tipos de archivos, operaciones sobre archivos, operaciones con registros. Se estudia el proceso de un archivo con organización secuencial y el proceso de un archivo con organización directa. Se presenta el desarrollo de algoritmos para programas grandes que conforman sistemas de información.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejerzte estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que, a veces, aunque haya diferencias, las dos soluciones son correctas.

En el siguiente capítulo se estudian otros tipos de datos y otros temas.

Los tipos de datos que se han tratado hasta este momento, están limitados por residir sólo en la memoria principal, esto significa que si se apaga la computadora desaparecen. Otra limitante es la cantidad de datos que se podrían almacenar, ya que este tipo de memoria es muy limitada por su alto costo.

En este capítulo estudiaremos la estructura de datos llamada **archivos**, los cuales se almacenan en medios externos, como son los discos magnéticos duros o flexibles, dispositivos ópticos, cintas magnéticas, entre otros, mismos que permiten el almacenamiento permanente de datos para usos futuros, y constituyen un elemento esencial en el uso de las computadoras.

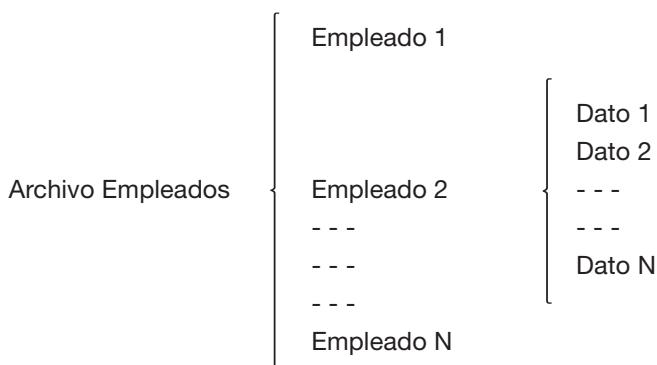
Un **archivo** es un conjunto ordenado de registros que contienen datos acerca de un mismo objeto o sujeto de proceso de datos.

Un **registro lógico** es un conjunto de datos que describen a un objeto o sujeto de proceso de datos.

Un **dato** describe un atributo o característica de un objeto o sujeto de proceso de datos.

Ejemplo:

Un archivo de Empleados es un conjunto de registros; cada registro pertenece a un empleado y contiene un conjunto de datos elementales que describen al empleado. Esquematizando tenemos:



Otra forma de esquematizarlo sería:

|            | Número | Nombre           | Dept. | Puesto | Sueldo |
|------------|--------|------------------|-------|--------|--------|
| Registro 1 | 10     | IGNACIO CAMACHO  | 1     | 1      | 1200   |
| Registro 2 | 20     | MEDARDO OBESO    | 1     | 2      | 1345   |
| Registro 3 | 30     | PORFIRIO LÓPEZ   | 2     | 1      | 1750   |
| Registro 4 | 40     | CAMEROLY OBESO   | 2     | 3      | 1450   |
| Registro 5 | 50     | CELEDONIO FLORES | 3     | 1      | 3400   |
| ---        |        |                  |       |        |        |
| ---        |        |                  |       |        |        |
| Registro N | 90     | ANTONIO HIGUERA  | 4     | 5      | 1250   |

**Nota:** El Depto. 1 puede ser Compras, el 2 Contabilidad, el 3 Ventas, etc. El Puesto 1 puede ser Director, el 2 Sub-director, el 3 Secretaria, el 4 Auxiliar, etcétera.

**La llave o clave** es un dato o grupo de datos dentro del registro que sirve para identificar de manera única a cada registro. En el caso del archivo de Empleados, el número es la llave que identifica a cada empleado.

**El registro físico o bloque** es la unidad de datos de lectura o escritura de un medio físico de archivo de datos, el cual puede contener varios registros lógicos; su tamaño depende de las características físicas de la computadora. En el caso de las micros es fijo, generalmente de 256, en las minis de 512; en las computadoras grandes es variable, y puede definirlo el usuario en múltiplos de 256.

**El factor de bloque** es el número de registros lógicos que contiene un registro físico o bloque.

De aquí en adelante, cuando mencionemos la palabra **registro**, nos estaremos refiriendo al concepto de **registro lógico**.

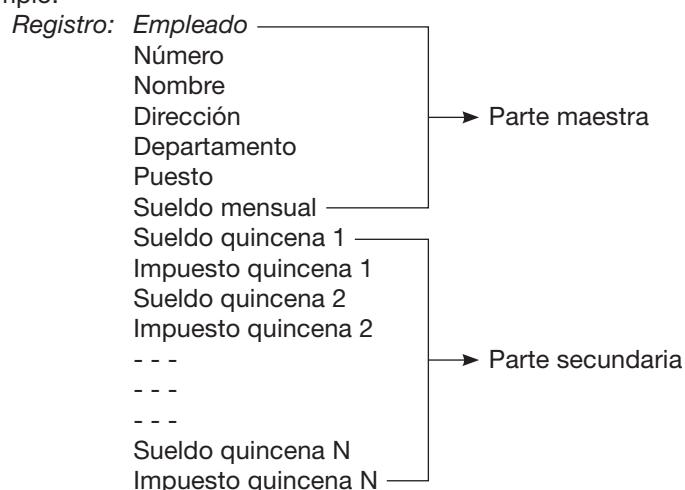
### Características de los archivos

- Son independientes con respecto a los programas.
- Un archivo pueden utilizarlo distintos programas en diversos momentos.
- La información almacenada es permanente.
- Tienen gran capacidad de almacenamiento.
- La recuperación de datos se hace con rapidez.
- Su índice de confiabilidad es muy alto.

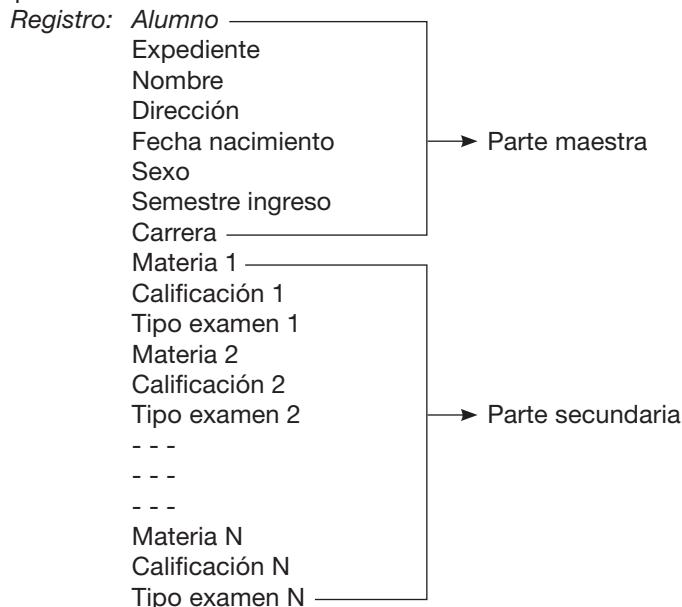
### Partes de un registro

La **Parte maestra** está definida por la información sujeta a pocos cambios, y sirve como historia, identificación y referencia. La(s) **Parte(s) secundaria(s)** está(n) definida(s) por la información sujeta a cambios constantes, sobre todo cambios de expansión.

Ejemplo:



Ejemplo:



### Tipos de archivos

**Archivos maestros.** Contienen registros con información general de poca variación. Sirven como identificación, referencia, estadística, historia. Ejemplo: Archivo maestro de Empleados, archivo de alumnos, archivo de artículos, archivo de clientes, etcétera.

**Archivos de transacciones.** Contienen registros con datos que describen las operaciones de la organización. Surgen como resultado de algún proceso de transacciones y sirven para preparar documentos de movimientos diarios, semanales, quincenales, etc. Generalmente son temporales y se utilizan para actualizar los archivos maestros. Ejemplos: archivo de movimientos de almacén, archivo de ventas, archivo de abonos, archivo de inscripción, etcétera.

**Archivos de trabajo.** Se utilizan como auxiliares en el proceso de los dos tipos de archivos anteriores. Su duración normalmente es menor a la duración de la ejecución de un programa.

### Operaciones sobre archivos

|                      |                                                                                     |
|----------------------|-------------------------------------------------------------------------------------|
| <b>Creación</b>      | Crear el archivo y escritura de sus registros.                                      |
| <b>Consulta</b>      | Lectura de registros.                                                               |
| <b>Actualización</b> | Inserción (altas), supresión (bajas) y modificación (cambios) en algunos registros. |
| <b>Clasificación</b> | Reordenamiento de los registros de acuerdo con cierto criterio.                     |
| <b>Borrado</b>       | Eliminación total del archivo.                                                      |

### Operaciones con registros

**Inserción** Añadir un nuevo registro al archivo.

**Supresión** Quitar un registro del archivo.

**Consulta** Leer el contenido de un registro.

**Actualización** Cambiar datos de un registro.

## 8.1 Organización de archivos

---

Es el procedimiento que sirve para relacionar la forma de registrar los datos con la forma de procesarlos. La organización de archivos proporciona los métodos para la creación, actualización y acceso de archivos.

### Tipos de organización de archivos

- Secuencial
- Directa o relativa
- Secuencial indexado
- Indexado no secuencial
- Otras

El **registro de los datos** es la operación que consiste en introducir los datos al archivo, con el propósito de que queden almacenados para usos posteriores, y puede ser:

- Secuencial
- Directo
- En desorden

El **proceso o acceso** consiste en buscar datos en el archivo con el propósito de consultarlos o modificarlos, y puede ser:

- Secuencial
- Directo

### Organización secuencial

Esta forma de organización de archivos tiene las siguientes características:

- Los registros se almacenan físicamente de acuerdo con una llave dada en secuencia ascendente o descendente.
- Es muy eficaz para el proceso periódico en lotes.
- Cuando se requiere buscar un registro determinado, es necesario recorrer todo el archivo en forma secuencial, uno a uno, hasta encontrarlo.
- Es conveniente que el último registro del archivo sea un centinela que indique el fin.

- No es eficiente para la búsqueda de información.
- Si se encuentra posicionado en un determinado registro, no se puede regresar; se debe reiniciar el proceso desde el principio.
- Si quiere dar de baja un registro, no puede hacerlo directamente, sólo se marca como dado de baja. Posteriormente, se deberá hacer un proceso especial de reorganización para que los registros se den de baja físicamente.
- Cuando se desea actualizar el archivo con altas, bajas y cambios, es necesario crear otro archivo con las modificaciones, luego pasar los dos archivos completos, creando uno nuevo actualizado.

Ejemplo:

Se tiene el archivo de Empleados, en el que se han registrado los datos en orden secuencial de acuerdo con el Número, como se muestra a continuación:

| Número | Nombre             | Dept. | Puesto | Sueldo |
|--------|--------------------|-------|--------|--------|
| 1      | MARICELA VILLA     | 1     | 1      | 1200   |
| 2      | EDILIA CAMACHO     | 1     | 2      | 1450   |
| 3      | DANIA CAMACHO      | 1     | 3      | 1750   |
| 4      | JUAN DE DIOS LÓPEZ | 2     | 1      | 1450   |
| 5      | CAROLINA OBESO     | 2     | 2      | 2300   |
| 6      | MANUEL MEZA        | 3     | 1      | 3000   |
| 7      | ARACELI LÓPEZ      | 3     | 2      | 1500   |

### Organización directa o relativa

Cuando hablamos de organización directa o de acceso directo a un archivo, lo que quiere decir es que los datos se localizan en una posición conocida por el dispositivo, el cual es de acceso directo. Los registros se almacenan en una dirección relativa conocida, y siempre de acuerdo con la llave, así, el registro con la llave número 1 se guarda a partir del byte 0 del archivo, el registro con la llave 2 se guarda a partir del byte siguiente después de donde se almacenó el registro anterior y así sucesivamente, como se muestra a continuación: (Se supone que el registro de un empleado ocupa 40 bytes.)

Archivo de Empleados:

| Dirección del byte | Nro. | Nombre            | Dept. | Puesto | Sueldo |
|--------------------|------|-------------------|-------|--------|--------|
| 0                  | 1    | JESÚS CRISTO      | 4     | 1      | 1200   |
| 40                 | 2    | MARÍA ALIMATEA    | 4     | 2      | 1450   |
| 80                 | 3    | CAROLINA O. LÓPEZ | 8     | 1      | 1750   |
| 120                | 4    | PEDRO APÓSTOL     | 9     | 1      | 1450   |
| 160                | 5    | CLAUDIA OBESO     | 9     | 2      | 2300   |
| N-1*40             | N    | MARÍA L. LÓPEZ    | 10    | 3      | 2500   |

Para el acceso a los registros, se debe conocer la llave, se busca la posición correspondiente de acuerdo con el número de empleado y a los 40 bytes que ocupa el registro del empleado.

## 8.2 Manejo de registros en seudocódigo

Como se explicó al inicio de este capítulo, el concepto de registro está íntimamente ligado al concepto de archivos, sin embargo, es posible manejarlo en forma independiente. El registro es un tipo de dato estructurado constituido por un conjunto de elementos individuales (datos) que pueden ser de diferentes tipos de datos.

Ejemplo:

## Registro: empleado

Elementos del registro empleado: numero  
nombre  
depto  
puesto  
sueldo

*En donde:*

- numero es tipo Entero
  - nombre es Cadena[30]
  - depto es Entero
  - puesto es Entero
  - sueldo es Real

## Definición de registros

El registro se define en la parte de declaraciones; existen dos formas de definirlo:

- a) Se define directamente en la declaración de variables.

*Formato:*

*En donde:*

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| nomRegistro   | Es el nombre mediante el cual se identificará.                                 |
| Registro      | Es la palabra reservada que identifica al tipo de dato como un registro.       |
| dato1, dato2, | Son los identificadores de los datos individuales                              |
| datoN         | que forman el registro, cada uno con su definición particular de tipo de dato. |
| FinRegistro   | Indica que se termina la definición del registro.                              |

*Ejemplo:*

```
Declarar
    Variables
        regEmpleado: Registro
            numero: Entero
            nombre: Cadena[30]
            depto : Entero
            puesto: Entero
            sueldo: Real
    FinRegistro
```

*Explicación:*

Se declara la variable regEmpleado de tipo registro que está compuesta por el conjunto de datos: numero de tipo Entero, nombre de tipo Cadena[30], depto de tipo Entero, puesto de tipo Entero y sueldo de tipo Real.

- b) Se define en la declaración de tipos un nuevo tipo de dato como un registro, luego puede utilizarse en cualquier otra parte, quedando de una forma más generalizada, ejemplo:

```
Declarar
    Tipos
        persona: Registro
            numero: Entero
            nombre: Cadena[30]
            depto: Entero
            puesto: Entero
            sueldo: Real
    FinRegistro
    Variables
        regEmpleado: persona
```

**Explicación:**

Se declara el nuevo tipo de dato persona que es un registro que está compuesto por el conjunto de datos: numero de tipo Entero, nombre de tipo Cadena[30], depto de tipo Entero, puesto de tipo Entero y sueldo de tipo Real. Luego se define la variable regEmpleado de tipo persona, es decir, de tipo registro que está compuesto por el conjunto de datos: numero de tipo Entero, nombre de tipo Cadena[30], depto de tipo Entero, puesto de tipo Entero y sueldo de tipo Real.

**Proceso de un registro**

Los elementos de un registro se pueden utilizar como cualquier otra variable de tipo simple, es decir, en acciones de lectura, escritura, para hacer cálculos, etc. Se relaciona concatenando, mediante un punto, el nombre del dato con el del registro.

**Ejemplo:**

Si se tiene la definición:

```
Declarar
    Tipos
        persona: Registro
            numero: Entero
            nombre: Cadena[30]
            depto: Entero
            puesto: Entero
            sueldo: Real
        FinRegistro
    Variables
        regEmpleado: persona
```

Se manipulan de la forma siguiente:

```
Leer regEmpleado.numero, regEmpleado.nombre
Leer regEmpleado.sueldo
sueldoBruto = regEmpleado.sueldo / 2
Imprimir regEmpleado.numero, regEmpleado.nombre,
        sueldoBruto
```

Observe cómo se concatenan los datos individuales numero, nombre y sueldo con regEmpleado mediante el punto.

**Ejemplo:**

Elaborar un algoritmo que lea e imprima los datos de varios empleados utilizando el registro y los datos que hemos explicado anteriormente.

A continuación tenemos el algoritmo:

```
Algoritmo LEE IMPRIME EMPLEADO
1. Declarar
    Tipos
        persona: Registro
            numero: Entero
            nombre: Cadena[30]
            depto: Entero
            puesto: Entero
            sueldo: Real
    FinRegistro
    Variables
        regEmpleado: persona
        desea: Carácter
2. do
    a. Solicitar datos del empleado
    b. Leer regEmpleado.numero, regEmpleado.nombre,
       regEmpleado.depto, regEmpleado.puesto
       regEmpleado.sueldo
    c. Imprimir regEmpleado.numero, regEmpleado.nombre,
       regEmpleado.depto, regEmpleado.puesto
       regEmpleado.sueldo
    d. Preguntar "¿Desea procesar otro empleado (S/N) ?"
    e. Leer desea
3. while desea == 'S'
4. Fin
```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C801.C



#### *Explicación:*

Se declara la variable `regEmpleado` que es de tipo `persona`, es decir, un registro con los datos indicados.

Se entra en un ciclo donde se solicitan y leen cada uno de los datos en la variable registro `regEmpleado`; luego se imprimen desde la misma variable.

Se termina el ciclo y el algoritmo.

#### Ámbito de los registros

Aunque no es recomendable, dos o más registros pueden tener elementos con idéntico nombre, pero serán diferentes porque pertenecen a distintos registros, así, cada registro tiene su propio ámbito, ejemplo:

```

Declarar
    Tipos
        uno: Registro
            x: Real
            y: Cadena[1]
            z: Entero
        FinRegistro
        dos: Registro
            x: Entero
            y: Real
            z: Carácter
        FinRegistro

```

Para el registro uno existen los elementos uno.x, uno.y, uno.z; y, para el registro dos existen los elementos dos.x, dos.y, dos.z, y no tienen nada que ver entre sí, es decir, son totalmente distintos porque pertenecen a distintos registros.

### Registros jerárquicos

Los registros jerárquicos se presentan cuando un dato de tipo registro contiene un dato o más, que a su vez es un registro; y, en este registro, también podría presentarse la misma situación. Se dice que están jerarquizados porque algunos registros están subordinados o dependen de otros registros.

Ejemplo:

```

Algoritmo REGISTRO JERÁRQUICO
1. Declarar
    Tipos
        fecha: Registro
            dia: Cadena[2]
            mes: Cadena[2]
            anio: Cadena[4]
        FinRegistro
        nomPersona: Registro
            apellidos : Cadena[15]
            nombrePila: Cadena[15]
        FinRegistro
        persona: Registro
            nombre: nomPersona
            fechaIngreso: fecha
            fechaNacim: fecha
            sueldo: Real
        FinRegistro
    Variables
        regEmpleado: persona
        desea: Carácter

```

```
2. do
    a. Solicitar apellidos del empleado
    b. Leer regEmpleado.nombre.apellidos
    c. Solicitar nombre pila
    d. Leer regEmpleado.nombre.nombrePila
    e. Solicitar día de ingreso
    f. Leer regEmpleado.fechaIngreso.dia
    g. Solicitar mes de ingreso
    h. Leer regEmpleado.fechaIngreso.mes
    i. Solicitar Año de ingreso
    j. Leer regEmpleado.fechaIngreso.anio
    k. Solicitar día de nacimiento
    l. Leer regEmpleado.fechaNacim.dia
    m. Solicitar mes de nacimiento
    n. Leer regEmpleado.fechaNacim.mes
    o. Solicitar Año de nacimiento
    p. Leer regEmpleado.fechaNacim.anio
    q. Solicitar sueldo
    r. Leer regEmpleado.sueldo
    s. Imprimir regEmpleado.nombre.apellidos,
        regEmpleado.nombre.nombrePila,
        regEmpleado.fechaIngreso.dia,
        regEmpleado.fechaIngreso.mes,
        regEmpleado.fechaIngreso.anio,
        regEmpleado.fechaNacim.dia,
        regEmpleado.fechaNacim.mes,
        regEmpleado.fechaNacim.anio
    t. Preguntar "¿desea procesar otro empleado (S/N) ?"
    u. Leer desea
3. while desea == 'S'
4. Fin
```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C802.C



*Explicación:*

regEmpleado es una variable declarada de tipo persona.  
persona es un registro que tiene los datos nombre, fechalngreso, fechaNacim y sueldo.  
nombre es a su vez un registro que tiene los datos apellidos y nombrePila.  
fechalngreso y fechaNacim son a su vez registros que contienen los datos dia, mes y año.

Como puede observarse, regEmpleado tiene subordinados los registros nombre, fechalngreso, fechaNacim.

Observe cómo se relaciona cada elemento individual: por ejemplo, al leer el dato apellidos del nombre que depende de regEmpleado:

#### Leer regEmpleado.nombre.apellidos

Se coloca primero el registro con mayor jerarquía (regEmpleado); se concatena con el punto al registro que tiene subordinado (nombre), y por último se concatena con otro punto el identificador del dato apellidos.

#### Arreglos de registros

Se pueden hacer combinaciones de los tipos arreglo y registro, conformando arreglos de registros.

Ejemplo:

Elaborar un algoritmo que permita manejar una agenda en un arreglo de 20 elementos, donde cada elemento es un dato tipo registro que tiene los datos: nombre, Dirección, Teléfono, Fax, Email, que permita crear la agenda colocando nulos a todos los elementos del arreglo de registros, dar altas de los datos de personas en la agenda, bajas de personas de la agenda, cambios en datos de las personas de la agenda, hacer consultas de personas y obtener un listado con los datos de las personas de la agenda. Debe ofrecer el menú siguiente:

| SISTEMA DE AGENDA    |
|----------------------|
| 1. CREAR AGENDA      |
| 2. ALTAS             |
| 3. BAJAS             |
| 4. CAMBIOS           |
| 5. CONSULTAS         |
| 6. LISTADO DE AGENDA |
| 7. FIN               |
| ESCOGER OPCIÓN       |

A continuación tenemos el algoritmo:

Algoritmo SISTEMA DE AGENDA ARREGLOS DE REGISTROS

1. Declarar

Tipos

regAmigo: Registro

    nombre: Cadena[30]

    dirección: Cadena[30]

    teléfono: Cadena[9]

    fax: Cadena[9]

```
email: Cadena[30]
FinRegistro
Variables
agenda: Arreglo[20] regAmigo
nom: Cadena[30]
i, n, totPer, opcion: Entero
desea, seguro: Carácter
```

## 2. Función principal()

a. do

1. Imprimir el menú de opciones

| SISTEMA DE AGENDA                                                                                       |
|---------------------------------------------------------------------------------------------------------|
| 1. CREAR AGENDA<br>2. ALTAS<br>3. BAJAS<br>4. CAMBIOS<br>5. CONSULTAS<br>6. LISTADO DE AGENDA<br>7. FIN |
| ESCOGER OPCIÓN                                                                                          |

2. Leer opcion

3. switch opcion

- 1: crear()
- 2: altas()
- 3: bajas()
- 4: cambios()
- 5: consultas()
- 6: listado()

4. endswitch

b. while opcion != 7

c. Fin Función principal

## 3. Función crear()

a. for i=0; i<=19; i++

- 1. agenda[i].nombre = ""
- 2. agenda[i].direccion = ""
- 3. agenda[i].telefono = ""
- 4. agenda[i].fax = ""
- 5. agenda[i].email = ""

b. endfor

c. Fin Función crear

## 4. Función altas()

a. do

```
1. i = -1
2. do
   a. i = i + 1
3. while(agenda[i].nombre == "") OR (i > 19)
4. if i <= 19 then
   a. Solicitar datos del empleado
   b. Leer agenda[i].nombre, agenda[i].direccion,
      agenda[i].telefono, agenda[i].fax,
      agenda[i].email
5. else
   a. Imprimir "NO PROCEDE LA ALTA; NO HAY ESPACIO"
6. endif
7. Preguntar "¿Desea procesar otra alta(S/N)?"
8. Leer desea
   b. while desea == 'S'
   c. Fin Función altas
5. Función bajas()
a. do
   1. Solicitar nombre a dar de baja
   2. Leer nom
   3. i = -1
   4. do
      a. i = i + 1
   5. while(agenda[i].nombre == nom) OR (i > 19)
   6. if i <= 19 then
      a. Imprimir agenda[i].direccion, agenda[i].
         telefono, agenda[i].fax,
         agenda[i].email
      b. Preguntar si está seguro
      c. Leer seguro
      d. if seguro == 'S' then
         1. agenda[i].nombre = ""
         2. agenda[i].direccion = ""
         3. agenda[i].telefono = ""
         4. agenda[i].fax = ""
         5. agenda[i].email = ""
      e. endif
   7. else
      a. Imprimir "NO PROCEDE LA BAJA; NO EXISTE"
   8. endif
   9. Preguntar "¿Desea procesar otra baja(S/N)?"
  10. Leer desea
   b. while desea == 'S'
```

```
c. Fin Función bajas
6. Función cambios()
    a. do
        1. Solicitar nombre a hacer cambios
        2. Leer nom
        3. i = -1
        4. do
            a. i = i + 1
        5. while(agenda[i].nombre == nom) OR (i > 19)
        6. if i <= 19 then
            a. Imprimir agenda[i].direccion, agenda[i].
                telefono, agenda[i].fax,
                agenda[i].email
            b. Preguntar dato a cambiar(1,2,3,4,0=Fin)
            c. Leer opcion
            d. while (opcion > 0) AND (opcion < 5)
                1. switch opcion
                    1. Leer agenda[i].direccion
                    2. Leer agenda[i].telefono
                    3. Leer agenda[i].fax
                    4. Leer agenda[i].email
                2. endswitch
                3. Preguntar dato a cambiar(1,2,3,4,0=Fin)
                4. Leer opcion
            e. endwhile
        7. else
            a. Imprimir "NO PROCEDE EL CAMBIO; NO EXISTE"
        8. endif
        9. Preguntar "¿Desea procesar otro cambio(S/N)?"
        10. Leer desea
    b. while desea == 'S'
    c. Fin Función cambios
7. Función consultas()
    a. do
        1. Solicitar nombre a consultar
        2. Leer nom
        3. i = -1
        4. do
            a. i = i + 1
        5. while(agenda[i].nombre == nom) OR (i > 19)
        6. if i <= 19 then
            a. Imprimir agenda[i].direccion, agenda[i].
                telefono, agenda[i].fax,
```

```

agenda[i].email
7. else
    a. Imprimir "NO PROCEDE LA CONSULTA; NO EXISTE"
8. endif
9. Preguntar "¿Desea procesar otra consulta(S/N) ?"
10. Leer desea
    b. while desea == 'S'
        c. Fin Función consultas
8. Función listado()
    a. Imprimir "LISTADO DE AGENDA"
        "NOMBRE DIRECCIÓN TELÉFONO FAX EMAIL"
        "-----"
    b. totPer = 0
    c. for i=0; i<=19; i++
        1. if agenda[i].nombre != "" then
            a. Imprimir agenda[i].nombre, agenda[i].
                dirección, agenda[i].
                teléfono, agenda[i].
                fax, agenda[i].email
        b. totPer = totPer + 1
        2. endif
    d. endfor
    e. Imprimir totPer
    f. Fin Función listado
Fin

```



En la zona de descarga de la Web del libro, está disponible:  
Programa en C: C803.C

*Explicación:*

agenda es un arreglo de 20 elementos de tipo regAmigo, el cual tiene los datos nombre, dirección, teléfono, fax, email.

Las funciones:

- crear() permite crear en nulos los elementos del arreglo de registros de la agenda.
- altas() permite dar altas de registros de personas.
- bajas() permite dar bajas de registros de personas.
- cambios() permite hacer cambios en datos de personas.
- consultas() permite hacer consultas en datos de personas.
- listado() permite obtener un listado con los datos de las personas de la agenda.

## 8.3 Operaciones para el manejo de archivos en seudocódigo

La estructura de datos archivo está formada por un conjunto de elementos (registros) de un mismo tipo de datos, y se almacena en un dispositivo auxiliar de almacenamiento como discos magnéticos, duros, flexibles, dispositivos ópticos, cintas, etc. Es decir, los datos que se registran en archivos quedan almacenados en forma permanente.

### Definición de archivos

Cuando se define un archivo, no se indica el tamaño del mismo en número de componentes, se inicia con el byte 0 (cero), cuando se escribe sobre él, se crea el siguiente byte y así sucesivamente, la única limitante en el número de componentes (bytes) del archivo, es la capacidad del dispositivo de almacenamiento.

Un archivo, como cualquier otra estructura de datos, debe definirse en la parte de declaraciones como una variable, con el siguiente formato:

```
nomArchivo : Archivo de Tipo de dato
```

*En donde:*

|              |                                                              |
|--------------|--------------------------------------------------------------|
| nomArchivo   | Es el nombre mediante el que se identificará.                |
| Archivo de   | Es la palabra reservada que indica que es un archivo.        |
| Tipo de dato | Es el tipo de datos del cual será cada elemento del archivo. |

### Archivos como colección de datos simples

Se puede manejar un archivo como colección de datos simples, como Entero, Real, etcétera.

Ejemplo: Archivos de números enteros y reales.

```
Declarar
    Variables
        numsEnteros: Archivo de Entero
        numsReales: Archivo de Real
```

numsEnteros es un archivo de números enteros, es decir, cada elemento será de tipo entero.

numsReales es un archivo de números reales, es decir, cada elemento será de tipo real.

Otro ejemplo:

```
Declarar
    Tipos
        Cadena30: Cadena[30]
    Variables
        nombres: Archivo de Cadena30
```

nombr es un archivo de nombres, donde cada elemento será un tipo Cadena30, es decir, una cadena de 30 caracteres que contendrá el nombre de una persona.

### Archivos como colección de registros

Se puede manejar un archivo como colección de registros, si se quiere manejar el archivo de Empleados que se explicó anteriormente, el cual está compuesto por los registros de los Empleados donde cada registro está conformado por los datos: numero, nombre, depto, puesto y sueldo, se debe:

1. Declarar un nuevo tipo de datos con el registro del empleado:

```
Declarar
    Tipos
        persona: Registro
            numero: Entero
            nombre: Cadena[30]
            depto: Entero
            puesto: Entero
            sueldo: Real
        FinRegistro
```

2. Declarar la variable archivo:

```
VARIABLES
    empleados: Archivo de persona
```

empleados es un archivo compuesto por una colección de datos de tipo persona cada uno, el cual, a su vez, es un tipo registro que tiene los datos numero, nombre, depto, puesto y sueldo.

### Creación de archivos (Crear)

Cuando se va a utilizar un archivo, primero es necesario crearlo, a fin de que le sea asignado un lugar en el dispositivo físico de acuerdo con los datos que contendrá, y asignar algunas identificaciones que requiere la máquina. En caso de crear un archivo que ya existe, significaría destruir el existente y comenzar de nuevo. En el momento en que se crea un archivo, no contiene ningún componente, en consecuencia, lo único que se puede hacer es escribir en el mismo. El apuntador se coloca en el primer componente (byte).

*Formato:*

```
Crear (nomArchivo, nomFisico, organización, modo)
```

*En donde:*

|            |                                                                                                                 |
|------------|-----------------------------------------------------------------------------------------------------------------|
| crear      | Identifica la operación de crear inicialmente el archivo.                                                       |
| nomArchivo | Es el nombre del archivo que se creará, mismo quedebé definirse como una variable de tipo archivo (Archivo de). |

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| nomFisico    | Es el nombre físico (en el disco) del archivo que se creará, debe incluir el path, ejemplo: "C:\AREMPL.DAT".                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| organización | Es el tipo de organización del archivo; secuencial o directo.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| modo         | El modo puede ser: <ul style="list-style-type: none"><li>w Indica que es sólo de escritura, en secuencial o directo, se usa al crear el archivo, el apuntador se coloca en el primer componente, que está vacío, por ello sólo se puede escribir.</li><li>r Indica que es sólo de lectura, para secuencial, el apuntador se coloca en el primer componente y sólo se puede leer.</li><li>a Indica que se va a expandir el archivo, para secuencial, el apuntador se coloca en el siguiente componente después del fin del archivo, para agregar.</li><li>rw Indica que es de escritura/lectura, para directo, el apuntador se coloca en el primer componente y se puede leer o escribir, o se puede mover el apuntador para posteriormente leer o escribir.</li></ul> |

Ejemplo:

```
Crear (empleados, "C:\AREMPL.DAT", secuencial, w)
```

Explicación:

Se crea el archivo empleados con organización secuencial, su nombre físico en el disco es "C:\AREMPL.DAT", en modo escritura, sólo se puede escribir en él.

### Abrir archivos (Abrir)

Esta operación permite preparar (abrir) un archivo existente para ser utilizado. El apuntador se coloca en el primer componente y permite leer y escribir sobre el archivo, es decir, se pueden hacer altas, bajas, cambios, consultas, etc. En caso de que el archivo no exista habrá error.

Formato:

```
Abrir (nomArchivo, nomFisico, organización, modo)
```

En donde:

|            |                                                              |
|------------|--------------------------------------------------------------|
| Abrir      | Identifica la operación de apertura de archivo.              |
| nomArchivo | Es el nombre que identifica al archivo.                      |
| nomFisico  | Es el nombre físico (en el disco) del archivo que se abrirá. |
| modo       | El modo puede ser como se indica en el punto anterior.       |

*Ejemplo:*

```
Abrir (empleados, "C:\AREMPL.DAT", secuencial, r)
```

*Explicación:*

Se abre el archivo empleados con organización secuencial, su nombre físico en el disco es "C:\AREMPL.DAT", se abre en modo lectura; sólo se puede leer de él.

### Expansión de un archivo (Añadir)

Cuando un archivo ya existe y tiene componentes registrados, se puede abrir con el modo añadir; se abre y el apuntador se coloca en el inicio de un nuevo componente después del último, y a partir de ahí se pueden escribir o registrar nuevos componentes, ejemplo:

```
Abrir (empleados, "C:\AREMPL.DAT", secuencial, a)
```

*Explicación:*

Se abre el archivo empleados con organización secuencial, su nombre físico en el disco es "C:\AREMPL.DAT" y se abre para expansión; sólo se puede imprimir en él.

### Escritura de registros (Imprimir)

La operación de escribir o imprimir un elemento, consiste en grabarlo en el componente donde esté ubicado el apuntador del archivo en ese momento; es necesario controlar la posición donde se encuentra el apuntador del archivo. Después de imprimir, el apuntador avanza al siguiente componente del archivo. En caso de que el siguiente componente no exista, se crea el esqueleto del mismo, hacia el cual avanza el apuntador.

*Formato:*

```
Imprimir (nomArchivo, datos)
```

*En donde:*

|            |                                                                                                                                                                                             |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Imprimir   | Identifica la acción de escritura.                                                                                                                                                          |
| nomArchivo | Es el nombre de identificación del archivo en el que se hará la escritura.                                                                                                                  |
| datos      | Es el nombre de identificación de la variable que contiene los datos que se grabarán en el componente actual del archivo, mismos que debieron ser previamente alimentados en esta variable. |

*Ejemplo:*

```
Algoritmo LEE IMPRIME EMPLEADO
1. Declarar
    Tipos
```

```
persona: Registro
    numero: Entero
    nombre: Cadena[30]
    depto: Entero
    puesto: Entero
    sueldo: Real
FinRegistro

Variables
regEmpleado: persona
empleados: Archivo de persona
2. Crear (empleados, "C:\AREMPL.DAT", secuencial, w)
3. Solicitar datos del empleado
4. Leer regEmpleado.numero, regEmpleado.nombre,
   regEmpleado.depto, regEmpleado.puesto
   regEmpleado.sueldo
5. Imprimir (empleados, regEmpleado)
--
--
N. Fin
```

*Explicación:*

1. Se declara el tipo persona, y las variables:  
regEmpleado como un registro tipo persona  
empleados como Archivo de tipo persona
  2. Se crea el archivo empleados que en el disco se llama “AREMPL.DAT”, con organización secuencial y en modo escritura
  3. Se solicitan los datos del empleado
  4. Se leen los datos del empleado en cada uno de los elementos de la variable regEmpleado, en regEmpleado.numero, regEmpleado.nombre, regEmpleado.depto, regEmpleado.puesto, regEmpleado.sueldo
  5. Se imprime el contenido de regEmpleado en el archivo empleados
- --
- N. Fin

**Nota:** Observe que regEmpleado es una variable que reside en la memoria principal, que se usa para dos cosas:

1. Leer los datos que se solicitan uno por uno en forma interactiva.
2. Llevarlos al archivos mediante Imprimir (empleados, regEmpleado), así se imprimen de la memoria hacia el archivo.

### Lectura de registros (Leer)

Esta operación permite leer los datos del elemento donde se encuentra ubicado actualmente el apuntador del archivo. Estos datos se recuperan (se colocan) en una variable. Después de la lectura, el apuntador avanza al siguiente componente. Cuando se lee el último componente el apuntador avanza al fin del archivo (EOF). Si trata de leer cuando el apuntador está en el fin del archivo (EOF) habrá error.

*Formato:*

```
Leer (nomArchivo, datos)
```

*En donde:*

|            |                                                                                                         |
|------------|---------------------------------------------------------------------------------------------------------|
| Leer       | Identifica la operación de lectura.                                                                     |
| nomArchivo | Es el nombre de identificación del archivo.                                                             |
| datos      | Es el nombre de identificación de la variable en la cual se van a recuperar (colocar) los datos leídos. |



Observe que regEmpleado es una variable que reside en la memoria principal, que se usa para dos cosas:

1. Leer los datos del archivo y colocarlos en la misma, es decir, llevarlos del archivo a la memoria mediante `Leer(empleados, regEmpleado)`.
2. En la memoria (en regEmpleado) se utilizan uno por uno para calcular, imprimir, etc.

**Ejemplo: Utilizando las definiciones del ejemplo anterior.**

```
Leer (empleados, regEmpleado)
```

Se lee el contenido del componente actual del archivo empleados y se coloca en la variable regEmpleado.

### Localización de componente (Encontrar)

Es una operación que permite colocar o mover el apuntador del archivo en un determinado componente (byte). Esta operación sólo aplica para archivos con organización directa.

*Formato:*

```
Encontrar (nomArchivo, n)
```

*En donde:*

|            |                                                                                                                                                                                                                            |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Encontrar  | Identifica la acción de localizar un componente.                                                                                                                                                                           |
| nomArchivo | Es el nombre del archivo en el que se hará.                                                                                                                                                                                |
| n          | Es el número del componente (byte) en el que se colocará el apuntador del archivo. Puede ser una variable, constante o expresión de tipo entero, entre 0 y n-1, donde n es el tamaño del archivo en número de componentes. |

**Ejemplo:**

Si se desea leer el elemento 50 del archivo empleados:

```
Encontrar (empleados, 50)
Leer (empleados, RegEmpleado)
```

Si el archivo empleados tiene  $n$  bytes y se desea expandir, se hace moviendo el apuntador más allá del byte  $n$ , así:

Encontrar (empleados, n)

**Observación:** Como el último byte es el  $n-1$ , el nuevo que se agregará es el  $n$ .

## Cerrar archivos (Cerrar)

Esta operación permite liberar (cerrar) un archivo que esté abierto; todo archivo debe cerrarse al terminarse de usar.

*Formato:*

Cerrar (nomArchivo)

*En donde:*

Cerrar Identifica la operación de cierre del archivo.

**nomArchivo** Es el nombre del archivo que se cerrará.

### Ejemplo:

Cerrar (empleados)

## Funciones para el proceso de archivos

|                  |                                                                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Eof (nomArchivo) | Es una función booleana que proporciona un valor verdadero (true), si el apuntador está al final del archivo; en caso contrario proporciona el valor falso (false). |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**TamañoAr (nomArchivo)** Proporciona el número de componentes (bytes) que tiene el archivo.

**PosicionAr (nomArchivo)**  
Proporciona el número del componente en el que se encuentra actualmente el apuntador del archivo.

**Renombra (nomArchivo, "nuevoNombre")**  
Permite renombrar o cambiar el nombre de un archivo.

BorrarAr (nomArchivo)

**TruncaAr (nomArchivo)** Permite truncar el tamaño del archivo suprimiendo los elementos después de la posición actual del apuntador.

## Funciones de directorio

Mkdir Crea un subdirectorio

*Formato:*

```
MkDir("nombreDir")
```

*En donde:*

|           |                                 |
|-----------|---------------------------------|
| MkDir     | Identifica la función.          |
| nombreDir | Es el nombre del directorio.    |
| RmDir     | Remueve un subdirectorio vacío. |

*Formato:*

```
RmDir("nombreDir")
```

*En donde:*

|           |                              |
|-----------|------------------------------|
| RmDir     | Identifica la función.       |
| nombreDir | Es el nombre del directorio. |
| ChDir     | Cambia de directorio.        |

*Formato:*

```
ChDir("nombreDir")
```

*En donde:*

|           |                                                          |
|-----------|----------------------------------------------------------|
| ChDir     | Identifica la función.                                   |
| nombreDir | Es el nombre del directorio.                             |
| GetDir    | Proporciona el directorio actual del drive especificado. |

*Formato:*

```
GetDir(drive, nomDrive)
```

*En donde:*

|           |                                                                                         |
|-----------|-----------------------------------------------------------------------------------------|
| GetDir    | Identifica la función.                                                                  |
| drive:    | Indica el número de drive, 0 es el actual, 1 el A, 2 el B, etcétera.                    |
| nomDrive: | Es una variable tipo string donde se almacenará la descripción del directorio indicado. |

## 8.4 Proceso de un archivo secuencial

Cuando se va a utilizar un archivo con organización secuencial, primero debe crearse el archivo, enseguida se le hacen altas; posteriormente puede agrandarse añadiéndole más registros con las nuevas altas que se generen. El último

registro se recomienda que sea una marca o “centinela” que indique el fin del archivo; sin embargo, en algunos lenguajes no se facilita hacerlo, por ello, en este libro no utilizaremos dicho concepto. Un archivo con organización secuencial se puede actualizar con altas, bajas y cambios, mediante la creación de otro archivo que contenga estos movimientos. Posteriormente, se hace un proceso especial de actualización en el que se crea un nuevo archivo actualizado. Finalmente el archivo se utiliza para leer sus registros y emitir reportes o listados de sus datos. A continuación se explica en qué consiste cada uno de dichos procesos.

### Creación

Esta operación permite crear el archivo dejándolo listo para recibir altas de registros. Se dan de alta registros. Después de dar todas las altas, se debe cerrar el archivo y al final del archivo se coloca una marca de fin del archivo. Este proceso se hace una sola vez.

### Expansión

Un archivo secuencial existente, como el creado con el proceso de creación anterior, puede agrandarse agregándole nuevos registros a partir del último. Se le agregan los nuevos registros (altas) a partir de la marca de fin del archivo, y al final, al archivo se le coloca de nuevo la marca de fin del archivo. Este proceso se hace tantas veces como necesidad se tenga de agregar nuevos registros al archivo.

### Obtención de reportes

Un archivo al que ya se le dieron de alta registros, se puede utilizar para emitir información en forma de reportes o listados de sus datos. A partir de un archivo secuencial, el procedimiento es el siguiente: abrir el archivo, leer el primer registro, hacer los cálculos necesarios y enlistarlo, enseguida leer el siguiente registro, hacer cálculos y enlistarlo, y así sucesivamente hasta encontrar el fin del archivo. Ejemplo.

Obtener un reporte del archivo de empleados con el siguiente formato:

| CATÁLOGO DE EMPLEADOS |                |        |        |            |
|-----------------------|----------------|--------|--------|------------|
| NÚMERO                | NOMBRE         | DEPTO. | PUESTO | SUELDO     |
| 99                    | XXXXXXXXXXXXXX | 99     | 99     | 99,999.99  |
| 99                    | XXXXXXXXXXXXXX | 99     | 99     | 99,999.99  |
| - - -                 |                |        |        |            |
| - - -                 |                |        |        |            |
| 99                    | XXXXXXXXXXXXXX | 99     | 99     | 99,999.99  |
| TOTAL                 | 999 EMPLEADOS  |        |        | 999,999.99 |

Otro problema:

Con base en el archivo de empleados, emitir el reporte nómina quincenal:

| NÓMINA QUINCENAL |                      |            |            |            |
|------------------|----------------------|------------|------------|------------|
| NÚMERO           | NOMBRE               | S. BRUTO   | IMPUESTO   | S. NETO    |
| 99               | XXXXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99  |
| 99               | XXXXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99  |
| ---              |                      |            |            |            |
| 99               | XXXXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99  |
| TOTALES          | 999 EMPLEADOS        | 999,999.99 | 999,999.99 | 999,999.99 |

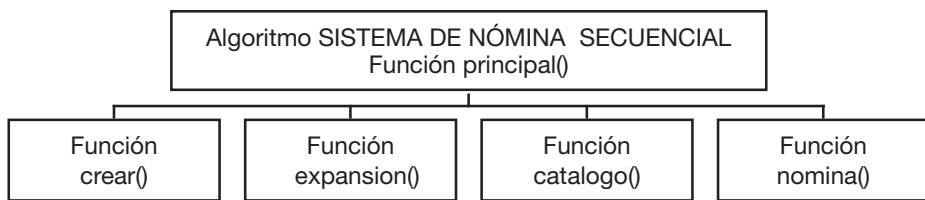
S. BRUTO es el sueldo que trae en el archivo dividido entre 2, porque en el mes hay dos quincenas.

IMPUESTO es el 5% sobre el excedente del bruto sobre el salario mínimo quincenal.

S. NETO es el S. BRUTO menos el IMPUESTO.

Ejemplo:

A continuación se elabora un algoritmo que permite procesar un archivo con organización secuencial, para aplicar los conceptos antes decritos. En la función principal() se ofrece un menú con las opciones: Crear el archivo, hacerle expansión al archivo, emitir el reporte catálogo de empleados y emitir el reporte nómina quincenal.



Algoritmo SISTEMA DE NÓMINA SECUENCIAL

1. Declarar

Tipos

persona: Registro

    numero: Entero

    nombre: Cadena[30]

    depto: Entero

    puesto: Entero

    sueldo: Real

FinRegistro

Variables

regEmpleado: persona

empleados: Archivo de persona

opcion: Entero

desea: Carácter

2. Función principal()

a. do

1. Imprimir MENÚ

| SISTEMA DE NÓMINA |                    |
|-------------------|--------------------|
| 1.                | CREAR ARCHIVO      |
| 2.                | EXPANSIÓN          |
| 3.                | CATÁLOGO EMPLEADOS |
| 4.                | NÓMINA QUINCENAL   |
| 5.                | FIN                |
| ESCOGER OPCIÓN    |                    |

2. Leer opcion

3. switch opcion

1: crear()

2: expansion()

3: catalogo()

4: nomina()

4. endswitch

b. while opcion != 5

c. Fin Función principal

3. Función crear()

a. Crear (empleados, "C:\AREMSE.DAT", secuencial, w)

b. do

1. Imprimir PANTALLA de captura

| ALTAS ARCHIVO DE EMPLEADOS |  |
|----------------------------|--|
| NÚMERO:                    |  |
| NOMBRE:                    |  |
| DEPARTAMENTO:              |  |
| PUESTO:                    |  |
| SUELDO:                    |  |
| ¿OTRO EMPLEADO (S/N) ?     |  |

2. Leer regEmpleado.numero, regEmpleado.nombre,  
regEmpleado.depto, regEmpleado.puesto,  
regEmpleado.sueldo

3. Imprimir (empleados, regEmpleado)

4. Leer desea

c. while desea == 'S'

d. Cerrar (empleados)

e. Fin Función crear

4. Función expansion()
- Abrir (empleados, "C:\AREMSE.DAT", secuencial, a)
  - do
    - Imprimir PANTALLA de captura

| ALTAS ARCHIVO DE EMPLEADOS |  |
|----------------------------|--|
| NÚMERO:                    |  |
| NOMBRE:                    |  |
| DEPARTAMENTO:              |  |
| PUESTO:                    |  |
| SUELDO:                    |  |
| ¿OTRO EMPLEADO (S/N) ?     |  |

2. Leer regEmpleado.numero, regEmpleado.nombre, regEmpleado.depto, regEmpleado.puesto, regEmpleado.sueldo
3. Imprimir (empleados, regEmpleado)
4. Leer desea
  - while desea == 'S'
  - Cerrar (empleados)
  - Fin Función expansion
5. Función catalogo()
  - Declarar
 

Variables

totEmp : Entero

totSueldos : Real
  - Abrir (empleados, "C:\AREMSE.DAT", secuencial, r)
  - totEmp = 0
  - totSueldos = 0
  - Imprimir encabezado
  - Leer (empleados, regEmpleado)
  - while NOT(Eof(empleados))
    - Imprimir regEmpleado.numero, regEmpleado.nombre, regEmpleado.depto, regEmpleado.puesto, regEmpleado.sueldo
    - totEmp = totEmp + 1
    - totSueldos = totSueldos + regEmpleado.sueldo
    - Leer (empleados, regEmpleado)
  - endwhile
  - Imprimir totEmp, totSueldos
  - Cerrar (empleados)
  - Fin Función catalogo

```
6. Función nomina()
a. Declarar
    Variables
        totEmp: Entero
        bruto, impuesto, neto, totBruto,
        totNeto, totImpuesto, salMin: Real
b. Abrir (empleados, "C:\AREMSE.DAT", secuencial, r)
c. totEmp = 0; totBruto = 0; totNeto = 0; totImpuesto = 0
d. Solicitar Salario mínimo quincenal
e. Leer salMin
f. Imprimir encabezado
g. Leer (empleados, regEmpleado)
h. while NOT(Eof(empleados))
    1. bruto = regEmpleado.sueldo / 2
    2. if bruto > salMin then
        a. impuesto = (bruto - salMin) * 0.05
    3. else
        a. impuesto = 0
    4. endif
    5. neto = bruto - impuesto
    6. Imprimir regEmpleado.numero, regEmpleado.
        nombre, bruto, impuesto, neto
    7. totEmp = totEmp + 1
        totBruto = totBruto + bruto
        totNeto = totNeto + neto
        totImpuesto = totImpuesto + impuesto
    8. Leer (empleados, regEmpleado)
i. endwhile
j. Imprimir totEmp, totBruto, totImpuesto, totNeto
k. Cerrar (empleados)
l. Fin Función nomina
Fin
```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C804.C



*Explicación:*

1. Se hacen declaraciones globales

Se declara el tipo

persona que es un registro con los elementos: numero,  
nombre, depto, puesto, sueldo

Se declaran variables globales

regEmpleado que es un registro tipo persona

empleados que es un Archivo de persona

opcion: Entero

desea: Carácter

2. Función principal()

a. Inicia ciclo do

1. Imprimir MENÚ y solicita la opcion

2. Se lee opcion

3. Inicia switch opcion

Si opcion == 1: Llama Función crear()

Si opcion == 2: Llama Función expansion()

Si opcion == 3: Llama Función catalogo()

Si opcion == 4: Llama Función nomina()

4. Fin switch

b. Fin (do while) mientras opcion != 5

c. Fin Función

3. Función crear()

a. Crea archivo empleados como secuencial

b. Inicia ciclo do

1. Imprime PANTALLA de captura donde solicita los datos

2. Lee en regEmpleado.numero, regEmpleado.nombre,  
regEmpleado.depto, regEmpleado.puesto,  
regEmpleado.sueldo

3. Imprime en empleados el registro regEmpleado

4. Lee en desea en respuesta a ¿otro?

c. Fin ciclo (do while) mientras desea == 'S'

d. Cierra el archivo empleados

e. Fin Función

4. Función expansion()

a. Abre el archivo empleados como Secuencial

b. Coloca el apuntador del archivo en el componente donde se  
encuentra el "centinela de fin de archivo"

c. Inicia ciclo do

1. Imprime PANTALLA de captura donde solicita los datos

2. Lee en regEmpleado.numero, regEmpleado.nombre,  
regEmpleado.depto, regEmpleado.puesto,  
regEmpleado.sueldo

3. Imprime en empleados el registro regEmpleado
  4. Lee en desea
  - d. Fin ciclo (do while) mientras desea == 'S'
  - e. Cierra el archivo empleados
  - f. Fin Función
5. Función catalogo()
    - a. Se declaran variables para los cálculos locales
    - b. Abre el archivo empleados como secuencial
    - c. Inicia totEmp en 0  
Inicia totSueldos en 0
    - d. Imprime encabezado
    - e. Lee el primer registro de empleados en regEmpleado
    - f. Inicia ciclo while NOT(Eof(empleados))
      1. Imprime los datos del registro leído
      2. Incrementa totEmp en 1
      3. Incrementa totSueldos con regEmpleado.sueldo
      4. Lee siguiente registro de empleados en regEmpleado
    - g. Fin while
    - h. Imprime totEmp, totSueldos
    - i. Cierra archivo empleados
    - j. Fin Función
    6. Función nomina()
      - a. Se declaran variables para los cálculos locales
      - b. Abre el archivo empleados como secuencial
      - c. Inicia en cero los totales generales
      - d. Sigue la ejecución
      - e. Lee en salMin
      - f. Imprime encabezado
      - g. Lee el primer registro de empleados en regEmpleado
      - h. Inicia ciclo while NOT(Eof(empleados))
        1. Calcula bruto
        2. Si bruto > salMin Entonces
          - a. Calcula impuesto = (bruto – salMin) \* 0.05
        3. Si no else
          - a. Calcula impuesto = 0
        4. Fin if

5. Calcula neto
  6. Imprime regEmpleado.numero, regEmpleado.nombre, bruto, impuesto, neto
  7. Incrementa totEmp con 1  
Incrementa totBruto con bruto  
Incrementa totNeto con neto  
Incrementa totImpuesto con impuesto
  8. Lee siguiente registro de empleados en regEmpleado
  - i. Fin while
  - j. Imprime totEmp, totBruto, totImpuesto, totNeto
  - k. Cierra archivo empleados
  - l. Fin Función
- Fin del algoritmo

### Actualización con altas, bajas y cambios

Cuando se tiene un archivo secuencial, como el de empleados, y se desea actualizar con altas, bajas y cambios, es necesario:

1. Crear otro archivo con los movimientos, el cual contendrá, además de los datos del archivo de empleados, el tipo de movimiento que puede ser alta, baja o cambio (A,B,C).
2. Procesar los dos archivos anteriores; este proceso hace que los movimientos se reflejen en el de empleados, y generan un nuevo archivo, el de empleados actualizado.

#### Ejemplo:

Si tenemos los archivos con los datos siguientes:

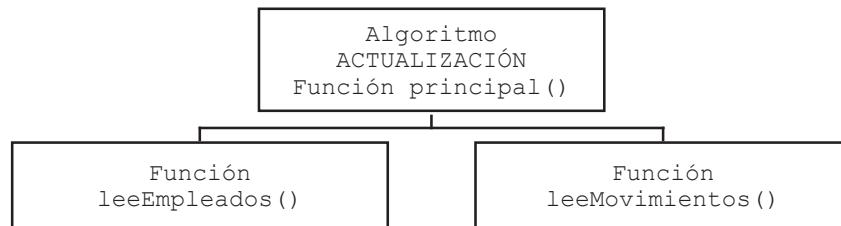
| EMPLEADOS |       | MOVIMIENTOS |           |       |
|-----------|-------|-------------|-----------|-------|
| NÚMERO    | DATOS | NÚMERO      | TIPO MOV. | DATOS |
| 2         | ----- | 1           | A         | ----- |
| 4         | ----- | 3           | A         | ----- |
| 7         | ----- | 4           | C         | ----- |
| 10        | ----- | 5           | A         | ----- |
| 15        | ----- | 7           | B         | ----- |
| 16        | ----- | 8           | A         | ----- |
| 20        | ----- | 15          | C         | ----- |
|           |       | 16          | B         | ----- |
|           |       | 25          | A         | ----- |
|           |       | 27          | A         | ----- |

Se genera el archivo:

| EMPLEADOS ACTUALIZADO |       |
|-----------------------|-------|
| NÚMERO                | DATOS |
| 1                     | ----- |
| 2                     | ----- |
| 3                     | ----- |
| 4                     | ----- |
| 5                     | ----- |
| 8                     | ----- |
| 10                    | ----- |
| 15                    | ----- |
| 20                    | ----- |
| 25                    | ----- |
| 27                    | ----- |

Suponiendo que ya ha sido creado el archivo de movimientos y que contiene registros con los movimientos. A continuación, se presenta un algoritmo que realiza el proceso de actualización.

Estructura general del algoritmo:



#### Algoritmo ACTUALIZACIÓN

1. Declarar

Tipos

```

persona: Registro
    numero: Entero
    nombre: Cadena[30]
    depto: Entero
    puesto: Entero
    sueldo: Real
FinRegistro

movimiento: Registro
    numero: Entero
    tipoMov: Carácter
    nombre: Cadena[30]
    depto: Entero
    puesto: Entero
    sueldo: Real
FinRegistro
  
```

## Variables

```
regEmpleado: persona
RegMovto: movimiento
RegActual: persona
empleados: Archivo de persona
movimientos: Archivo de movimiento
empleadosActual: Archivo de persona

2. Función principal()
    a. Abrir (empleados, "C:\AREMSE.DAT", secuencial, r)
    b. Abrir (movimientos, "C:\ARMOVE.DAT", secuencial,
r)
    c. Crear (empleadosActual, "C:\AREMAC.DAT",
secuencial, w)
    d. leeEmpleados()
    e. leeMovimientos()
    f. while (NOT(Eof(empleados))) AND
(NOT(Eof(movimientos)))
        1. if regEmpleado.numero == regMovto.numero then
            a. if regMovto.tipoMov == 'C' then
                1. regActual.numero = regMovto.numero,
                regActual.nombre = regMovto.nombre,
                regActual.depto = regMovto.depto,
                regActual.puesto = regMovto.puesto,
                regActual.sueldo = regMovto.sueldo
                2. Imprimir (empleadosActual, regActual)
            b. else
                1. if regMovto.tipoMov == 'A' then
                    a. Imprimir (empleadosActual, regEmpleado)
                2. endif
            c. endif
            d. leeEmpleados()
            e. leeMovimientos()
        2. else
            a. if regEmpleado.numero > regMovto.numero then
                1. if regMovto.tipoMov == 'A' then
                    a. regActual.numero = regMovto.numero,
                    regActual.nombre = regMovto.nombre,
                    regActual.depto = regMovto.depto,
                    regActual.puesto = regMovto.puesto,
                    regActual.sueldo = regMovto.sueldo
                    b. Imprimir (empleadosActual, regActual)
                2. endif
            3. leeMovimientos()
```

```

b. else
    1. Imprimir (empleadosActual, regEmpleado)
    2. leeEmpleados()
c. endif
3. endif
g. endwhile
h. Cerrar (empleados)
i. Cerrar (movimientos)
j. Cerrar (empleadosActual)
k. Fin Función principal
3. Función leeEmpleados()
    a. Leer (empleados, regEmpleado)
    b. if Eof(empleados) then
        1. do
            a. if regMovto.tipoMov == 'A' then
                1. regActual.numero = regMovto.numero,
                   regActual.nombre = regMovto.nombre,
                   regActual.depto = regMovto.depto,
                   regActual.puesto = regMovto.puesto,
                   regActual.sueldo = regMovto.sueldo
                2. Imprimir (empleadosActual, regActual)
            b. endif
            c. Leer (movimientos, regMovto)
        2. while (NOT(Eof(movimientos)))
            c. endif
            d. Fin Función leeEmpleados
4. Función leeMovimientos()
    a. Leer (movimientos, regMovto)
    b. if Eof(movimientos) then
        1. do
            a. Imprimir (empleadosActual, regEmpleado)
            b. Leer (empleados, regEmpleado)
        2. while NOT(Eof(empleados))
            c. endif
            d. Fin Función leeMovimientos
Fin

```

**Explicación:**

1. Se hacen declaraciones globales

Se declaran los tipos:

persona es un registro que contiene los datos:

numero, nombre, depto, puesto, sueldo

movimiento es un registro que contiene los datos:

numero, tipoMov, nombre, depto, puesto, sueldo



**Nota:** Este algoritmo se le puede añadir como una función de actualización al primer algoritmo de este punto (Algoritmo PROCESA SECUENCIAL). También podría añadirse una función para crear el archivo de movimientos. Sin embargo, tiene una complejidad que es innecesario enfrentarla en este nivel, y es más práctico y actual manejarlo como se hace con el archivo directo (que se explica en el siguiente punto).

Se declaran las variables

```
regEmpleado tipo persona  
regMovto tipo movimiento  
regActual tipo persona  
empleados tipo Archivo de persona  
movimientos tipo Archivo de movimiento  
empleadosActual tipo Archivo de persona
```

2. Función principal()
  - a. Abre el archivo empleados como secuencial
  - b. Abre el archivo movimientos como secuencial
  - c. Crea el archivo empleadosActual como secuencial
  - d. Llama leeEmpleados()
  - e. Llama leeMovimientos()
  - f. while NOT (Eof(empleados)) AND  
NOT (Eof(movimientos))
    1. if regEmpleado.numero == regMovto.numero then
      - a. if regMovto.tipoMov == 'C' then
        1. Coloca los datos del regMovto en regActual
        2. Imprime regActual en empleadosActual
      - b. else
        1. if regMovto.tipoMov == 'A' then
          - a. Imprime regEmpleado en empleadosActual
        2. endif
      - c. endif
      - d. Llama leeEmpleados()
      - e. Llama leeMovimientos()
    2. else
      - a. if regEmpleado.numero > regMovto.numero then
        1. if regMovto.tipoMov == 'A' then
          - a. Coloca los datos del regMovto en regActual
          - b. Imprime regActual en empleadosActual
        2. endif
        3. Llama leeMovimientos
      - b. else
        1. Imprime regEmpleado en empleadosActual
        2. Llamar leeEmpleados()
      - c. endif
    3. endif

- g. endwhile
  - h. Cerrar archivo empleados
  - i. Cerrar archivo movimientos
  - j. Cerrar archivo empleadosActual
  - k. Fin Función
3. Función leeEmpleados()
- a. Lee registro de empleados
  - b. if Eof(empleados) then
    - 1. do
      - a. if regMovto.tipoMov == 'A' then
        - 1. Coloca los datos del regMovto en regActual
        - 2. Imprime regActual en empleadosActual
      - b. endif
      - c. Leer (movimientos, regMovto)
    - 2. while NOT(Eof(movimientos))
  - c. endif
  - d. Fin Función
4. Función leeMovimientos()
- a. Lee registro de movimientos
  - b. if Eof(movimientos) then
    - 1. do
      - a. Imprime regEmpleado en EmpleadosActual
      - b. Lee registro de Empleados
    - 2. while NOT(Eof(empleados))
  - c. endif
  - d. Fin Función

Fin del algoritmo

### Emisión de reportes con cortes de control

Si suponemos que los registros del archivo están ordenados por departamento, es posible obtener reportes agrupados por departamento, es decir, que todos los Empleados del primer departamento aparezcan juntos, al inicio, con totales por departamento. Después los del siguiente departamento, y así sucesivamente hasta llegar al final, donde habrá un gran total que agrupe a todos los departamentos. Ejemplo:

Supongamos que el archivo de Empleados contiene los registros siguientes:

| Número | Nombre             | Dept. | Puesto | Sueldo |
|--------|--------------------|-------|--------|--------|
| 1      | MARICELA VILLA     | 1     | 1      | 1200   |
| 2      | EDILIA CAMACHO     | 1     | 2      | 1450   |
| 3      | DANIA CAMACHO      | 1     | 3      | 1750   |
| 4      | JUAN DE DIOS LÓPEZ | 2     | 1      | 1450   |
| 5      | CAROLINA OBESO     | 2     | 2      | 2300   |
| 6      | MANUEL MEZA        | 3     | 1      | 3000   |
| 7      | ARACELI LÓPEZ      | 3     | 2      | 1500   |

Emitir el catálogo de empleados con cortes de control nos arrojaría el siguiente reporte:

| CATÁLOGO DE EMPLEADOS |                    |        |        |          |
|-----------------------|--------------------|--------|--------|----------|
| NÚM.                  | NOMBRE             | DEPTO. | PUESTO | SUELDO   |
| 1                     | MARICELA VILLA     | 1      | 1      | 1200.00  |
| 2                     | EDILIA CAMACHO     | 1      | 2      | 1450.00  |
| 3                     | DANIA CAMACHO      | 1      | 3      | 1750.00  |
| TOTAL DEPTO. 1        | 3 EMPLEADOS        |        |        | 4400.00  |
| 4                     | JUAN DE DIOS LÓPEZ | 2      | 1      | 1450.00  |
| 5                     | CAROLINA OBESO     | 2      | 2      | 2300.00  |
| TOTAL DEPTO. 2        | 2 EMPLEADOS        |        |        | 3750.00  |
| 6                     | MANUEL MEZA        | 3      | 1      | 3000.00  |
| 7                     | ARACELI LÓPEZ      | 3      | 2      | 1500.00  |
| TOTAL DEPTO. 3        | 2 EMPLEADOS        |        |        | 4500.00  |
| TOTAL GENERAL         | 7 EMPLEADOS        |        |        | 12650.00 |

Los datos se imprimen agrupados por departamento. Por cada departamento se requieren dos datos totalizadores: total de empleados y total de sueldos. Al final se requiere un total general con los mismos datos que por departamento, sólo que agrupando a todos los departamentos. A continuación se elabora una función que emite el catálogo de empleados con cortes de control.

```

7. Función catalogoConCortes()
    a. Declarar
        Variables
            totEmp, totEmpDep, deptoProceso: Entero
            totSueldos, totSueldosDep: Real
    b. Abrir (empleados, "C:\AREMSE.DAT", secuencial, r)
    c. totEmp = 0
        totSueldos = 0
    d. Imprimir encabezado
    e. Leer (empleados, regEmpleado)

```

```
f. while NOT(Eof(Empleados))
    1. totEmpDep = 0 ; totSueldosDep = 0
    2. deptoProceso = regEmpleado.depto
    3. while (regEmpleado.depto==DeptоПроцесо)
        AND (NOT(Eof(Empleados)))
        a. Imprimir regEmpleado.numero,
            regEmpleado.nombre, regEmpleado.depto,
            regEmpleado.puesto, regEmpleado.sueldo
        b. totEmpDep = totEmpDep + 1
        c. totSueldosDep = totSueldosDep +
            regEmpleado.sueldo
        d. Leer (empleados, regEmpleado)
    4. endwhile
    5. Imprimir totEmpDep, totSueldosDep
    6. totEmp = totEmp + totEmpDep
        totSueldos = totSueldos + totSueldosDep
    g. endwhile
    h. Imprimir totEmp, totSueldos
    i. Cerrar (empleados)
    j. Fin Función catalogoConCortes
```

*Explicación:*

- a. Se declaran variables para los cálculos locales  
totEmp para contar el total de empleados global  
totEmpDep para contar el total de empleados por departamento  
deptoProceso para controlar el departamento que se está procesando  
totSueldos para el total de sueldos global  
totSueldosDep para el total de sueldos por departamento
- b. Abre el archivo empleados, secuencial para lectura
- c. Inicia totEmp en 0  
Inicia totSueldos en 0
- d. Imprime encabezado
- e. Lee primer registro de empleados
- f. while NOT(Eof(Empleados))
  1. Inicia totEmpDep en 0; y totSueldosDep en 0
  2. Coloca en deptoProceso el regEmpleado.depto
  3. while (regEmpleado.depto==DeptоПроцесо) AND (NOT(Eof(Empleados)))
    - a. Imprime regEmpleado.numero, regEmpleado.nombre,  
regEmpleado.depto, regEmpleado.puesto,  
regEmpleado.sueldo



Este algoritmo se le puede añadir como una función que emite el catálogo con cortes al primer algoritmo de este punto (Algoritmo PROCESA SECUENCIAL).

- b. Incrementa totEmpDep en 1
- c. Incrementa totSueldosDep con regEmpleado.sueldo
- d. Lee siguiente registro de empleados
- 4. endwhile
- 5. Imprime totEmpDep, totSueldosDep
- 6. Incrementa totEmp con totEmpDep  
Incrementa totSueldos con totSueldosDep
- g. endwhile
- h. Imprime totEmp, totSueldos
- i. Cerrar archivo (empleados)
- j. Fin Función

### Problema

Considerando el mismo archivo de Empleados, emitir el siguiente reporte con cortes de control:

| NÓMINA QUINCENAL |                    |        |         |           |         |  |
|------------------|--------------------|--------|---------|-----------|---------|--|
| NÚM.             | NOMBRE             | DEPTO. | S.BRUTO | IMPUUESTO | S. NETO |  |
| 1                | MARICELA VILLA     | 1      | 9999.99 | 9999.99   | 9999.99 |  |
| 2                | EDILIA CAMACHO     | 1      | 9999.99 | 9999.99   | 9999.99 |  |
| 3                | DANIA CAMACHO      | 1      | 9999.99 | 9999.99   | 9999.99 |  |
| TOTAL DEPTO. 1   | 3 EMPLEADOS        |        | 9999.99 | 9999.99   | 9999.99 |  |
| 4                | JUAN DE DIOS LÓPEZ | 2      | 9999.99 | 9999.99   | 9999.99 |  |
| 5                | CAROLINA OBESO     | 2      | 9999.99 | 9999.99   | 9999.99 |  |
| TOTAL DEPTO. 2   | 2 EMPLEADOS        |        | 9999.99 | 9999.99   | 9999.99 |  |
| 6                | MANUEL MEZA        |        | 9999.99 | 9999.99   | 9999.99 |  |
| 7                | ARACELI LÓPEZ      |        | 9999.99 | 9999.99   | 9999.99 |  |
| TOTAL DEPTO. 3   | 2 EMPLEADOS        |        | 9999.99 | 9999.99   | 9999.99 |  |
| TOTAL GENERAL    | 7 EMPLEADOS        |        | 9999.99 | 9999.99   | 9999.99 |  |

S. BRUTO es el sueldo que trae en el archivo dividido entre 2, porque en el mes hay dos quincenas.

IMPUUESTO es el 5% sobre el excedente del bruto sobre el salario mínimo quincenal.

S. NETO es el S. BRUTO menos el IMPUESTO.

Además se requieren totales por departamento y un total general de los datos: total de empleados, total de sueldo bruto, total de impuesto y total de sueldo neto. A continuación se elabora una función que emite la nómina quincenal con cortes de control.

#### 8. Función nominaConCortes()

a. Declarar

Variables

```
totEmp, totEmpDep, deptoProceso: Entero
bruto, impuesto, neto, totBruto, totBrutoDep,
```

```
        totImpuesto, totImpuestoDep, totNeto,
        totNetoDep, salMin: Real
b. Abrir (empleados, "C:\AREMSE.DAT", secuencial, r)
c. totEmp = 0; totBruto = 0; totImpuesto = 0; totNeto = 0
d. Solicitar Salario mínimo quincenal
e. Leer salMin
f. Imprimir encabezado
g. Leer (empleados, regEmpleado)
h. while NOT(Eof(Empleados))
    1. totEmpDep = 0 ; totBrutoDep = 0
        totImpuestoDep = 0; totNetoDep = 0
    2. deptoProceso = regEmpleado.depto
    3. while (regEmpleado.depto==DeptProceso) AND
        (NOT(Eof(Empleados)))
        a. bruto = regEmpleado.sueldo / 2
        b. if bruto > salMin then
            1. impuesto = (bruto - salMin) * 0.05
        c. else
            1. impuesto = 0
        d. endif
        e. neto = bruto - impuesto
        f. Imprimir regEmpleado.numero,
            regEmpleado.nombre, regEmpleado.depto,
            bruto, impuesto, neto
        g. totEmpDep = totEmpDep + 1
            totBrutoDep = totBrutoDep + bruto
            totImpuestoDep = totImpuestoDep + impuesto
            totNetoDep = totNetoDep + neto
        h. Leer (empleados, regEmpleado)
    4. endwhile
    5. Imprimir totEmpDep, totBrutoDep,
        totImpuestoDep, totNetoDep
    6. totEmp = totEmp + totEmpDep
        totBruto = totBruto + totBrutoDep
        totImpuesto = totImpuesto + totImpuestoDep
        totNeto = totNeto + totNetoDep
    i. endwhile
    j. Imprimir totEmp, totBruto, totImpuesto, totNeto
    k. Cerrar (empleados)
    l. Fin Función nominaConCortes
```



Este algoritmo se le puede añadir como una función que emite la nómina con cortes al primer algoritmo de este punto (Algoritmo PROCESA SECUENCIAL).

*Explicación:*

- a. Se declaran variables para los cálculos locales  
totEmp para contar el total de empleados global  
totEmpDep para contar el total de empleados por departamento  
deptoProceso para controlar el departamento que se está procesando  
totBruto para el total de sueldo bruto global  
totImpuesto para el total de impuesto global  
totNeto para el total de sueldo neto global  
totBrutoDep para el total de sueldo bruto por departamento  
totImpuestoDep para el total de impuesto por departamento  
totNetoDep para el total de sueldo neto por departamento
- b. Abre el archivo empleados, secuencial para lectura
- c. Inicia totEmp en 0; totBruto en 0; totImpuesto en 0; totNeto en 0
- d. Solicita Salario mínimo quincenal
- e. Lee salMin
- f. Imprime encabezado
- g. Lee primer registro de empleados
- h. while NOT(Eof(Empleados))
  1. Inicia totEmpDep en 0 ; totBrutoDep en 0  
totImpuestoDep en 0; totNetoDep en 0
  2. Coloca en deptoProceso el regEmpleado.depto
  3. while (regEmpleado.depto==DeptProceso) AND (NOT(Eof(Empleados)))
    - a. Calcula bruto = regEmpleado.sueldo / 2
    - b. Si bruto > salMin Entonces
      1. Calcula impuesto = (bruto - salMin) \* 0.05
    - c. else
      1. Calcula impuesto = 0
    - d. endif
    - e. Calcula neto = bruto – impuesto
    - f. Imprime regEmpleado.depto, regEmpleado.numero,  
regEmpleado.nombre, bruto, impuesto, neto
    - g. Incrementa totEmpDep con 1  
totBrutoDep con bruto  
totImpuestoDep con impuesto  
totNetoDep con neto
    - h. Lee siguiente registro de empleados
  4. endwhile

5. Imprime totEmpDep, totBrutoDep, totImpuestoDep, totNetoDep
  6. Incrementa totEmp con totEmpDep  
totBruto con totBrutoDep  
totImpuesto con totImpuestoDep  
totNeto con totNetoDep
- i. endwhile
  - j. Imprime totEmp, totBruto, totImpuesto, totNeto
  - k. Cerrar archivo Empleados
  - l. Fin Función



En la Web del libro encontrará un simulador que muestra la ejecución de un programa que lleva a cabo operaciones con archivos de acceso secuencial.

## 8.5 Proceso de un archivo directo

Cuando se va a utilizar un archivo con organización directa, es aconsejable crear los registros del archivo con ceros y nulos, al final de éste, se coloca una marca de fin de archivo. Después de esto se pueden hacer altas, bajas y cambios de forma directa. En caso de ser necesario, el archivo puede agrandarse, agregándole más registros en ceros y nulos, recorriendo la marca de fin del archivo.

### Creación

Esta operación, como su nombre lo indica, nos permite crear el archivo. A continuación se graban los registros; los datos numéricos en ceros y los datos cadena de caracteres en nulos. Al final del archivo, al cerrarse se coloca una marca de fin del archivo.

### Altas

Las altas al archivo se harán de la siguiente forma: El empleado número 1 se almacenará a partir del byte 0 (cero) del archivo, el empleado número 2 a partir del siguiente byte después del último usado por el anterior empleado, y así sucesivamente.

### Bajas

Para dar de baja a un empleado, se teclea su número, se lee el registro, se despliegan (imprimen) los datos y se pregunta si está seguro de darlos de baja, si es así, se graba en su lugar un registro con ceros y nulos.

### Cambios

Para realizar cambios en algunos de los datos de cada registro, es necesario indicar el número del empleado. Se lee el registro correspondiente; se despliegan los datos, se realizan los cambios y se vuelve a grabar corregido.

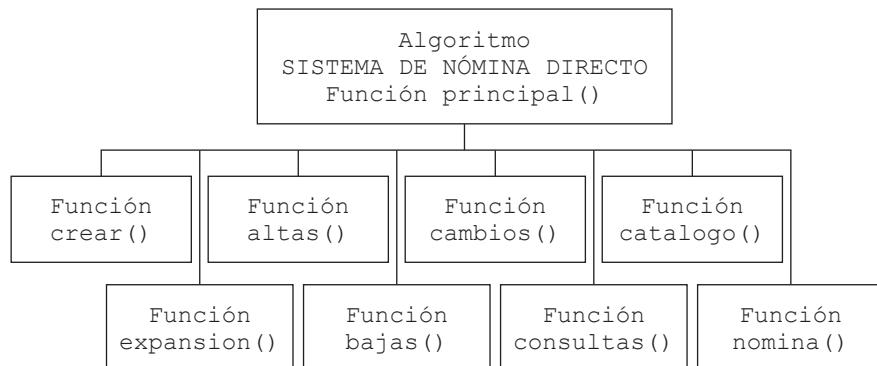
### Consultas

Para consultar los datos de un empleado se solicita y se lee el número, luego se busca el registro en el archivo, se lee del archivo y se imprimen (despliegan) los datos en la pantalla.

#### Ejemplo:

A continuación se elabora un algoritmo similar al del punto anterior, sólo que ahora permite procesar un archivo con organización directa, para aplicar los conceptos antes decritos. En la función principal se ofrece un menú con las opciones: Crear el archivo, hacerle altas, bajas, cambios, consultas al archivo, emitir el reporte catálogo de empleados con cortes de control y emitir el reporte nómina quincenal con cortes de control. A continuación se tiene el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*



```

Algoritmo SISTEMA DE NÓMINA DIRECTO
1. Declarar
  Tipos
    persona: Registro
      numero: Entero
      nombre: Cadena[30]
      depto: Entero
      puesto: Entero
      sueldo: Real
    FinRegistro
  Variables
    regEmpleado: persona
    empleados: Archivo de persona
    i, num, numRegs, opcion, opcion2: Entero
    desea, seguro: Carácter
  
```

2. Función principal()

a. do

1. Imprimir MENÚ

| SISTEMA DE NÓMINA     |
|-----------------------|
| 1. CREAR ARCHIVO      |
| 2. EXPANSIÓN          |
| 3. ALTAS              |
| 4. BAJAS              |
| 5. CAMBIOS            |
| 6. CONSULTAS          |
| 7. CATÁLOGO EMPLEADOS |
| 8. NÓMINA QUINCENAL   |
| 9. FIN                |
| ESCOGER OPCIÓN:       |

2. Leer opcion

3. switch opcion

- 1: crear()
- 2: expansion()
- 3: altas()
- 4: bajas()
- 5: cambios()
- 6: consultas()
- 7: catalogo()
- 8: nomina()

4. endswitch

b. while opcion != 9

c. Fin Función principal

3. Función crear()

a. Crear (empleados, "C:\AREMDI.DAT", directo, w)

b. Solicitar Cantidad de registros que tendrá  
el archivo

c. Leer numRegs

d. regEmpleado.numero = 0, regEmpleado.nombre = ""  
regEmpleado.depto = 0, regEmpleado.puesto = 0  
regEmpleado.sueldo = 0

e. for i=1; i<=numRegs; i++  
1. Imprimir (empleados, regEmpleado)

f. endfor

g. Cerrar (empleados)

h. Fin Función crear

4. Función expansion()
  - a. Abrir (empleados, "C:\AREMDI.DAT", directo, rw)
  - b. Encontrar(empleados, TamañoAr(empleados))
  - c. Solicitar Cantidad de registros que se agregarán al archivo
  - d. Leer numRegs
  - e. regEmpleado.numero = 0, regEmpleado.nombre = ""  
regEmpleado.depto = 0, regEmpleado.puesto = 0  
regEmpleado.sueldo = 0
  - f. for i=1; i<=numRegs; i++
    1. Imprimir (empleados, regEmpleado)
  - g. endfor
  - h. Cerrar (empleados)
  - i. Fin Función expansion
5. Función altas()
  - a. Abrir (empleados, "C:\AREMDI.DAT", directo, rw)
  - b. do
    1. Imprimir PANTALLA de captura

|                                                           |  |
|-----------------------------------------------------------|--|
| SISTEMA DE NÓMINA<br>ALTAS ARCHIVO DE EMPLEADOS           |  |
| NÚMERO:<br>NOMBRE:<br>DEPARTAMENTO:<br>PUESTO:<br>SUELDO: |  |
| ¿OTRA ALTA (S/N) ?:                                       |  |

2. Leer regEmpleado.numero, regEmpleado.nombre,  
regEmpleado.depto, regEmpleado.puesto,  
regEmpleado.sueldo
3. Encontrar (empleados, (regEmpleado.numero-1)\*40)
4. Imprimir (empleados, regEmpleado)
5. Leer desea
  - c. while desea == 'S'
  - d. Cerrar (empleados)
  - e. Fin Función altas

6. Función bajas()
- Abrir (empleados, "C:\AREMDI.DAT", directo, rw)
  - do
    - Imprimir PANTALLA de captura
- |                                                 |
|-------------------------------------------------|
| SISTEMA DE NÓMINA<br>BAJAS ARCHIVO DE EMPLEADOS |
| NÚMERO:                                         |
| NOMBRE:                                         |
| DEPARTAMENTO:                                   |
| PUESTO:                                         |
| SUELDO:                                         |
| SON LOS DATOS; ¿SEGURO(S/N)?:                   |
| ¿OTRA BAJA (S/N)?:                              |
- Leer num
  - Encontrar (empleados, (num-1)\*40)
  - Leer (empleados, regEmpleado)
  - Imprimir regEmpleado.nombre, regEmpleado.depto, regEmpleado.puesto, regEmpleado.sueldo
  - Preguntar si está seguro
  - Leer seguro
  - if seguro == 'S' then
    - regEmpleado.numero = 0, regEmpleado.nombre = ""  
regEmpleado.depto = 0, regEmpleado.puesto = 0  
regEmpleado.sueldo = 0
    - Encontrar (empleados, (num-1)\*40)
    - Imprimir (empleados, regEmpleado)
  - endif
  - Leer desea
  - while desea == 'S'
  - Cerrar (empleados)
  - Fin Función bajas

7. Función cambios()
- Abrir (empleados, "C:\AREMDI.DAT", directo, rw)
  - do
    - Imprimir PANTALLA de captura

|                                                   |  |
|---------------------------------------------------|--|
| SISTEMA DE NÓMINA<br>CAMBIOS ARCHIVO DE EMPLEADOS |  |
| NÚMERO:                                           |  |
| 1. NOMBRE:                                        |  |
| 2. DEPARTAMENTO:                                  |  |
| 3. PUESTO:                                        |  |
| 4. SUELDO:                                        |  |
| DATO A CAMBIAR(1,2,3,4,0=FIN) :                   |  |
| ¿OTRO CAMBIO (S/N) ?:                             |  |

- Leer num
- Encontrar (empleados, (num-1)\*40)
- Leer (empleados, regEmpleado)
- Imprimir regEmpleado.nombre, regEmpleado.depto, regEmpleado.puesto, regEmpleado.sueldo
- Leer opcion2
- while (opcion2>0)AND(opcion2<5)
  - switch opcion2
    - Leer regEmpleado.nombre
    - Leer regEmpleado.depto
    - Leer regEmpleado.puesto
    - Leer regEmpleado.sueldo
  - endswitch
  - Leer opcion2
- endwhile
- Encontrar (empleados, (num-1)\*40)
- Imprimir (empleados, regEmpleado)
- Leer desea
- while desea == 'S'
- Cerrar (empleados)
- Fin Función cambios

8. Función consultas()
- Abrir (empleados, "C:\AREMDI.DAT", directo, rw)
  - do
    - Imprimir PANTALLA de captura
 

|                                                     |  |
|-----------------------------------------------------|--|
| SISTEMA DE NÓMINA<br>CONSULTAS ARCHIVO DE EMPLEADOS |  |
| NÚMERO:                                             |  |
| NOMBRE:                                             |  |
| DEPARTAMENTO:                                       |  |
| PUESTO:                                             |  |
| SUELDO:                                             |  |
| ¿OTRA CONSULTA (S/N) ?:                             |  |
    - Leer num
    - Encontrar (empleados, (num-1)\*40)
    - Leer (empleados, regEmpleado)
    - Imprimir regEmpleado.nombre, regEmpleado.depto, regEmpleado.puesto, regEmpleado.sueldo
    - Leer desea
      - while desea == 'S'
      - Cerrar (empleados)
      - Fin Función consultas
9. Función catalogo()
- Declarar
 

Variables

```
totEmp, totEmpDep, deptoProceso: Entero
totSueldos, totSueldosDep: Real
```
  - Abrir (empleados, "C:\AREMDI.DAT", directo, rw)
  - totEmp = 0
 

```
totSueldos = 0
```
  - Imprimir encabezado
  - Leer (empleados, regEmpleado)
  - while NOT(Eof(Empleados))
    - totEmpDep = 0 ; totSueldosDep = 0
    - deptoProceso = regEmpleado.depto
    - while (regEmpleado.depto==DeptProceso)
 AND (NOT(Eof(Empleados)))
      - Imprimir regEmpleado.numero,
 regEmpleado.nombre, regEmpleado.depto,
 regEmpleado.puesto, regEmpleado.sueldo
      - totEmpDep = totEmpDep + 1
      - totSueldosDep = totSueldosDep
 + regEmpleado.sueldo
      - Leer (empleados, regEmpleado)

```
    4. endwhile
    5. Imprimir totEmpDep, totSueldosDep
    6. totEmp = totEmp + totEmpDep
       totSueldos = totSueldos + totSueldosDep
    g. endwhile
    h. Imprimir totEmp, totSueldos
    i. Cerrar (empleados)
    j. Fin Función catalogo
10.Función nomina()
    a. Declarar
        Variables
            totEmp, totEmpDep, deptoProceso: Entero
            bruto, impuesto, neto, totBruto, totBrutoDep,
            totImpuesto, totImpuestoDep, totNeto,
            totNetoDep, salMin: Real
    b. Abrir (empleados, "C:\AREMDI.DAT", directo, rw)
    c. totEmp = 0; totBruto = 0; totImpuesto = 0;
       totNeto = 0
    d. Solicitar Salario mínimo quincenal
    e. Leer salMin
    f. Imprimir encabezado
    g. Leer (empleados, regEmpleado)
    h. while NOT(Eof(Empleados))
        1. totEmpDep = 0 ; totBrutoDep = 0
           totImpuestoDep = 0; totNetoDep = 0
        2. deptoProceso = regEmpleado.depto
        3. while (regEmpleado.depto==DeptProceso)
           AND(NOT(Eof(Empleados)))
            a. bruto = regEmpleado.sueldo / 2
            b. if bruto > salMin then
                1. impuesto = (bruto - salMin) * 0.05
            c. else
                1. impuesto = 0
            d. endif
            e. neto = bruto - impuesto
            f. Imprimir regEmpleado.numero,
               regEmpleado.nombre, regEmpleado.depto,
               bruto, impuesto, neto
            g. totEmpDep = totEmpDep + 1
               totBrutoDep = totBrutoDep + bruto
               totImpuestoDep = totImpuestoDep + impuesto
               totNetoDep = totNetoDep + neto
            h. Leer (empleados, regEmpleado)
```

```
4. endwhile  
5. Imprimir totEmpDep, totBrutoDep,  
       totImpuestoDep, totNetoDep  
6. totEmp = totEmp + totEmpDep  
   totBruto = totBruto + totBrutoDep  
   totImpuesto = totImpuesto + totImpuestoDep  
   totNeto = totNeto + totNetoDep  
i. endwhile  
j. Imprimir totEmp, totBruto, totImpuesto, totNeto  
k. Cerrar (empleados)  
l. Fin Función nomina  
Fin
```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C805.C



*Explicación:*

1. Se hacen declaraciones globales
2. Función principal()
  - a. Inicia ciclo do
    1. Imprime MENÚ y se solicita la opción
    2. Lee opcion
    3. switch opcion
      - Si opcion == 1: Llama Función crear()
      - Si opcion == 2: Llama Función expansion()
      - Si opcion == 3: Llama Función altas()
      - Si opcion == 4: Llama Función bajas()
      - Si opcion == 5: Llama Función cambios()
      - Si opcion == 6: Llama Función consultas()
      - Si opcion == 7: Llama Función catalogo()
      - Si opcion == 8: Llama Función nomina()
    4. endswitch
  - b. Fin ciclo while opcion != 9
  - c. Fin Función
3. Función crear()
  - a. Crea el archivo Empleados organización Directo
  - b. Solicita la Cantidad de registros que tendrá el archivo
  - c. Lee en numRegs

- d. Hace el regEmpleado en ceros los datos numéricos y nulo el cadena, es decir, crea un registro “limpio” sin basura
- e. Inicia ciclo for desde i = 1 hasta numRegs
  1. Imprime un registro limpio en empleados
- f. Fin for
- g. Cerrar archivo empleados
- h. Fin Función
4. Función expansion()
  - a. Abre archivo empleados como directo
  - b. Coloca el apuntador del archivo en el siguiente componente después del último
  - c. Solicita la Cantidad de registros que se añadirán al archivo
  - d. Lee en numRegs
  - e. Hace el regEmpleado en ceros los datos numéricos y nulo el cadena, es decir, crea un registro “limpio” sin basura
  - f. Inicia ciclo for desde i = 1 hasta numRegs
    1. Imprime un registro limpio en empleados
  - g. Fin for
  - h. Cerrar archivo empleados
  - i. Fin Función
5. Función altas()
  - a. Abre archivo empleados como directo
  - b. Inicia ciclo do
    1. Imprimir PANTALLA de captura donde solicita los datos
    2. Se leen los datos
    3. Se coloca el apuntador del archivo en el byte a partir del cual le corresponde guardarse, éste se calcula así: A regEmpleado. numero se le resta 1 y luego se multiplica por 40, que es la cantidad de bytes que ocupa el registro de cada empleado
    4. Imprime el registro en empleados
    5. Lee desea
  - c. Fin ciclo while desea == ‘S’
  - d. Cerrar archivo empleados
  - e. Fin Función

6. Función bajas()

- a. Abre archivo empleados como directo
- b. Inicia ciclo do
  1. Imprime PANTALLA de captura donde solicita los datos
  2. Lee en num el número de empleado a dar de baja
  3. Coloca el apuntador en el byte a partir del cual están guardados los datos del empleado número num, éste se calcula así: A num se le resta 1 y luego se multiplica por 40, que es la cantidad de bytes que ocupa el registro de cada empleado
  4. Lee el registro del archivo empleados
  5. Imprime los datos del empleado
  6. Preguntar si está seguro de querer darlos de baja
  7. Lee en seguro la respuesta
  8. Si seguro == 'S' Entonces
    - a. Hace el regEmpleado en ceros y nulos, es decir, crea un registro "limpio"
    - b. Coloca el apuntador en el byte a partir del cual están guardados los datos del empleado número num
    - c. Imprime el registro en empleados, lo que significa "borrar" los datos que estaban en el registro
  9. Fin if
10. Lee en desea la respuesta

- c. Fin ciclo while desea == 'S'
- d. Cerrar archivo empleados
- e. Fin Función

7. Función cambios()

- a. Abre archivo empleados como directo
- b. Inicia ciclo do
  1. Imprime PANTALLA de captura donde solicita los datos
  2. Lee num el número de empleado a hacerle cambio
  3. Coloca el apuntador en el byte a partir del cual están guardados los datos del empleado número num, éste se calcula así: A num se le resta 1 y luego se multiplica por 40, que es la cantidad de bytes que ocupa el registro de cada empleado
  4. Lee el registro del archivo empleados
  5. Imprime los datos del empleado
  6. Lee en opcion2 el número de dato a cambiar
  7. Inicia ciclo while, si está entre 1 y 4
    - a. switch opcion2

- Si opcion2 == 1: Lee regEmpleado.nombre  
Si opcion2 == 2: Lee regEmpleado.depto  
Si opcion2 == 3: Lee regEmpleado.puesto  
Si opcion2 == 4: Lee regEmpleado.sueldo
- b. endswitch
  - c. Lee en opcion2 el número de dato a cambiar
8. Fin while
  9. Coloca el apuntador en el byte a partir del cual están guardados los datos del empleado número num
  10. Imprime el registro regEmpleado actualizado en Empleados
  11. Lee en desea si desea continuar
- c. Fin ciclo while desea == 'S'
  - d. Cerrar archivo empleados
  - e. Fin Función
8. Función consultas()
- a. Abre archivo empleados como directo
  - b. Inicia ciclo do
    1. Imprimir PANTALLA de captura donde solicita los datos
    2. Lee num el número de empleado a hacerle cambio
    3. Coloca el apuntador en el byte a partir del cual están guardados los datos del empleado número num, éste se calcula así: A num se le resta 1 y luego se multiplica por 40, que es la cantidad de bytes que ocupa el registro de cada empleado
    4. Lee el registro del archivo empleados
    5. Imprime los datos del empleado
    6. Lee en desea si desea continuar
  - c. Fin ciclo do...while desea == 'S'
  - d. Cerrar archivo empleados
  - e. Fin Función
9. Función catalogo()  
Ya se explicó en el punto de emisión de reportes con cortes de control
  10. Función nomina()  
Ya se explicó en el punto de emisión de reportes con cortes de control

Observaciones:

Aquí tenemos un algoritmo completo que permite manejar un pequeño sistema de nómina, aplicando todas las operaciones sobre registros y archivos, sin embargo, el algoritmo no contempla validaciones, por ejemplo:

- Al crear el archivo no se da la oportunidad de retractarse, y se puede crear un archivo ya existente erróneamente, perdiéndose los datos que ya tenía.
- Al hacer un alta no se checa si el empleado ya está dado de alta, se puede dar de alta de nuevo. Tampoco se checa si el número de empleado cabe en el archivo de acuerdo con el número de registros con que fue creado.
- Al hacer la captura de datos no se detecta la ocurrencia de errores, y mucho menos recuperarse de los mismos, por ejemplo, si se lee un dato de tipo entero, no se debe permitir que se tecleen letras o símbolos especiales, etc. Al leer el nombre de una persona no deben permitirse números o símbolos especiales.
- Al hacer una consulta, baja o cambio, dar un mensaje de error si se teclea un número de empleado que no existe.



En la Web del libro encontrará un simulador que muestra la ejecución de un programa que lleva a cabo operaciones con archivos de acceso directo.

A manera de ejercicio se recomienda que elabore el mismo sistema de nómina, pero ahora contemplando algunas o todas las validaciones antes mencionadas.

## 8.6 Ejercicios resueltos

La tabla 8.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 8 de la Web del libro.

**Tabla 8.1**

| Ejercicio       | Descripción                      |
|-----------------|----------------------------------|
| Ejercicio 8.6.1 | Procesa un archivo de obreros    |
| Ejercicio 8.6.2 | Procesa un archivo de vendedores |



En la Web del libro encontrará un simulador que muestra la ejecución de un programa que lleva a cabo operaciones con registros.

## 8.7 Ejercicios propuestos

1. Elaborar un algoritmo que maneje un arreglo de 30 elementos, donde cada elemento es un registro de un alumno que contiene los datos NOMBRE, CALIFICACIÓN 1, CALIFICACIÓN 2 y CALIFICACIÓN 3; que permita leer los datos de los 30 alumnos e imprimir el reporte:

| NOMBRE                    | REPORTE DE CALIFICACIONES |          |          |              | OBSERVACIÓN |
|---------------------------|---------------------------|----------|----------|--------------|-------------|
|                           | CALIF.1                   | CALIF. 2 | CALIF. 3 | CALIF. FINAL |             |
| X-----X                   | - - -                     | - - -    | - - -    | - - -        | APROBADO    |
| X-----X                   | - - -                     | - - -    | - - -    | - - -        | REPROBADO   |
| - - -                     |                           |          |          |              |             |
| - - -                     |                           |          |          |              |             |
| X-----X                   | - - -                     | - - -    | - - -    | - - -        | APROBADO    |
| PROMEDIOS                 | - - -                     | - - -    | - - -    | - - -        |             |
| TOTAL ALUMNOS APROBADOS:  | - - -                     |          |          |              |             |
| TOTAL ALUMNOS REPROBADOS: | - - -                     |          |          |              |             |

2. Elaborar un algoritmo que maneje un arreglo de 25 registros, cada registro contiene los datos de una persona que será invitada a la reunión semanal. El programa debe permitir crear en nulos los 25 registros y luego dar de alta las personas que se invitarán, también debe permitir hacer bajas, cambios y consultas. Debe controlar el estado de la invitación: cuando se da de alta a la persona se le coloca en el estado 'N', es decir, que no se ha invitado; pero una vez invitado, se le debe colocar 'S'. Los datos que contiene cada registro son: NOMBRE cadena de 30, TELÉFONO cadena de 9 y ESTADO cadena de 1. Debe permitir la emisión de un reporte que muestre los datos de todas las personas y un listado donde aparezcan sólo las personas que no han sido invitadas todavía.
3. Hacer un algoritmo que permita crear un archivo de artículos, realizar altas, bajas, cambios y consultas. Los datos que se tienen de cada artículo son los siguientes:
  - Número
  - Descripción
  - Precio anterior
  - Precio actual

Asimismo, emitir el siguiente listado de inflación de artículos:

| ARTÍCULO     | ANÁLISIS DE INFLACIÓN |               |                 |
|--------------|-----------------------|---------------|-----------------|
|              | PRECIO ANTERIOR       | PRECIO ACTUAL | PTJE. INFLACIÓN |
| XXXXXXXXXXXX | 99,999.99             | 99,999.99     | 99.99           |
| XXXXXXXXXXXX | 99,999.99             | 99,999.99     | 99.99           |
| ---          | ---                   | ---           | ---             |
| ---          | ---                   | ---           | ---             |
| XXXXXXXXXXXX | 99,999.99             | 99,999.99     | 99.99           |

Promedio de inflación: 99.99

Artículo con mayor inflación: XXXXXXXXXXXXXXXXXXXXXXXXX

Porcentaje mayor de inflación: 99.99

Forma de calcular el porcentaje de inflación:

$$\text{PTJE. INFLACIÓN} = \frac{\text{Precio actual} - \text{Precio anterior}}{\text{Precio anterior}} \times 100$$

4. Elaborar un algoritmo que permita crear un archivo de alumnos, realizar altas, bajas, cambios y consultas. Los datos que se tienen de cada alumno son los siguientes:
  - Número
  - Nombre
  - Calificación 1
  - Calificación 2
  - Calificación 3
  - Calificación 4

Emitir el siguiente reporte:

| NOMBRE               | ANÁLISIS DE CALIFICACIONES |       |       |       | PROMEDIO |
|----------------------|----------------------------|-------|-------|-------|----------|
|                      | CAL.1                      | CAL.2 | CAL.3 | CAL.4 |          |
| XXXXXXXXXXXXXXXXXXXX | 99.99                      | 99.99 | 99.99 | 99.99 | 99.99    |
| XXXXXXXXXXXXXXXXXXXX | 99.99                      | 99.99 | 99.99 | 99.99 | 99.99    |
| - - -                | - - -                      | - - - | - - - | - - - | - - -    |
| - - -                | - - -                      | - - - | - - - | - - - | - - -    |
| XXXXXXXXXXXXXXXXXXXX | 99.99                      | 99.99 | 99.99 | 99.99 | 99.99    |
| PROMEDIOS GENERALES  | 99.99                      | 99.99 | 99.99 | 99.99 | 99.99    |

El promedio general de cada calificación se calcula sumando las calificaciones de todos los alumnos y dividiendo el resultado entre el número de alumnos.

5. Hacer un algoritmo igual al del ejercicio de empleados del capítulo 8, pero se añadirá un archivo que contenga la siguiente tabla para el cálculo del impuesto:

|          | Lim.Inferior | Lim.Superior | Cuota Fija | Ptje.Impuesto |
|----------|--------------|--------------|------------|---------------|
| nivel 1  |              |              |            |               |
| nivel 2  |              |              |            |               |
| nivel 3  |              |              |            |               |
| - - -    |              |              |            |               |
| nivel 15 |              |              |            |               |

Proceso para calcular el impuesto:

Tomando como base el excedente del sueldo bruto sobre el salario mínimo:

- Se busca el nivel en el que se encuentra dicho excedente.
- Se obtiene la diferencia entre el excedente y el límite inferior del nivel; a ésta se le aplica el porcentaje de impuesto y se le suma la cuota fija.

6. Hacer un algoritmo que permita crear un archivo de pedidos, realizar altas, bajas, cambios y consultas. Los datos de cada pedido son los siguientes:
- Nombre del cliente
  - Dirección
  - Artículo
  - Cantidad
  - Precio unitario

Además, deberá emitir el siguiente listado de pedidos:

| LISTADO DE PEDIDOS   |            |          |              |             |
|----------------------|------------|----------|--------------|-------------|
| NOMBRE DEL CLIENTE   | ARTÍCULO   | CANTIDAD | PRECIO.UNIT. | TOTAL       |
| XXXXXXXXXXXXXXXXXXXX | XXXXXXXXXX | 999      | 99,999.99    | 999,999.99  |
| XXXXXXXXXXXXXXXXXXXX | XXXXXXXXXX | 999      | 99,999.99    | 999,999.99  |
| XXXXXXXXXXXXXXXXXXXX | XXXXXXXXXX | 999      | 99,999.99    | 999,999.99  |
| - - -                | - - -      | - - -    | - - -        | - - -       |
| XXXXXXXXXXXXXXXXXXXX | XXXXXXXXXX | 999      | 99,999.99    | 999,999.99  |
| TOTAL 999            |            |          | 999,999.99   | 9999,999.99 |

7. Hacer un algoritmo que permita crear un archivo de obreros, realizar altas, bajas, cambios y consultas. Los datos de cada obrero son los siguientes:
- Nombre del obrero
  - Producción de los 12 meses en un arreglo de 12 elementos
- Asimismo, que emita un premio anual al obrero más productivo. El premio se calcula de la siguiente manera:
- Cantidad más alta producida menos cantidad más baja multiplicado por 10.
- Imprimir:
- Nombre del obrero : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
- Valor del premio: 99,999,999.99

8. Hacer un algoritmo que permita crear un archivo de cuentas de cheques, realizar altas, bajas, cambios y consultas. Los datos de cada cuenta son:
- Número de cuenta
  - Nombre del cliente
  - Saldo

Además debe permitir que se hagan depósitos y retiros. Se requiere que emita el siguiente reporte de estados de cuenta:

| NÚM. CUENTA        | ESTADOS DE CUENTA    |              |
|--------------------|----------------------|--------------|
|                    | NOMBRE DEL CLIENTE   | SALDO        |
| 99999999-9         | XXXXXXXXXXXXXXXXXXXX | 999,999.99   |
| TOTAL 999 CLIENTES |                      | 9,999,999.99 |

9. Elaborar un algoritmo que maneje el sistema de nómina que se elaboró anteriormente, sólo que, en lugar de imprimir el departamento y puesto como un número, que lo haga en forma de nombre del departamento y nombre de puesto. Debe manejar un archivo de departamentos y otro de puestos, que se les haga altas, bajas y cambios; de ahí tomar los nombres en forma de nombres. Los datos de estos nuevos archivos son:

|               |              |
|---------------|--------------|
| departamentos | puestos      |
| depto.        | puesto       |
| nombreDept    | nombrePuesto |

10. Elaborar un algoritmo que maneje un sistema de control de usuarios del agua potable de una localidad, que permita crear, hacer altas, bajas y cambios a un archivo de usuarios y otro de colonias. El archivo de usuarios contiene los datos de todos los usuarios de la localidad, la cual se divide en varias colonias, es por ello que cada usuario tiene un dato que es la clave de la colonia a la que pertenece. El archivo de colonias contiene los nombres de todas las colonias de la localidad. También se manejará un archivo de tarifas, que contendrá un arreglo de 8 elementos, en el elemento 1 tendrá la tarifa que le corresponde al usuario tipo 1, en el elemento 2 la tarifa para el usuario tipo 2, y así sucesivamente. Cada vez que se desee cambiar las tarifas, simplemente se crea el archivo con las tarifas correspondientes.

| usuarios     | colonias     | tarifas   |
|--------------|--------------|-----------|
| numero       | claveColonia | tarifa[8] |
| tipoUsuario  | colonia      |           |
| nombre       |              |           |
| claveColonia |              |           |
| direccion    |              |           |
| consumo      |              |           |
| pago         |              |           |

*En donde:*

|              |                                                                                                                                         |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| numero       | Es el número de identificación del usuario.                                                                                             |
| tipoUsuario  | Es el tipo de usuario (1,2,3,4,5,6,7,8).                                                                                                |
| nombre       | Es el nombre del usuario.                                                                                                               |
| claveColonia | Es la clave de identificación de la colonia.                                                                                            |
| direccion    | Es la dirección del domicilio del usuario.                                                                                              |
| consumo      | Es el consumo en metros cúbicos de agua que consumió el usuario en el periodo correspondiente.                                          |
| pago         | Es el pago realizado por el usuario por el periodo correspondiente.<br>Si está en cero, es que no ha pagado.                            |
| colonia      | Es el nombre de la colonia en la que vive el usuario.                                                                                   |
| tarifa[8]    | Es un arreglo que tiene 8 elementos, en el 1 tiene la tarifa que le corresponde al usuario tipo 1, en el 2 al usuario tipo 2, etcétera. |



Para efectos de este reporte, el archivo debe estar ordenado por colonia, es decir, todos los usuarios de la colonia 1 están juntos al principio del archivo, luego siguen los de la colonia 2, y así sucesivamente. Se pide el total de pagos no hechos, es decir, lo que debe cada usuario, totalizado por cada colonia y en general.

Para la captura del consumo se debe hacer un proceso especial, al igual que para la captura de los pagos. El programa también debe proporcionar la posibilidad de hacer consultas de usuarios y de colonias con dar la clave correspondiente; además debe emitir los siguientes reportes:

| REPORTE DE PAGOS         |                |         |      |
|--------------------------|----------------|---------|------|
| NÚMERO                   | NOMBRE USUARIO | COLONIA | PAGO |
| ---                      | X-----X        | X-----X | ---  |
| ---                      |                |         |      |
| ---                      |                |         |      |
| ---                      | X-----X        | X-----X | ---  |
| TOTAL COLONIA - USUARIOS |                |         | ---  |
| ---                      |                |         |      |
| ---                      |                |         |      |
| TOTAL COLONIA - USUARIOS |                |         | ---  |
| TOTAL GENERAL - USUARIOS |                |         | ---  |

Se pide el total de pagos hechos por cada colonia y el total general.

| REPORTE DE PAGOS NO REALIZADOS |                |         |      |
|--------------------------------|----------------|---------|------|
| NÚMERO                         | NOMBRE USUARIO | COLONIA | DEBE |
| ---                            | X-----X        | X-----X | ---  |
| ---                            |                |         |      |
| ---                            | X-----X        | X-----X | ---  |
| TOTAL COLONIA - USUARIOS       |                |         | ---  |
| TOTAL COLONIA - USUARIOS       |                |         | ---  |
| TOTAL GENERAL - USUARIOS       |                |         | ---  |

| REPORTE DE FACTURACIÓN   |         |         |         |         |              |       |
|--------------------------|---------|---------|---------|---------|--------------|-------|
| NÚM.                     | NOMBRE  | COLONIA | CONSUMO | IMPORTE | SOBRECONSUMO | TOTAL |
| ---                      | X-----X | X-----X | ---     | ---     | ---          | ---   |
| ---                      |         |         |         |         |              |       |
| ---                      | X-----X | X-----X | ---     | ---     | ---          | ---   |
| TOTAL COLONIA - USUARIOS |         |         | ---     | ---     | ---          | ---   |
| ---                      |         |         |         |         |              |       |
| ---                      |         |         |         |         |              |       |
| TOTAL COLONIA - USUARIOS |         |         | ---     | ---     | ---          | ---   |
| TOTAL GENERAL - USUARIOS |         |         | ---     | ---     | ---          | ---   |

**Nota:** Para efectos de este reporte, el archivo debe estar ordenado por colonia, es decir, todos los usuarios de la colonia 1 están juntos al principio del archivo, luego siguen los de la colonia 2, y así sucesivamente.

*En donde:*

CONSUMO

Se lee junto con los datos del registro del usuario y es la cantidad de metros cúbicos de agua que consumió el usuario.

|              |                                                                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IMPORTE      | Se calcula multiplicando el consumo por la tarifa correspondiente, misma que debe obtenerse del arreglo de tarifas de acuerdo con el tipo de usuario.      |
| SOBRECONSUMO | Si consumió arriba de 50 metros cúbicos, se le cobra un 10% de sobreconsumo, si pasó de 100 un 20%, si pasó de 200 un 30% más y si pasó de 300 un 50% más. |
| TOTAL        | Es el total a pagar; se suman el IMPORTE y SOBRECONSUMO.                                                                                                   |

Se pide el total de consumo, importe, sobreconsumo y total por cada colonia y en general.

| RECIBO DEL AGUA |                  |
|-----------------|------------------|
| NÚMERO:         | Recibo Número:   |
| USUARIO:        |                  |
| DIRECCIÓN:      |                  |
| COLONIA:        | Páguese antes de |
| C.P.:           |                  |
| TIPO USUARIO:   |                  |
| CONSUMO:        |                  |
| TARIFA:         |                  |
|                 | IMPORTE:         |
|                 | SOBRECONSUMO:    |
|                 | TOTAL A PAGAR:   |

**Nota:** Se debe listar un recibo por cada usuario.

11. Elaborar un algoritmo que maneje un sistema de control de ventas de libros de una editorial, que permita crear, hacer altas, bajas y cambios a los archivos de libros, autores, ventas, clientes, ciudades, países y estados. A continuación se describen los datos que contiene cada uno de los archivos:

| libros       | clientes     | autores        |
|--------------|--------------|----------------|
| claveLibro   | claveCliente | claveAutor     |
| título       | razonSocial  | nombre         |
| claveAutor   | direccionCte | direccionAutor |
| edicion      | claveCiudad  | claveCiudad    |
| fechaEdicion | cpCliente    | cpAutor        |
| existencia   | claveEstado  | claveEstado    |
| stockMinimo  | clavePais    | clavePais      |
| stockMaximo  | telCliente   | telAutor       |
| nivelReorden | faxCliente   | faxAutor       |
| precioVenta  |              |                |

| <u>ventas</u> | <u>ciudades</u> | <u>estados</u> |
|---------------|-----------------|----------------|
| claveLibro    | claveCiudad     | claveEstado    |
| claveCliente  | ciudad          | estado         |
| cantidad      |                 |                |
| precio        |                 |                |
| fecha         |                 |                |

| <u>paises</u> |
|---------------|
| clavePais     |
| pais          |

*En donde:*

|                |                                                                                                    |
|----------------|----------------------------------------------------------------------------------------------------|
| claveLibro     | Es la clave de identificación del libro.                                                           |
| titulo         | Es el título del libro.                                                                            |
| claveAutor     | Clave que identifica al autor.                                                                     |
| edicion        | Es el número de edición del libro.                                                                 |
| fechaEdicion   | Es la fecha de edición.                                                                            |
| existencia     | Es la cantidad de unidades que hay en existencia.                                                  |
| stockMinimo    | Es la cantidad mínima que se debe tener en existencia.                                             |
| stockMaximo    | Es la cantidad máxima que se debe tener en existencia.                                             |
| nivelReorden   | Es la cantidad de unidades en existencia que indica que se debe ordenar más unidades (reimprimir). |
| precioVenta    | Es el precio unitario de venta del libro.                                                          |
| claveCliente   | Es la clave de identificación del cliente.                                                         |
| razonSocial    | Es la razón social del cliente.                                                                    |
| direccionCte   | Es la dirección del cliente.                                                                       |
| claveCiudad    | Es la clave de la ciudad donde radica el cliente.                                                  |
| cpCliente      | Es el código postal del domicilio del cliente.                                                     |
| claveEstado    | Es la clave del estado o entidad federativa donde radica el cliente.                               |
| clavePais      | Es la clave del país donde radica el cliente.                                                      |
| telCliente     | Es el número telefónico del cliente.                                                               |
| faxCliente     | Es el número de Fax del cliente.                                                                   |
| nombre         | El nombre del autor.                                                                               |
| direccionAutor | Es la dirección del autor.                                                                         |
| claveCiudad    | Es la clave de la ciudad donde radica el autor.                                                    |
| cpAutor        | Es el código postal del domicilio del autor.                                                       |
| claveEstado    | Es la clave del estado o entidad federativa donde radica el autor.                                 |

|           |                                               |
|-----------|-----------------------------------------------|
| clavePais | Es la clave del país donde radica el autor.   |
| telAutor  | Es el número telefónico del autor.            |
| faxAutor  | Es el número de fax del autor.                |
| cantidad  | Es la cantidad de libros comprados.           |
| precio    | Es el precio unitario por libro comprado.     |
| fecha     | Es la fecha de la compra.                     |
| ciudad    | Es el nombre de la ciudad.                    |
| estado    | Es el nombre del estado o entidad federativa. |
| pais      | Es el nombre del país.                        |

El algoritmo también debe proporcionar la posibilidad de hacer consultas a los datos de cada uno de los archivos con dar la clave correspondiente; además debe emitir los siguientes reportes:

| REPORTE DE LIBROS |         |         |       |            |       |            |        |
|-------------------|---------|---------|-------|------------|-------|------------|--------|
| CLAVE             | LIBRO   | TÍTULO  | AUTOR | PRECIO     | VALOR | STOCK      | STOCK  |
|                   |         |         |       | EXISTENCIA | VENTA | EXISTENCIA | MÍNIMO |
| - - -             | X-----X | X-----X | - - - | - - -      | - - - | - - -      | - - -  |
| - - -             |         |         |       |            |       |            |        |
| - - -             | X-----X | X-----X | - - - | - - -      | - - - | - - -      | - - -  |
| TOTAL - LIBROS    |         |         | - - - | - - -      | - - - | - - -      | - - -  |

| REPORTE DE VENTAS POR AUTOR |         |         |        |       |            |          |
|-----------------------------|---------|---------|--------|-------|------------|----------|
| CLAVE                       | AUTOR   | AUTOR   | TÍTULO | LIBRO | TOT. UNID. | PRECIO   |
|                             |         |         |        |       | VENDIDAS   | UNITARIO |
| - - -                       | X-----X | X-----X | - - -  | - - - | - - -      | - - -    |
| - - -                       |         |         |        |       |            |          |
| - - -                       | X-----X | X-----X | - - -  | - - - | - - -      | - - -    |
| TOTAL AUTOR - LIBROS        |         |         | - - -  | - - - | - - -      | - - -    |
| - - -                       |         |         |        |       |            |          |
| TOTAL AUTOR - LIBROS        |         |         | - - -  | - - - | - - -      | - - -    |
| TOTAL GENERAL - LIBROS      |         |         | - - -  | - - - | - - -      | - - -    |

| REPORTE DE VENTAS POR PAÍSES |         |         |       |       |
|------------------------------|---------|---------|-------|-------|
| CLAVE                        | LIBRO   | TÍTULO  | LIBRO | AUTOR |
|                              |         |         |       | PAÍS  |
| - - -                        | X-----X | X-----X | - - - | - - - |
| - - -                        |         |         |       |       |
| - - -                        | X-----X | X-----X | - - - | - - - |
| TOTAL LIBRO - PAÍSES         |         |         | - - - | - - - |
| - - -                        |         |         |       |       |
| - - -                        |         |         |       |       |
| TOTAL LIBRO - PAÍSES         |         |         | - - - | - - - |
| TOTAL GENERAL - LIBROS       |         |         | - - - | - - - |

| REPORTE DE VENTAS DETALLADO |                        |         |         |         |         |         | TOT   | UNID. | PRECIO |
|-----------------------------|------------------------|---------|---------|---------|---------|---------|-------|-------|--------|
| CLAVE LIBRO                 | TÍTULO LIBRO           | AUTOR   | CLIENTE | CIUDAD  | ESTADO  | PAÍS    | VEND. | UNIT. | TOTAL  |
| - - -                       | X-----X                | X-----X | X-----X | X-----X | X-----X | X-----X | - - - | - - - | - - -  |
|                             |                        |         | X-----X | X-----X | X-----X | X-----X | - - - | - - - | - - -  |
|                             |                        |         |         | - - -   |         |         |       |       |        |
|                             |                        |         | X-----X | X-----X | X-----X | X-----X | - - - | - - - | - - -  |
|                             | TOTAL LIBRO - VENTAS   |         |         |         |         |         | - - - | - - - | - - -  |
|                             | TOTAL LIBRO - VENTAS   |         |         |         |         |         | - - - | - - - | - - -  |
|                             | TOTAL GENERAL - LIBROS |         |         |         |         |         | - - - | - - - | - - -  |

12. Desarrollar un algoritmo para el manejo de un sistema de control de inventarios, que permita crear, realizar altas, bajas y cambios a un archivo de artículos y uno de movimientos que contengan los siguientes datos:

| ARCHIVO DE Artículos | ARCHIVO DE Movimientos |
|----------------------|------------------------|
| numero               | numero                 |
| linea                | tipoMovimiento         |
| descripcion          | fechaMovimiento        |
| localizacion         | cantidad               |
| unidadDeMedida       | precioUnitario         |
| existencia           |                        |
| stockMinimo          |                        |
| stockMaximo          |                        |
| precioDeCompra       |                        |
| precioDeVenta        |                        |
| fechaUltCompra       |                        |
| fechaUltVenta        |                        |
| proveedor1           |                        |
| proveedor2           |                        |

Además, dicho algoritmo deberá permitirnos emitir los siguientes reportes:

| CATÁLOGO DE ARTÍCULOS |                           |                         |                       |                           |                           |             |  |
|-----------------------|---------------------------|-------------------------|-----------------------|---------------------------|---------------------------|-------------|--|
| NÚMERO LÍNEA          | UNIDAD MEDIDA DESCRIPCIÓN | LOCALIZACIÓN EXISTENCIA | STOCK MÍN. STOCK MÁX. | PCIO.ULT.CPA PCIO.ULT.VTA | FCHA.ULT.ENT FCHA.ULT.SAL | PROV1 PROV2 |  |
| 99999                 | XXXXXXXXXXXX              | XXXXXXXX                | 9999                  | 999,999.99                | 99/99/99                  | XXXXXX      |  |
| 99                    | XXXXXXXXXXXX              | 9999                    | 9999                  | 999,999.99                | 99/99/99                  | XXXXXX      |  |
| - - -                 |                           |                         |                       |                           |                           |             |  |
| - - -                 | TOTAL LÍNEA 99            |                         | 999 ARTÍCULOS         |                           |                           |             |  |
| - - -                 |                           |                         |                       |                           |                           |             |  |
| - - -                 | TOTAL LÍNEA 99            |                         | 999 ARTÍCULOS         |                           |                           |             |  |
| - - -                 |                           |                         |                       |                           |                           |             |  |
| - - -                 | TOTAL GENERAL             |                         | 9999 ARTÍCULOS        |                           |                           |             |  |

## EXISTENCIA VALORIZADA

| LÍNEA          | NÚMERO         | DESCRIPCIÓN | EXISTENCIA | PCIO.CPA | PCIO.VTA | EXISTEN.<br>PCIO.CPA | VALORIZADA<br>PCIO.VTA |
|----------------|----------------|-------------|------------|----------|----------|----------------------|------------------------|
| 99             | 99999          | XXXXXXXXXX  | 9999       | 99999.99 | 99999.99 | 99999.99             | 99999.99               |
| - - -          |                |             |            |          |          |                      |                        |
| - - -          |                |             |            |          |          |                      |                        |
| - - -          |                |             |            |          |          |                      |                        |
| TOTAL LÍNEA 99 | 999 ARTÍCULOS  |             |            |          |          | 999999.99            | 999999.99              |
| - - -          |                |             |            |          |          |                      |                        |
| - - -          |                |             |            |          |          |                      |                        |
| - - -          |                |             |            |          |          |                      |                        |
| TOTAL LÍNEA 99 | 999 ARTÍCULOS  |             |            |          |          | 999999.99            | 999999.99              |
| - - -          |                |             |            |          |          |                      |                        |
| - - -          |                |             |            |          |          |                      |                        |
| TOTAL GENERAL  | 9999 ARTÍCULOS |             |            |          |          | 999999.99            | 999999.99              |

## ARTÍCULOS CON POCO MOVIMIENTO

| LÍNEA          | NÚMERO        | DESCRIPCIÓN    | UNIDAD<br>MEDIDA | EXIST. | FECH.ULT<br>PCIO.CPA. | FECH.ULT<br>PCIO.VTA. | COMPRA   | VENTA    |
|----------------|---------------|----------------|------------------|--------|-----------------------|-----------------------|----------|----------|
| 99             | 99999         | XXXXXXXXXXXXXX | XXXXXX           | 9999   | 99999.99              | 99999.99              | 99/99/99 | 99/99/99 |
| - - -          |               |                |                  |        |                       |                       |          |          |
| - - -          |               |                |                  |        |                       |                       |          |          |
| TOTAL LÍNEA 99 | 999 ARTÍCULOS |                |                  |        |                       |                       |          |          |
| - - -          |               |                |                  |        |                       |                       |          |          |
| - - -          |               |                |                  |        |                       |                       |          |          |
| TOTAL LÍNEA 99 | 999 ARTÍCULOS |                |                  |        |                       |                       |          |          |
| - - -          |               |                |                  |        |                       |                       |          |          |
| - - -          |               |                |                  |        |                       |                       |          |          |
| TOTAL LÍNEA 99 | 999 ARTÍCULOS |                |                  |        |                       |                       |          |          |

## LISTA DE PRECIOS

| LÍNEA          | NÚMERO          | DESCRIPCIÓN        | UNIDAD MEDIDA  | PRECIO VENTA |
|----------------|-----------------|--------------------|----------------|--------------|
| 99             | 99999           | XXXXXXXXXXXXXXXXXX | XXXXXXXXXXXXXX | 999,999.99   |
| - - -          |                 |                    |                |              |
| - - -          |                 |                    |                |              |
| TOTAL LÍNEA 99 | 999 ARTÍCULOS   |                    |                |              |
| - - -          |                 |                    |                |              |
| - - -          |                 |                    |                |              |
| TOTAL LÍNEA 99 | 999 ARTÍCULOS   |                    |                |              |
| - - -          |                 |                    |                |              |
| - - -          |                 |                    |                |              |
| TOTAL LÍNEA 99 | 99999 ARTÍCULOS |                    |                |              |

## LISTADO DE PEDIDOS

| LÍNEA         | NÚMERO        | DESCRIPCIÓN  | UNIDAD<br>MEDIDA | CANTIDAD | FECHA<br>PEDIDO | FECHA<br>RECIBIDO | PROVEEDOR  |
|---------------|---------------|--------------|------------------|----------|-----------------|-------------------|------------|
| 99            | 99999         | XXXXXXXXXXXX | XXXXX            | 9999     | 99/99/99        | 99/99/99          | XXXXXXXXXX |
| - - -         |               |              |                  |          |                 |                   |            |
| - - -         |               |              |                  |          |                 |                   |            |
| TOTAL PEDIDOS | 999 ARTÍCULOS |              |                  |          |                 |                   |            |

| <b>PRODUCTOS BAJO MÍNIMO</b> |        |                      |                |          |                |
|------------------------------|--------|----------------------|----------------|----------|----------------|
| LÍNEA                        | NÚMERO | DESCRIPCIÓN          | UNIDAD MEDIDA  | CANTIDAD | COSTO ESTIMADO |
| 99                           | 99999  | XXXXXXXXXXXXXX       | XXXXXXXXXXXXXX | 9999     | 999,999.99     |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL LÍNEA 99               |        | 999 ARTÍCULOS        |                |          | 9,999,999.99   |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL LÍNEA 99               |        | 999 ARTÍCULOS        |                |          | 9,999,999.99   |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL GENERAL                |        | 9999 ARTÍCULOS       |                |          | 99,999,999.99  |
| <b>MOVIMIENTOS DEL MES</b>   |        |                      |                |          |                |
| LÍNEA                        | NÚMERO | DESCRIPCIÓN          | FECHA MOV.     | TIPO     | CANTIDAD       |
| 99                           | 99999  | XXXXXXXXXXXXXXXXXXXX | 99/99/99       | X        | 9999           |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL DE MOVIMIENTOS 999     |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL DE MOVIMIENTOS 999     |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL LÍNEA 99               |        | 999 ARTÍCULOS        |                |          |                |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL LÍNEA 99               |        | 999 ARTÍCULOS        |                |          |                |
| ---                          |        |                      |                |          |                |
| ---                          |        |                      |                |          |                |
| TOTAL GENERAL                |        | 9999 ARTÍCULOS       |                |          |                |

El algoritmo deberá considerar las siguientes pantallas y menús:

| SISTEMA DE INVENTARIOS<br>MENÚ PRINCIPAL                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. CREAR ARCHIVOS<br>2. ACTUALIZACIÓN ARCH. ARTÍCULOS<br>3. CAPTURA DE MOVIMIENTOS<br>4. CONSULTAS DE ARTÍCULOS<br>5. EMISIÓN DE REPORTES<br>6. FIN |
| ESCOGER OPCIÓN                                                                                                                                      |

|                                                                  |
|------------------------------------------------------------------|
| SISTEMA DE INVENTARIOS<br>CREACIÓN DE ARCHIVOS                   |
| 1. CREAR ARCH. ARTÍCULOS<br>2. CREAR ARCH. MOVIMIENTOS<br>3. FIN |
| ESCOGER OPCIÓN                                                   |

|                                                                                     |
|-------------------------------------------------------------------------------------|
| SISTEMA DE INVENTARIOS<br>ACTUALIZACIÓN ARCH. ARTÍCULOS                             |
| 1. ALTAS DE ARTÍCULOS<br>2. BAJAS DE ARTÍCULOS<br>3. CAMBIOS DE ARTÍCULOS<br>4. FIN |
| ESCOGER OPCIÓN                                                                      |

|                                                                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SISTEMA DE INVENTARIOS<br>ALTAS DE ARTÍCULOS                                                                                                                                                                                   |
| DESCRIPCIÓN:<br>UNIDAD MEDIDA:<br>LOCALIZACIÓN:<br>EXISTENCIA:<br>STOCK MÍNIMO:<br>STOCK MÁXIMO:<br>PRECIO ÚLTIMA COMPRA:<br>PRECIO DE VENTA:<br>FECHA ÚLTIMA ENTRADA:<br>FECHA ÚLTIMA SALIDA:<br>PROVEEDOR 1:<br>PROVEEDOR 2: |
| ¿OTRA ALTA (S/N)?:                                                                                                                                                                                                             |

| SISTEMA DE INVENTARIOS<br>BAJAS DE ARTÍCULOS                                                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DESCRIPCIÓN:<br>UNIDAD MEDIDA:<br>LOCALIZACIÓN:<br>EXISTENCIA:<br>STOCK MÍNIMO:<br>STOCK MÁXIMO:<br>PRECIO ÚLTIMA COMPRA:<br>PRECIO DE VENTA:<br>FECHA ÚLTIMA ENTRADA:<br>FECHA ÚLTIMA SALIDA:<br>PROVEEDOR 1:<br>PROVEEDOR 2: |
| ¿OTRA BAJA (S/N)?:                                                                                                                                                                                                             |

| SISTEMA DE INVENTARIOS<br>CAMBIOS DE ARTÍCULOS                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1: LÍNEA:<br>2: DESCRIPCIÓN:<br>3: UNIDAD MEDIDA:<br>4: LOCALIZACIÓN:<br>5: EXISTENCIA:<br>6: STOCK MÍNIMO:<br>7: STOCK MÁXIMO:<br>8: PRECIO ÚLTIMA COMPRA:<br>9: PRECIO DE VENTA:<br>10: FECHA ÚLTIMA ENTRADA:<br>11: FECHA ÚLTIMA SALIDA:<br>12: PROVEEDOR 1:<br>13: PROVEEDOR 2: |
| NÚMERO DE DATO A CAMBIAR (0=FIN):<br>¿OTRO CAMBIO (S/N)?:                                                                                                                                                                                                                           |

|                                                  |
|--------------------------------------------------|
| SISTEMA DE INVENTARIOS<br>CAPTURA DE MOVIMIENTOS |
| NÚMERO:                                          |
| DESCRIPCIÓN:                                     |
| TIPO DE MOVIMIENTO (E=ENTRADA S=SALIDA):         |
| FECHA:                                           |
| CANTIDAD (UNIDADES):                             |
| PRECIO:                                          |
| PROVEEDOR 2:                                     |
| ¿OTRO MOVIMIENTO (S/N)?:                         |

|                                                  |
|--------------------------------------------------|
| SISTEMA DE INVENTARIOS<br>CONSULTAS DE ARTÍCULOS |
| NÚMERO:                                          |
| LÍNEA:                                           |
| DESCRIPCIÓN:                                     |
| UNIDAD MEDIDA:                                   |
| LOCALIZACIÓN:                                    |
| EXISTENCIA:                                      |
| STOCK MÍNIMO:                                    |
| STOCK MÁXIMO:                                    |
| PRECIO ÚLTIMA COMPRA:                            |
| PRECIO DE VENTA:                                 |
| FECHA ÚLTIMA ENTRADA:                            |
| FECHA ÚLTIMA SALIDA:                             |
| PROVEEDOR 1:                                     |
| PROVEEDOR 2:                                     |
| ¿OTRA CONSULTA (S/N)?:                           |

| SISTEMA DE INVENTARIOS<br>CONSULTAS DE ARTÍCULOS                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NÚMERO:<br>1. CATÁLOGO DE ARTÍCULOS<br>2. EXISTENCIA VALORIZADA<br>3. ARTÍCULOS CON POCO MOVIMIENTO<br>4. LISTA DE PRECIOS<br>5. ARTÍCULOS BAJO MÍNIMO<br>6. LISTADO DE PEDIDOS<br>7. FIN: |
| ESCOGER OPCIÓN                                                                                                                                                                             |

## 8.8 Resumen de conceptos que debe dominar

- Registros y archivos (conceptos generales)
- Manejo de registros en seudocódigo
- Manejo de archivos en seudocódigo
- Proceso de un archivo secuencial
- Proceso de un archivo directo
- Desarrollar algoritmos grandes conformando sistemas de información

## 8.9 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

- 8.9.1 Resumen gráfico del capítulo
- 8.9.2 Autoevaluación
- 8.9.3 Programas
- 8.9.4 Ejercicios resueltos
- 8.9.5 Power Point para el profesor (\*)

# 9

## Otros tipos de datos y otros temas

### Contenido

- 9.1 Clasificación (ordenación) de datos
- 9.2 Tipos de datos definidos por el usuario (Tipos)
- 9.3 Apuntadores (Pointer)
- 9.4 Recursividad en seudocódigo
- 9.5 Ejecución de otros programas en código ejecutable
- 9.6 Graficación en seudocódigo
- 9.7 Incluir archivos de programas
- 9.8 Resumen de conceptos que debe dominar
- 9.9 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*)  
sólo está disponible para docentes.
  - 9.9.1 Resumen gráfico del capítulo
  - 9.9.2 Autoevaluación
  - 9.9.3 Programas
  - 9.9.4 Power Point para el profesor (\*)

### Objetivos del capítulo

- Estudiar la clasificación (ordenación) de datos.
- Estudiar tipos de datos definidos por el usuario (Tipos).
- Estudiar apuntadores (Pointer).
- Estudiar la recursividad en seudocódigo.
- Aprender a ejecutar otros programas en código ejecutable.
- Estudiar la graficación en seudocódigo.
- Estudiar la inclusión de archivos de programas.

## Introducción

Con el estudio del capítulo anterior, usted ya domina las estructuras de datos registros y archivos; y cómo diseñar algoritmos usando esas estructuras.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando otros tipos de datos y otros temas varios, que permiten reforzar la técnica presentada, mismos que sirven como introducción a otros temas más avanzados y hacen que la metodología de este libro sea útil para diseñar programas que van más allá del alcance de esta obra.

Se explica que la clasificación (ordenación) de datos quiere decir ordenar los datos en orden secuencial, ya sea en orden ascendente o descendente, que existen diversos métodos que permiten llevar a cabo la clasificación u ordenamiento, y se muestra el uso del método denominado intercambio o burbuja.

Se presenta el concepto tipos de datos definidos por el usuario (Tipos), que quiere decir que los lenguajes de programación permiten que el usuario (programador) defina sus propios tipos de datos, y que existen dos categorías de nuevos tipos de datos que se pueden definir: el tipo enumerado y el tipo subrango.

Se estudia el concepto de apuntadores (Pointer); un apuntador es una variable que toma como valor la dirección de una posición de la memoria, es decir, “apunta” hacia una posición de memoria.

Se expone que los lenguajes de programación soportan la recursividad, esta significa que un elemento puede definirse en términos de sí mismo.

Se puntualiza que desde un programa o algoritmo se pueden ejecutar otros programas que estén en código ejecutable.

Se explica el uso de diversas funciones para graficar figuras como círculos, arcos, barras, etcétera.

Se describe cómo incluir archivos de programas.

En el siguiente capítulo se estudia la programación orientada a objetos usando el diagrama de clases.

## 9.1 Clasificación (ordenación) de datos

Clasificar quiere decir ordenar los datos en orden secuencial, ya sea en orden ascendente o descendente. Por ejemplo, si se tienen los valores 3 7 2 9 4 1, se dice que están desordenados, porque no existe un orden secuencial en la forma en que están organizados. Al ordenarlos en forma secuencial ascendente quedan en el siguiente orden: 1 2 3 4 7 9; o bien, si se ordenan en forma descendente quedan: 9 7 4 3 2 1.

### Métodos de ordenamiento o clasificación

Existen diversos métodos que permiten llevar a cabo la clasificación u ordenamiento de datos, por ejemplo, existe un método denominado intercambio o burbuja, el cual consiste en el siguiente proceso:

1. El primer elemento se compara con el segundo; si es mayor el primero que el segundo, se intercambian, es decir, el primero se coloca en la posición del segundo y el segundo en la posición del primero.
2. El segundo elemento se compara con el tercero; si es mayor, se intercambian. Luego se compara el tercero con el cuarto, haciendo lo propio, y así sucesivamente hasta el último elemento del arreglo.
3. Repetir los pasos 1 y 2 hasta que se dé el caso de que no se haya hecho ningún intercambio; lo que significa que el arreglo ya quedó ordenado, terminando el proceso de clasificación.

A este método se le conoce como la burbuja porque los elementos que están en una posición más abajo de acuerdo con su magnitud, tienden a “subir” hacia la posición que les corresponde, algo similar a lo que sucede en una botella de alguna bebida gaseosa donde las burbujas de aire suben.

Ejemplo:

Elaborar un algoritmo que permita leer 10 números enteros en un arreglo de 10 elementos, que imprima el arreglo, luego que clasifique los elementos del arreglo en orden ascendente y lo imprima.

A continuación se tiene el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*

```
Algoritmo CLASIFICA NÚMEROS
1. Declarar
    Variables
        numeros: Arreglo[10] Entero
        auxiliar, c, bandera: Entero
    2. for c=0; c<=9; c++
        a. Solicitar Número c
        b. Leer numeros[c]
    3. endfor
```

```
4. for c=0; c<=9; c++
   a. Imprimir numeros[c]
5. endfor
6. do
   a. bandera = 0
   b. for c=1; c<=9; c++
      1. if numeros[c-1] > numeros[c] then
         a. auxiliar = numeros[c]
         b. numeros[c] = numeros[c-1]
         c. numeros[c-1] = auxiliar
         d. bandera = 1
      2. endif
   c. endfor
7. while bandera != 0
8. for c=0; c<=9; c++
   a. Imprimir numeros[c]
9. endfor
10.Fin
```



En la zona de descarga de la Web del libro, está disponible:  
Programa en C: C901.C

*Explicación:*

1. Se hacen las declaraciones de variables
- 2y3. Se leen los números en numeros
- 4y5. Se imprimen los números
6. Inicia ciclo de clasificación do
  - a. Se pone la bandera en 0
  - b. Inicia ciclo for desde c = 1 hasta 9
    1. Compara Si el valor del elemento numeros[c-1] > numeros[c] Entonces
      - a. En auxiliar se coloca el elemanto numeros[c]
      - b. En numeros[c] se coloca el valor del elemento numeros[c-1]
      - c. En numeros[c-1] se coloca el valor de auxiliar
      - d. Se coloca la bandera en 1 (se apaga la bandera)
    2. Fin if
    - c. Fin del for
  7. Fin del do while bandera != 0
  - 8y9. Se imprime el arreglo numeros ya clasificado

Otro ejemplo:

Elaborar un algoritmo similar al anterior, excepto que ahora se deberán leer y clasificar 10 nombres de personas.

A continuación se tiene el algoritmo de la solución:

(Primero hágalo usted, después compare la solución.)

```
Algoritmo CLASIFICA NOMBRES
1. Declarar
    Variables
        nombres: Arreglo[10] Cadena[30]
        auxiliar: Cadena[30]
        c, bandera: Entero
2. for c=0; c<=9; c++
    a. Solicitar nombre, c
    b. Leer nombres[c]
3. endfor
4. for c=0; c<=9; c++
    a. Imprimir nombres[c]
5. endfor
6. do
    a. bandera = 0
    b. for c=1; c<=9; c++
        1. if nombres[c-1] > nombres[c] THEN
            a. auxiliar = nombres[c]
            b. nombres[c] = nombres[c-1]
            c. nombres[c-1] = auxiliar
            d. bandera = 1
        2. endif
    c. endfor
7. while bandera != 0
8. for c=0; c<=9; c++
    a. Imprimir nombres[c]
9. endfor
10.Fin
```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C902.C



*Explicación:*

1. Se hacen las declaraciones de variables
- 2y3. Se leen los nombres en nombres
- 4y5. Se imprimen los nombres

6. Inicia ciclo de clasificación do
  - a. Se pone la bandera en 0
  - b. Inicia ciclo for desde c = 1 hasta 9
    1. Compara Si el valor del elemento nombres[c-1] > nombres[c] Entonces
      - a. En auxiliar se coloca el elemanto nombres[c]
      - b. En nombres[c] se coloca el valor del elemento nombres[c-1]
      - c. En nombres[c-1] se coloca el valor de auxiliar
      - d. Se coloca la bandera en 1 (se apaga la bandera)
    2. Fin if
    - c. Fin del for
  7. Fin del do while bandera != 0
- 8y9. Se imprime el arreglo nombres ya clasificado

### Clasificación (ordenación) de un archivo

Ahora apliquemos el método de clasificación en un archivo.

Ejemplo:

Se tiene el archivo de empleados (AREMDI.DAT) que fue creado en el capítulo ocho en el punto de proceso de un archivo directo, el cual está ordenado por número de empleado. Tomar los registros de dicho archivo y crear otro archivo (AREMDI.CLA), donde se coloquen en orden secuencial ascendente por nombre e imprimir el catálogo de empleados clasificado por nombre.

```

Algoritmo CATÁLOGO DE EMPLEADOS CLASIFICADO
1. Declarar
    Tipos
        persona: Registro
            numero: Entero
            nombre: Cadena[30]
            depto: Entero
            puesto: Entero
            sueldo: Real
        FinRegistro
    Variables
        regEmpleado: persona
        empleados, empleados2: Archivo de persona
        nombres: Arreglo[100] Cadena[30]
        direcciones: Arreglo[100] Entero
        auxiliar: Cadena[30]
        i, c, bandera, auxiliar2: Entero

```

```
2. Función principal()
    a. clasificaArchivo()
    b. imprimeCatalogo()
    c. Fin Función principal
3. Función clasificaArchivo()
    a. Abrir (empleados, "AREMDI.DAT", directo, rw)
    b. Leer (empleados, regEmpleado)
    c. c = 0
    d. while NOT(Eof(empleados))
        1. c = c + 1
        2. nombres[c] = regEmpleado.nombre
        3. direcciones[c] = PosicionAr(empleados)
        4. Leer (empleados, regEmpleado)
    e. endwhile
    f. do
        1. bandera = 0
        2. for c=1; c<=99; c++
            a. if nombres[c-1] > nombres[c] then
                1. auxiliar = nombres[c]
                2. auxiliar2 = direcciones[c]
                3. nombres[c] = nombres[c-1]
                4. direcciones[c] = direcciones[c-1]
                5. nombres[c-1] = auxiliar
                6. direcciones[c-1] = auxiliar2
                7. bandera = 1
            b. endif
        3. endfor
    g. while bandera != 0
        /* Crea el archivo que contendrá los datos
           ordenados */
    h. Crear(empleados2, "AREMDI.CLA", secuencial, w)
    i. for i=0; i<=c; i++
        1. Encontrar (empleados, direcciones[i])
        2. Leer (empleados, regEmpleado)
        3. Imprimir (empleados2, regEmpleado)
    j. endfor
    k. Cerrar (empleados)
    l. Cerrar (empleados2)
    m. Fin Función clasificaArchivo
4. Función imprimeCatalogo()
    a. Declarar
        Variables
        totEmp: Entero
        totSueldos: Real
    b. totEmp = 0
        totSueldos = 0
```

```

c. Imprimir encabezado
d. Abrir (empleados2, "AREMDI.CLA", secuencial, r)
e. Leer (empleados2, regEmpleado)
f. while NOT(Eof(empleados2))
   1. Imprimir regEmpleado.numero,
      regEmpleado.nombre, regEmpleado.depto,
      regEmpleado.puesto, regEmpleado.sueldo
   2. totEmp = totEmp + 1
   3. totSueldos = totSueldos + regEmpleado.sueldo
   4. Leer (empleados2, regEmpleado)
g. endwhile
h. Imprimir totEmp, totSueldos
i. Cerrar (empleados2)
j. Fin Función imprimeCatalogo
Fin

```

En la zona de descarga de la Web del libro, está disponible:



Programa en C: C903.C

*Explicación:*

Se utiliza el archivo “AREMDI.DAT”, el cual fue creado en el capítulo 8.

Se crea un nuevo archivo “AREMDI.CLA”, el cual contendrá los registros clasificados por nombre.

Para hacer la clasificación:

Se utilizan dos arreglos nombres y direcciones.

Se lee el primer registro del archivo “AREMDI.DAT”, se coloca el nombre en el arreglo nombres, y el número de componente del archivo en el que se encuentra, se coloca en el arreglo direcciones, es decir, se colocan en los arreglos el nombre y la dirección que ocupa en el archivo original. Esto se hace para todos los registros del archivo. Los arreglos se definieron de 100 elementos, porque se está suponiendo que el archivo no tiene más de 100 elementos. Si se hace un algoritmo para clasificar un archivo de alrededor de 400 elementos, los arreglos deben definirse de un poco más, por ejemplo, de 500.

Se hace el ordenamiento de los datos de los arreglos, se va moviendo el nombre en el arreglo nombres y la dirección correspondiente en el arreglo direcciones.

Una vez que los datos han quedado ordenados en los arreglos, se procede a crear el nuevo archivo “AREMDI.CLA”, luego se toma la primera dirección del arreglo direcciones y se lee ese componente del archivo original “AREMDI.DAT”, posteriormente se graban los datos del registro en el nuevo archivo “AREMDI.CLA”; se toma la siguiente dirección y se hace lo propio, y así sucesivamente hasta que se terminen las direcciones.

Después de lo anterior tenemos el nuevo archivo “AREMDI.CLA” con los datos ordenados.

Por último se procede a enlistar los registros del nuevo archivo, emitiendo el catálogo de empleados clasificado por nombre.

### Clasificación de un archivo por dos datos concatenados

Ejemplo:

Se tiene el archivo de empleados (AREMDI.DAT) que fue creado en el capítulo ocho en el punto de proceso de un archivo directo, el cual está ordenado por número de empleado. Tomar los registros de dicho archivo y crear otro archivo (AREMDI.CLA), donde se coloquen en orden secuencial ascendente por departamento y por nombre e imprimir el catálogo de empleados clasificado por departamento y por nombre. Es decir, se clasifica por departamento, y dentro de cada departamento, estarán ordenados por nombre.

```
Algoritmo CATÁLOGO DE EMPLEADOS CLASIFICADO POR DEPTO.  
Y NOMBRE  
1. Declarar  
    Tipos  
        persona: Registro  
            numero: Entero  
            nombre: Cadena[30]  
            depto: Entero  
            puesto: Entero  
            sueldo: Real  
        FinRegistro  
    Variables  
        regEmpleado: persona  
        empleados, empleados2: Archivo de persona  
        nombres: Arreglo[100] Cadena[30]  
        direcciones: Arreglo[100] Entero  
        auxiliar: Cadena[30]  
        i, c, bandera, auxiliar2: Entero  
        depAlf: Cadena[5]  
2. Función principal()  
    a. clasificaArchivo()  
    b. imprimeCatalogo()  
    c. Fin Función principal  
3. Función clasificaArchivo()  
    a. Abrir (empleados, "C:\ AREMDI.DAT", directo, rw)  
    b. Leer (empleados, regEmpleado)  
    c. i = -1
```

```
d. while NOT(Eof(empleados))
   1. i = i + 1
   2. Cadena(regEmpleado.depto, deptoAlf)
   3. nombres[c] = deptoAlf + regEmpleado.nombre
   4. direcciones[c] = PosicionAr(empleados)
   5. Leer (empleados, regEmpleado)
e. endwhile
f. do
   1. bandera = 0
   2. for c=1; c<=i; c++
      a. if nombres[c-1] > nombres[c] then
          1. auxiliar = nombres[c]
          2. auxiliar2 = direcciones[c]
          3. nombres[c] = nombres[c-1]
          4. direcciones[c] = direcciones[c-1]
          5. nombres[c-1] = auxiliar
          6. direcciones[c-1] = auxiliar2
          7. bandera = 1
      b. endif
   3. endfor
g. while bandera != 0
   /* Crea el archivo que contendrá los datos
      ordenados */
h. Crear (empleados2, "AREMDI.CLA", secuencial, w)
i. for c=1; c<=i; c++
   1. Encontrar (empleados, direcciones[i])
   2. Leer (empleados, regEmpleado)
   3. Imprimir (empleados2, regEmpleado)
j. endfor
k. Cerrar (empleados)
l. Cerrar (empleados2)
m. Fin Función clasificaArchivo
4. Función imprimeCatalogo()
   a. Declaraciones
      Variables
         totEmp: Entero
         totSueldos: Real
   b. totEmp = 0
      totSueldos = 0
   c. Imprimir encabezado
   d. Abrir (empleados2, "C:\AREMDI.CLA", secuencial, r)
   e. Leer (empleados2, regEmpleado)
```

```
f. while NOT(Eof(empleados2))
    1. Imprimir regEmpleado.numero,
       regEmpleado.nombre, regEmpleado.depto,
       regEmpleado.puesto, regEmpleado.sueldo
    2. totEmp = totEmp + 1
    3. totSueldos = totSueldos + regEmpleado.sueldo
    4. Leer (empleados2, regEmpleado)
g. endwhile
h. Imprimir totEmp, totSueldos
i. Cerrar (empleados2)
j. Fin Función imprimeCatalogo
Fin
```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C904.C



*Explicación:*

El algoritmo de clasificación es el mismo, excepto las acciones:

Cadena (regEmpleado.depto, deptoAlf)  
en la cual se convierte en tipo cadena el dato departamento del empleado.  
nombres [c] = deptoAlf + regEmpleado.nombre  
En el elemento c del arreglo nombres se coloca el departamento concatenando con el nombre del empleado. Así, al hacer el ordenamiento, se contempla en primer término el departamento y en segundo término el nombre.

**Nota:** En este punto se ha dado una idea general de la clasificación y se ha estudiado un método de los más sencillos, existen varios libros de estructura de datos que tratan el tema con toda profundidad, el cual va más allá del alcance de este libro, sin embargo, es relevante la aplicación en la clasificación del archivo.

## 9.2 Tipos de datos definidos por el usuario (Tipos)

Los lenguajes de programación permiten que el usuario (programador) defina sus propios tipos de datos, es decir, que si los tipos de datos estándar no se adaptan a alguna situación que se desee manejar, el usuario puede optar por definir nuevos tipos de datos. Además del uso que ya se le ha dado a la definición de tipos, existen dos categorías de nuevos tipos de datos que se pueden definir: el tipo enumerado y el tipo subrango.

### Tipo enumerado

El tipo de dato enumerado está compuesto por una secuencia ordenada de valores, donde cada uno de éstos es un elemento del tipo de dato. Para definirlo se utiliza el siguiente:

*Formato:*

Declarar

Tipos

```
nombreTipo = (valor1, valor2, valor3, ..., valorN)
```

*En donde:*

Tipos Es la palabra reservada que indica que se está definiendo un nuevo tipo de datos.

nombreTipo Es el nombre de identificación que se le está dando al nuevo tipo de datos.

= Es parte del formato.

(valor1, valor2, valor3, ..., valorN) Son los valores que componen el nuevo tipo de datos, éstos se enumeran en secuencia de acuerdo con el lugar que ocupa cada uno dentro del tipo de datos, cada valor ocupa un número de lugar en la secuencia; el primero es el 1, el segundo es el 2, etcétera.

*Ejemplo:*

Declarar

Tipos

```
dias=(Domingo,Lunes,Martes,Miércoles,Jueves,  
Viernes,Sábado)
```

Variables

```
dia: dias
```

*Explicación:*

- Se define un nuevo tipo de datos `dias`, el cual está compuesto por los valores enumerados.
- Se define la variable `dia`, la cual es de tipo `dias`; esto quiere decir que la variable `dia` sólo puede tomar los valores del tipo de dato `dias`.

Puede ser utilizado para controlar un for

```
for dias=Lunes; dias<=Viernes; dias++  
---  
---  
---  
endfor
```

El tipo de dato enumerado no puede ser leído ni escrito, puede ser asignado por ejemplo:  
`dia = Lunes`



Puede ser utilizado para controlar un switch:

```
switch dia
    Domingo : Imprimir "Domingo"
    Lunes   : Imprimir "Lunes"
    Martes  : Imprimir "Martes"
    Miércoles: Imprimir "Miércoles"
    Jueves  : Imprimir "Jueves"
    Viernes : Imprimir "Viernes"
    Sábado  : Imprimir "Sábado"
endswitch
```

Otro ejemplo:

Declarar

```
Tipos
   edoCiv = (Soltero,Casado,Viudo,Divorciado,
              Unión libre)
   meses = (Enero,Febrero,Marzo,Abril,Mayo,Junio,
              Julio,Agosto,Septiembre,Octubre,Noviembre,
              Diciembre)
   baraja = (Espadas,Oros,Copas,Bastos)
Variables
    estadoCivil: edoCiv
    mes: meses
    carta: baraja
```

*Explicación:*

- En la parte de definición de tipos se definen tres nuevos tipos de datos: `edoCiv`, `meses` y `baraja`.
- Se definen variables con esos tipos de datos definidos: `estadoCivil` de tipo `edoCiv`, `mes` de tipo `meses` y `carta` de tipo `baraja`.

### Tipo subrango

Algunos lenguajes como Pascal y derivados de éste, manejan el tipo de dato subrango, el cual se define tomando como base cualquier tipo de datos ya existente, que esté constituido por un conjunto ordenado y finito de valores (el real no cumple con esta característica), por subrango se entiende una parte del rango original, por ejemplo, se puede plantear un subrango del tipo Entero, o bien, del tipo carácter, entre otros que cumplan con la característica antes descrita. Se define con el siguiente formato:

Declarar

```
Tipos
    nombreTipo = límiteInferior..límiteSuperior
```

*En donde:*

límiteInferior.. Son los límites del subrango, se indica el primer valor y  
 límiteSuperior el último valor del subrango, por ejemplo, 1..5, es un subrango de los enteros desde 1 hasta 5.

Ejemplo:

Declarar

```
Tipos
    dias = (Domingo,Lunes,Martes,Miércoles,Jueves,
             Viernes,Sábado)
    diaHabil = Lunes..Viernes
    mayusculas = 'A'..'Z'
    nuMes = 1..12
    diasMes = 1..31
    horas = 1..60
Variables
    diaTrabajo,diaVacacion: diaHabil
    letra: mayusculas
    mes: nuMes
    dia: diasMes
    horasTrabajadas: horas
```

*Explicación:*

- Se define el nuevo tipo de dato dias; luego se define el nuevo tipo diaHabil como un subrango del tipo dias. También se definen nuevos tipos de datos tomando como base los tipos de datos Carácter y Entero.
- Se declaran variables utilizando los nuevos tipos de datos definidos.

### Subrangos en la definición de variables

Al definir variables se pueden definir como de tipo subrango, ejemplos:

Declarar

```
Variables
    dia: 1..31
    mes: 1..12
    anio: 1900..2050
```

En este ejemplo se están definiendo tres variables tomando como base el tipo Entero, por tanto, serán de tipo entero.

### 9.3 Apuntadores (Pointer)

Un apuntador es una variable que toma como valor la dirección de una posición de la memoria, es decir, “apunta” hacia una posición de memoria.

Esquemáticamente:



*Explicación:*

La variable ap es un apuntador cuyo valor es 1025, es decir, “apunta” a la posición 1025, la cual contiene un valor.

#### Formato para definir variables tipo apuntador (Pointer)

Declarar

```

Variables
  ap: *Tipo de dato
  
```

*En donde:*

|              |                                                                                                                      |
|--------------|----------------------------------------------------------------------------------------------------------------------|
| ap           | Es el nombre de la variable.                                                                                         |
| *            | Es el símbolo que identifica el apuntador (Pointer), el cual indica “apunta hacia”,                                  |
| Tipo de dato | Es el tipo de dato que se almacenará en la posición a la cual apunta el apuntador, puede ser Entero, Real, etcétera. |

Ejemplo:

Declarar

```

Variables
  ap: *Entero
  
```

*Explicación:*

|    |                                                                                                                   |
|----|-------------------------------------------------------------------------------------------------------------------|
| ap | Es una variable tipo apuntador (Pointer) que apunta a una dirección en la que se almacenará un valor tipo Entero. |
|----|-------------------------------------------------------------------------------------------------------------------|

Esquemáticamente:



Una forma más general de definir apuntadores es definirlos en Tipos, ejemplo:  
Declarar

```
Tipos
    apuntaEntero = *Entero
Variables
    ap: apuntaEntero
```

*Explicación:*

- Se define `apuntaEntero` como un tipo de dato apuntador a un Entero.
- Se define la variable `ap` de tipo `apuntaEntero`, es decir, como un apuntador a un tipo Entero.

### Apuntador a un tipo Real

Declarar

```
Variables
    x: *Real
```

`x` es un apuntador a una dirección que contiene un dato de tipo Real. Esquemáticamente:



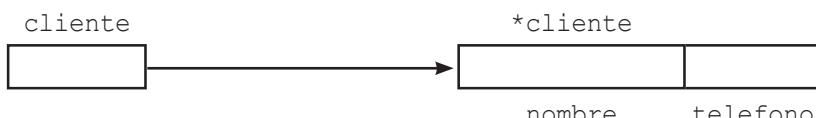
### Apuntador a un registro

Declarar

```
Tipos
    persona: Registro
        nombre: Cadena[30]
        telefono: Cadena[9]
    FinRegistro
    ap = *persona
Variables
    cliente: ap
```

*Explicación:*

- |                      |                                                                                                   |
|----------------------|---------------------------------------------------------------------------------------------------|
| <code>persona</code> | es un registro con los datos <code>nombre</code> y <code>telefono</code> .                        |
| <code>ap</code>      | es un apuntador a una dirección que contiene dato tipo <code>persona</code> .                     |
| <code>cliente</code> | es una variable de tipo <code>ap</code> , es decir, un apuntador a un tipo <code>persona</code> . |



## Variables estáticas, estructuras dinámicas y manejo de memoria

### Variables estáticas

Son las variables normales que hemos utilizado hasta el capítulo anterior, es decir, definimos la variable con algún tipo de dato, donde ya conocemos la cantidad de valores que tomará cada variable; por ejemplo:

```
Declarar
    Variables
        x, y: Real
        i, j: Entero
        m: Arreglo[10] Entero
```

Todas son estáticas, porque se define de antemano la cantidad de memoria que se requerirá.

### Variables dinámicas o estructuras de datos dinámicas

Son estructuras de datos en las que no se conoce la cantidad exacta de valores que se deberán almacenar, es decir, la cantidad de memoria a utilizar varía en el momento de la ejecución.

El apuntador (pointer) es un tipo de dato dinámico que permite el manejo de variables o estructuras de datos dinámicas.



Las variables dinámicas (apuntador) no pueden utilizarse para leer o imprimir su contenido.

### Manejo de memoria

Los lenguajes como Pascal o C, dividen la memoria de la computadora en:

- Segmento de código.
- Segmento de datos.
- Segmento de pila (Stack).
- Segmento de variables dinámicas (Heap).

En el segmento de código se almacenan las instrucciones de programa. En el segmento de datos se almacenan las variables estáticas. En el segmento de pila (Stack) se almacenan los datos temporales. En el segmento de variables dinámicas (Heap) se almacenan las variables tipo apuntador (pointer). El Heap (montículo) es un segmento de memoria en donde se almacenan los datos de tipo apuntador.

### Manejo de la memoria para variables dinámicas

Cuando se utilizan variables dinámicas (Pointer) es necesario asignar espacio en el segmento de memoria dinámica (Heap) para su almacenamiento, y después de utilizarlas se requiere que dicho espacio sea liberado. Con las funciones AsignarMem y LiberarMem se asigna y libera el espacio en la memoria Heap para las variables dinámicas.

Toda variable tipo apuntador primero debe declararse como se explicó anteriormente; después se le debe asignar memoria dinámica (del Heap) con AsignarMem, luego podrá utilizarse para tomar valores; y por ultimo, cuando no se necesite se debe liberar la memoria que tiene asignada con LiberarMem.

### AsignarMem

La función AsignarMem asigna espacio en el segmento de variables dinámicas (Heap) para el elemento apuntado por la variable dinámica correspondiente.

*Formato:*

```
AsignarMem (apuntador)
```

*En donde:*

|            |                                                |
|------------|------------------------------------------------|
| AsignarMem | Es la función que asigna espacio (en el Heap). |
| apuntador  | Es el nombre de la variable apuntador.         |

### LiberarMem

La función LiberarMem libera el espacio que fue asignado a una variable dinámica.

*Formato:*

```
LiberarMem (apuntador)
```

*En donde:*

|            |                                              |
|------------|----------------------------------------------|
| LiberarMem | Es la función que libera espacio (del Heap). |
| apuntador  | Es la variable dinámica (apuntador).         |

*Ejemplo:*

```
Algoritmo APUNTADOR
1. Declarar
    Variables
    i,j: *Entero
2. AsignarMem(i)
3. AsignarMem(j)
4. *i = 25
5. *j = *i - 10
6. Imprimir "APUNTADOR i = ", *i
7. Imprimir "APUNTADOR j = ", *j
8. LiberarMem(i)
9. LiberarMem(j)
10.Fin
```

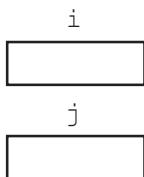


En la zona de descarga de la Web del libro, está disponible:

Programa en C: C905.C

*Explicación:*

1. Se declaran las variables i y j como apuntadores a Entero.



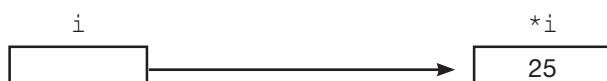
2. Se asigna espacio para \*i (a donde apunta i) en el Heap.



3. Se asigna espacio para \*j (a donde apunta j) en el Heap.



4. A donde apunta i (\*i) se le asigna 25.



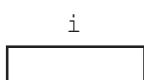
5. A donde apunta j (\*j) se le asigna el valor resultante de restarle 10 a donde apunta i (\*i), es decir 15.



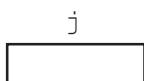
6. Se imprime el contenido de \*i (a donde apunta i).

7. Se imprime el contenido de \*j (a donde apunta j).

8. Se libera la memoria de donde apunta i, queda:



9. Se libera la memoria de donde apunta j, queda:



10. Fin del algoritmo

### La constante NULO

Algunos lenguajes tienen predefinida la constante NULO que significa sin valor.

Si se coloca NULO a una variable apuntador significa que dicha variable apunta a nulo, es decir, a nada.

### El operador @

El operador @ permite obtener la dirección de una variable.

Ejemplo:

```
Variables
z,i: Entero
x: *z
x = @z {Coloca en x la dirección de z}
```

### Asignación de una variable apuntador a otra

Se puede asignar el contenido de una variable dinámica a otra, ejemplo:

```
Variables
x: *Entero
y: *Entero
*x = 10
y = x
```

*Explicación:*

\*x = 10 A donde apunta x se le coloca 10.

y = x y toma la dirección de x que es un apuntador, por lo tanto, a donde apunta x; y, a donde apunta y, es lo mismo.

Ejemplo:

A continuación se presenta un programa que permite leer varios nombres de personas formando una lista enlazada de elementos donde cada elemento contiene el nombre de una persona y un apuntador al siguiente elemento, es decir, a la siguiente persona de la lista; el último componente (elemento) de la lista apuntará a nulo, es decir, a ningún componente. Habrá un apuntador que es la cabeza de la lista, es decir, apunta al primer componente de la lista de nombres.

```
Algoritmo LISTA ENLAZADA DE NOMBRES
1. Declarar
    Tipos
        apuntador: *persona
        persona: Registro
            nombre: Cadena[30]
            siguiente: apuntador
        FinRegistro
    Variables
        cabeza, nuevo, ultimo: apuntador
        otro: Carácter
        /* Lectura de nombres */
```

```
2. cabeza = nulo
3. do
    a. if Cabeza == NULO then
        1. AsignarMem(cabeza)
        2. Solicitar Nombre
        3. Leer *cabeza.nombre
        4. ultimo = cabeza
        5. *cabeza.siguiente = NULO
    b. else
        1. AsignarMem(nuevo)
        2. Solicitar Nombre
        3. Leer *nuevo.nombre
        4. *nuevo.siguiente = NULO
        5. *ultimo.siguiente = nuevo
        6. ultimo = nuevo
    c. endif
    d. Preguntar ¿Desea introducir otro nombre (S/N)?
    e. Leer otro
4. while otro == 'S'
    /* Listado de nombres */
5. nuevo = cabeza
6. while NOT(nuevo == NULO)
    a. Imprimir *nuevo.nombre
    b. nuevo = *nuevo.siguiente
7. endwhile
    /* Liberar espacio */
8. nuevo = cabeza
9. while NOT(nuevo == NULO)
    a. nuevo = *nuevo.siguiente
    b. LiberarMem(nuevo)
10.endwhile
11.Fin
```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C906.C



*Explicación:*

apuntador      Es un tipo de dato apuntador (pointer), el cual apunta a persona.

persona      Es un tipo registro que contiene los datos: nombre para leer el nombre de la persona y siguiente que es un apuntador que apunta hacia persona, es decir, indica el siguiente componente de la lista (que es otra persona).

|        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cabeza | Es una variable apuntador que se inicia con nulo; al crear el primer componente toma la dirección del mismo, y siempre apuntará ahí; en otras palabras, indica el primer componente de la lista (es decir, la cabeza de la lista).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| nuevo  | Es también un apuntador a persona que se utiliza para crear un nuevo elemento de la lista, en el cual se lee el nombre de la persona; y en el dato siguiente que es un apuntador a persona, se coloca nulo, es decir, no apunta a ningún otro componente.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| ultimo | Es un apuntador que siempre apunta al último componente de la lista, es decir, tiene la dirección del último componente de la lista. En la parte final del programa, se listan los nombres de los componentes de la lista.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| nuevo  | Toma el valor de cabeza, es decir, la dirección del primer componente de la lista. A continuación se plantea un ciclo while, al cual entrará mientras nuevo no apunte a nulo. Dentro del ciclo imprime el nombre del componente en proceso (nuevo), luego nuevo tomará la dirección del siguiente componente, para volver al encabezado del ciclo; y sigue así hasta que nuevo apunte a nulo. Por último se plantea un ciclo while, al cual entrará mientras nuevo no apunte a nulo. Dentro del ciclo libera el espacio por el componente en proceso (nuevo), luego nuevo tomará la dirección del siguiente componente, para volver al encabezado del ciclo; y sigue así hasta que nuevo apunte a nulo, es decir, se libera todo el espacio ocupado de la memoria dinámica. |

## 9.4 Recursividad en seudocódigo

Los lenguajes de programación soportan la recursividad, ésta significa que un elemento puede definirse en términos de sí mismo. Se implementa este concepto a través de funciones, las cuales pueden llamarse a sí mismas.

Ejemplo:

Elaborar un algoritmo que lea un número e imprima los números desde ese número hasta 1 con decrementos de 1; utilizar una función recursiva, es decir, que se llame a sí misma.

```

Algoritmo RECURSIVIDAD
1. Función principal()
   a. Declarar
      Variables
      num: Entero

```

```

    b. Solicitar un numero
    c. Leer num
    d. imprimeNumero(num)
    e. Fin Función principal
2. Función imprimeNumero(Val x: Entero)
    a. Imprimir x
    b. if x > 1 then
        1. x = x - 1
        2. imprimeNumero(x)
    c. endif
    d. Fin Función imprimeNumero
Fin

```

En la zona de descarga de la Web del libro, está disponible:

Programa en C: C907.C



#### *Explicación:*

En la Función principal() se lee num, luego se llama a la función imprimeNúmero, imprimeNúmero es una función que recibe el parámetro en x; imprime el valor de x, le disminuye 1 a x y llama a la función imprimeNúmero con el nuevo valor de x como parámetro, es decir, se llama a sí misma, esto lo hace si x es mayor a 1, si x no es mayor a 1 se termina la función, volviendo a la función principal(). Imprime desde num hasta 1.

Veamos otro ejemplo:

Elaborar un algoritmo que lea un número e imprima los números desde 1 hasta el número con incrementos de 1; utilizar una función recursiva.

```

Algoritmo RECURSIVIDAD2
1. Declarar
    Variables
        num: Entero
2. Función principal()
    a. Solicitar un numero
    b. Leer num
    c. imprimeNumero(1)
    d. Fin Función principal
3. Función imprimeNumero(Val x: Entero)
    a. Imprimir x
    b. if x < num then
        1. x = x + 1
        2. imprimeNumero(x)

```

```

    c. endif
    d. Fin Función imprimeNúmero
Fin

```



En la zona de descarga de la Web del libro, está disponible:  
Programa en C: C908.C

*Explicación:*

En la función principal se lee num, luego se llama a la función `imprimeNúmero`, `imprimeNúmero` es una función que recibe el valor 1 como parámetro en `x`; imprime el valor de `x`, le incrementa 1 a `x` y llama a la función `imprimeNúmero` con el nuevo valor de `x` como parámetro, es decir, se llama a sí misma, esto lo hace si `x` es menor que `num`, si `x` no es menor a `num` se termina la función, volviendo a la función principal(). Imprime desde 1 hasta `num`.

## 9.5 Ejecución de otros programas en código ejecutable

Mediante la función `Ejecutar` se puede ejecutar desde un programa, otros programas que estén en código ejecutable (.EXE o .COM). `Ejecutar` provoca una salida temporal hacia el sistema operativo y permite la ejecución de un comando del mismo.

*Formato:*

```
Ejecutar(Programa, Comando)
```

*En donde:*

|          |                                                                                                                         |
|----------|-------------------------------------------------------------------------------------------------------------------------|
| Programa | Es el nombre completo del archivo que contiene el comando o programa a ejecutar.                                        |
| Comando  | Es el nombre del comando que se ejecutará; debe colocarse como lo haría desde el indicador del sistema operativo (C:>). |

*Ejemplo 1:*

Ejecutar un comando interno del COMMAND.COM.

```
Ejecutar("C:\DOS\COMMAND.COM" , "DIR C:*.PAS")
```

*Explicación:*

C:\DOS\COMMAND.COM Es el nombre del archivo donde se encuentra el comando a ejecutar; en este caso es el archivo COMMAND.COM del directorio DOS del disco C, el cual contiene el comando DIR (que se ejecutará en este ejemplo).

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| DIR C:*.PAS | DIR es el comando (programa) a ejecutar.                                          |
| C           | es el disco donde se hará el DIR.                                                 |
| *.PAS       | es el parámetro que indica que se listarán todos los archivos con extensión .PAS. |

#### Ejemplo 2:

Suponiendo que hay el archivo C:\PROGRAMAS\PROG1.EXE el cual contiene un programa ejecutable, se puede ejecutar desde otro algoritmo (programa) así:

Ejecutar (C:\PROGRAMAS\PROG1.EXE, PROG1)

Se indica que se ejecuta PROG1 que está en C:\PROGRAMAS\PROG1.EXE.

## 9.6 Graficación en seudocódigo

Los lenguajes como Pascal o C permiten dibujar figuras geométricas, a esto se le llama graficación, al diseñar un programa, si usted tiene necesidad de graficar, podrá hacerlo con funciones como las siguientes (entre otras que usted prefiera):

### Arco

Dibuja un arco circular desde el anguloInicial hasta el anguloFinal, utilizando (x,y) como punto central y el radio. Sintaxis:

Arco (x, y, anguloInicial, anguloFinal, radio)

### Barra

Dibuja una barra rectangular desde x1,y1 hasta x2,y2, utilizando el relleno y color actual. Sintaxis:

Barra (x1, y1, x2, y2)

### Barra3D

Dibuja una barra en tres dimensiones de x1,y1 hasta x2,y2 con la profundidad prof y tapa indica si lleva o no tapa, utilizando el relleno y color actual. Sintaxis:

Barra3D (x1, y1, x2, y2, prof, tapa)

### Círculo

Dibuja un círculo utilizando (x,y) como punto central. Sintaxis:

Circulo (x, y, radio)

### CierraGraph

Quita de uso el sistema gráfico. Sintaxis:

CierraGraph

**DetectaGraph**

Verifica el hardware y determina el manejador y modo de graficación a utilizar.  
Sintaxis:

```
DetectaGraph(manejadorGraf, modoGraf)
```

**DibujaPoli**

Dibuja un polígono. Sintaxis:

```
DibujaPoli(numPuntos, puntos)
```

**Elipse**

Dibuja una elipse desde el anguloInicial hasta el anguloFinal, utilizando (x,y) como punto central. Sintaxis:

```
Elipse(x, y, anguloInicial, anguloFinal, xRadio, yRadio)
```

**IniciaGraph**

Inicia el sistema de gráficas y coloca el hardware en modo gráfico. Sintaxis:

```
IniciaGraph(manejador, modo, ruta)
```

**Línea**

Dibuja una línea desde (x1,y1) hasta (x2,y2). Sintaxis:

```
Línea(x1, y1, x2, y2)
```

**RebPastel**

Dibuja y rellena una rebanada de pastel desde el anguloInicial hasta el anguloFinal.

Sintaxis:

```
RebPastel(x, y, anguloInicial, anguloFinal, radio)
```

**Rectángulo**

Dibuja un rectángulo desde el punto (x1,y1) hasta (x2,y2). Sintaxis:

```
Rectangulo(x1, y1, x2, y2)
```

Entre otras funciones que usted puede añadir que traiga el manejador gráfico del lenguaje que esté utilizando, como RellenaElipse, RellenaPoli, etcétera.



## 9.7 Incluir archivos de programas

Cuando se diseñan programas grandes, a menudo el programador decide editar el programa utilizando más de un archivo, o bien, hay lenguajes (como C) que tienen una biblioteca de funciones que viene en varios archivos, en pseudocódigo podemos indicar que un algoritmo (programa) incluye a otros programas con el formato:

```
Incluir (archProg.Ext)
```

*En donde:*

|              |                                                                                                                                                                                                                |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Incluir      | Indica que en ese lugar se incluye un archivo.                                                                                                                                                                 |
| archProg.Ext | Es el nombre del archivo que se incluirá, el cual contiene un programa o conjunto de funciones que se han editado en ese archivo por separado del programa correspondiente al algoritmo que se está diseñando. |

*Ejemplo:*

```
Algoritmo EJEMPLO
1. Declarar
    Incluir archivos
        Incluir (funEntrada.Ext)
        Incluir (funSalida.Ext)
        Incluir (funGraficas.Ext)
    Tipos
    Variables
2. Función principal()
    --
    --
    n. Fin Función principal
3. Función otro()
    --
    --
    n. Fin Función otro
Fin
```

*Explicación:*

En la parte de declaraciones globales se coloca un apartado para incluir archivos, en el ejemplo, se incluyen los archivos funEntrada.Ext, funSalida.Ext y funGraficas; esta inclusión indicará al compilador que debe incluir esos archivos al momento de compilar el programa correspondiente al algoritmo diseñado.

## 9.8 Resumen de conceptos que debe dominar

- Clasificación (ordenación) de datos
- Tipos de datos definidos por el usuario (Tipos)
- Apuntadores (Pointer)
- La recursividad en sudocódigo
- Ejecutar programas en código ejecutable
- Graficación en seudocódigo
- Inclusión de archivos de programas

## 9.9 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

- 9.9.1 Resumen gráfico del capítulo
- 9.9.2 Autoevaluación
- 9.9.3 Programas
- 9.9.4 Power Point para el profesor (\*)

# 10

## Programación orientada a objetos usando el diagrama de clases

### Contenido

- 10.1 Objetos
  - 10.1.1 Qué son los objetos
  - 10.1.2 Cómo están formados los objetos
  - 10.1.3 Cuándo y cómo identificar los objetos
- 10.2 Clases y su relación con los objetos
  - 10.2.1 Determinar las clases
  - 10.2.2 Representación de la clase y sus instancias
- 10.3 Métodos y encapsulación
  - 10.3.1 Métodos
  - 10.3.2 Encapsulación
- 10.4 Diseño del diagrama de clases
  - 10.4.1 Modificadores de acceso (visibilidad)
- 10.5 Generar instancias de una clase
- 10.6 Arquitectura modelo-vista-controlador
- 10.7 Resumen de conceptos que debe dominar
- 10.8 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 10.8.1 Resumen gráfico del capítulo
  - 10.8.2 Autoevaluación
  - 10.8.3 Programas
  - 10.8.4 Power Point para el profesor (\*)

### Objetivos del capítulo

- Aprender los conceptos básicos de la programación orientada a objetos: objetos, clases, métodos y encapsulación.
- Aprender la aplicación de los conceptos básicos estudiados, en el diseño de programas o algoritmos orientados a objetos, a través del uso del diagrama de clases.
- Estudiar la arquitectura modelo-vista-controlador.

## Introducción

Con el estudio de los capítulos anteriores, usted ya domina la metodología de la programación estructurada; y cómo diseñar algoritmos usando esos conceptos y estructuras, donde destaca el uso del diseño descendente (Top Down Design), para realizar el diseño arquitectónico del algoritmo o programa, dividiendo la solución en un conjunto de funciones jerarquizadas.

El propósito de los siguientes capítulos es plantear una evolución de la metodología de la programación estructurada hacia la metodología de la programación orientada a objetos, destacando que el cambio sustancial radica en el diseño arquitectónico del algoritmo, que ahora, en lugar del diseño descendente (Top Down Design), se utiliza el diagrama de clases (adaptación hecha de UML: Unified Modeling Language, desarrollado por G. Booch, I. Jacobson y J. Rumbaugh), y la arquitectura modelo-vista-controlador. Sin embargo, aunque el diseño descendente (Top down design), no se usa en la programación orientada a objetos, los conceptos de funciones, parámetros y todo lo estudiado en los capítulos anteriores, son la base de la programación orientada a objetos; de aquí en adelante, se estudian los conceptos de objetos, integrándolos con los conceptos aprendidos en los capítulos anteriores.

En este capítulo se exponen los conceptos básicos de la programación orientada a objetos: objetos, clases, métodos y encapsulación; y su aplicación en el diseño de programas o algoritmos orientados a objetos, a través del uso del diagrama de clases y la arquitectura modelo-vista-controlador.

Se expone que un programa orientado a objetos está formado por un conjunto de objetos, donde cada objeto está formado por datos y un conjunto de métodos (denominados funciones o módulos en la programación estructurada).

Se explica qué son los objetos, cómo están formados, cuándo y cómo identificarlos, qué son las clases y su relación con los objetos, cómo generar instancias de una clase y cómo diseñar el diagrama de clases.

Cabe aclarar que en este capítulo no se presentan ejercicios, porque este tema se aplica en todos los ejercicios de los capítulos restantes.

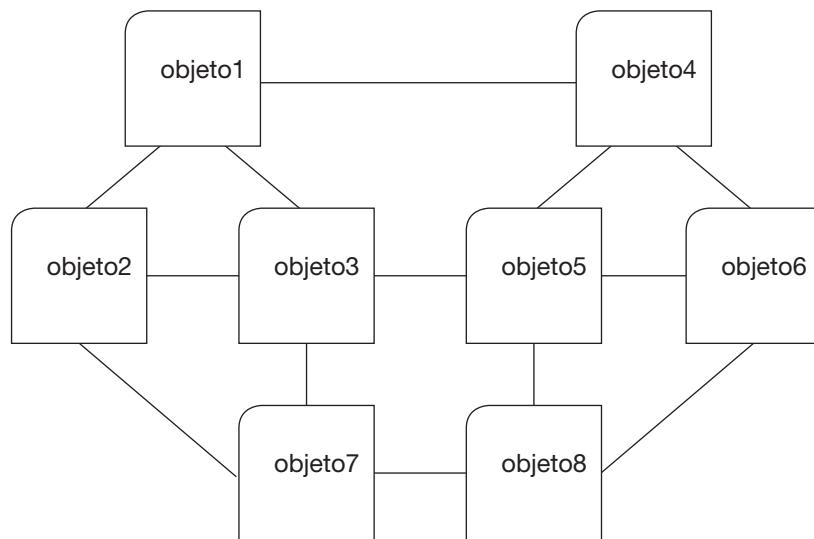
En el siguiente capítulo se estudia la programación orientada a objetos aplicando la estructura de secuenciación.

## 10.1 Objetos

La programación orientada a objetos (POO, también identificada como OOP: Object-Oriented Programming), es un método de programación muy parecido a la forma en que nosotros hacemos cosas. Es una evolución natural de las innovaciones más recientes en el diseño de los lenguajes de programación. La POO es más estructurada que las tentativas previas de la programación estructurada, y es más modular y abstracta que los intentos previos en abstracción de datos y ocultamiento de detalles. La programación orientada a objetos está caracterizada por las siguientes propiedades: Objetos, Clases, Encapsulación, Herencia y Polimorfismo.

Un programa orientado a objetos está formado por una colección de objetos interaccionando conjuntamente para representar y solucionar algún problema.

Gráficamente, podríamos observar un programa orientado a objetos, como se muestra en la siguiente figura.



*Explicación:*

El programa está constituido por un conjunto de objetos relacionados entre sí, los cuales permiten hacer la entrada de datos, los cálculos necesarios para resolver el problema y dar la salida de datos convertidos en información.

### 10.1.1 Qué son los objetos

Los objetos son entes, entidades, sujetos o cosas que encontramos en el dominio del problema de nuestra realidad. Entiéndase, en situaciones o problemas de nuestro mundo cotidiano empresarial, organizacional o institucional, que se requiere manejar o solucionar mediante la computadora.



El primer lenguaje orientado a objetos fue Java.  
En la Web del libro encontrará un video que explica como instalar Java y Eclipse que es una herramienta para facilitar el uso de este lenguaje.

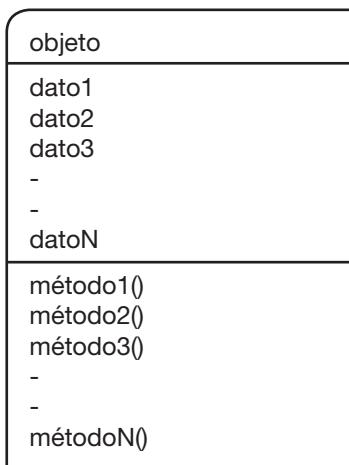
### 10.1.2 Cómo están formados los objetos

Los objetos están formados por dos elementos:

**Datos** que representan su estructura. En otras palabras, atributos que describen su forma o apariencia.

**Métodos** que implementan las funciones propias de su comportamiento, es decir, métodos que manipulan los datos para hacer la entrada, proceso y salida de los mismos. En el contexto no orientado a objetos, a los métodos se les conoce como procedimientos, módulos, funciones, subrutinas, etcétera.

Esquemáticamente, podríamos observar un objeto como se muestra en la siguiente figura.



*Explicación:*

El objeto se representa con un rectángulo redondeado y está formado por dos partes: Una parte, son los datos que representan la estructura del objeto; y la otra parte, son los métodos que implementan las operaciones que se deben realizar sobre los datos.

### 10.1.3 Cuándo y cómo identificar los objetos

En el entorno del proceso de programación, en el paso de análisis del problema, el elemento central es el objeto, es decir, se estudia el problema y se identifican los objetos involucrados y sus relaciones. A continuación vamos a plantear un problema para exemplificar cómo identificar los objetos.

*Definición del problema (es el paso 1 del proceso de programación):*

Elaborar un programa que permita calcular el sueldo de los empleados de una empresa.

*Análisis del problema (es el paso 2 del proceso de programación):*

### 1. Planteamiento del problema

En la empresa X, se tienen varios empleados; por cada empleado, se tienen los datos: Nombre, Número de horas trabajadas y la Cuota que se le paga por hora trabajada. Cada quincena se requiere emitir un cheque con los datos: Nombre del empleado y sueldo que se le debe pagar. El sueldo se calcula multiplicando horas trabajadas por cuota por hora. Resumiendo, desde la perspectiva de entrada-proceso-salida:

Información que debe imprimir como salida: Nombre y Sueldo.

Datos que se deben leer: Nombre, Horas trabajadas y Cuota por hora.

Proceso a seguir: Calcular Sueldo=Horas trabajadas x Cuota por hora.

### 2. Identificar objetos

Para identificar los objetos presentes en el dominio del problema, se debe estudiar el caso expuesto, con el propósito de distinguir y determinar el objeto o colección de objetos, que el programa debe representar y manejar, para solucionar la situación planteada.

¿Cómo se identifican los objetos? Se buscan los sustantivos presentes en la especificación; los sustantivos son los objetos. En nuestro problema se menciona un sustantivo: los empleados.

Aplicando esto, en el problema formulado anteriormente, tenemos que nuestro programa debe representar una colección de empleados como la que se muestra en la siguiente figura.

| objetoEmpleado          | objetoEmpleado           |     | objetoEmpleado          |
|-------------------------|--------------------------|-----|-------------------------|
| Nombre<br>Juan          | Nombre<br>Pedro          | --- | Nombre<br>Mateo         |
| Horas trabajadas<br>40  | Horas trabajadas<br>45   | --- | Horas trabajadas<br>55  |
| Cuota por hora<br>50.00 | Cuota por hora<br>100.00 | --- | Cuota por hora<br>80.00 |
| Sueldo<br>2000.00       | Sueldo<br>4500.00        | --- | Sueldo<br>4400.00       |



Observe que se está utilizando un rectángulo redondeado para representar a cada objeto.



En este momento se está esquematizando cada objeto desde el punto de vista de su estructura, es decir, de los datos que lo representan; en puntos subsiguientes se explicará cómo involucrar a los métodos.

*Explicación:*

Se muestra la colección de empleados que debe representar el programa, es decir, a un objetoEmpleado que es el empleado Juan, a otro objetoEmpleado que es el empleado Pedro, y a otro empleado, y a otro, hasta el último objetoEmpleado presente, que es el empleado Mateo.

Así, tenemos una colección de objetos iguales, es decir, se tiene un conjunto de empleados, donde cada empleado es un objeto que tiene la misma estructura de datos, y tiene el mismo comportamiento que todos los demás objetos (empleados).

## 10.2 Clases y su relación con los objetos

La clase es una representación abstracta que describe a un conjunto de objetos; en otras palabras, es un tipo de dato abstracto que representa a un conjunto de objetos que tienen en común una misma estructura (de datos) y un mismo comportamiento (las mismas funciones), es decir, son la misma cosa y hacen lo mismo.

En la fase de diseño del programa, el elemento central es la clase, porque un conjunto de objetos iguales, debe ser representado en forma abstracta por una clase, que fungirá como una plantilla o molde, para crear todos y cada uno de los objetos necesarios.

En el entorno del proceso de programación sigue el paso:

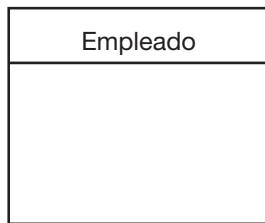
### 3. Diseño del programa

Para diseñar el programa, se elabora o diseña el algoritmo que soluciona el problema, en dos pasos: Primero, se diseña el diagrama de clases; segundo, se diseña la lógica de las clases en seudocódigo; después se hace la prueba de escritorio. En este capítulo se explica cómo diseñar el diagrama de clases.

#### 10.2.1 Determinar las clases

Una vez identificados los objetos del problema, se procede a diseñar el diagrama de clases, para lo cual, debemos primero determinar la(s) clase(s) que necesitaremos, y luego dibujar el diagrama con la representación de la(s) clase(s) y sus instancias.

Los objetos identificados en el punto anterior se convierten en clases, que como ya se mencionó, son la representación abstracta de un conjunto de objetos; aplicando este concepto al problema que nos ocupa, el cual tiene un conjunto de objetos de empleados, tenemos que dicha colección se representa como clase, mediante el símbolo que se indica en la siguiente figura.

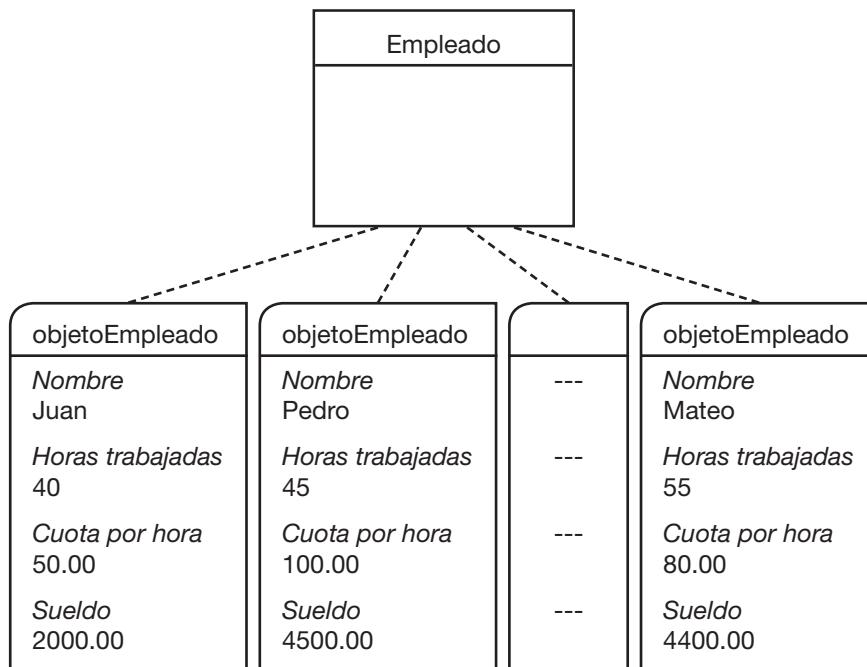


*Explicación:*

Una clase se representa con un rectángulo. En nuestro ejemplo, estamos representando la clase Empleado.

### 10.2.2 Representación de la clase y sus instancias

En el programa que elaboraremos, debemos definir una clase como un nuevo tipo de datos abstracto, y con base en la referida clase, vamos a generar instancias u ocurrencias específicas de dicha clase, cada una de estas instancias son los objetos que representarán a cada uno de los empleados presentes en nuestra aplicación; la relación entre una clase y sus instancias se representa como se muestra en la siguiente figura.



*Explicación:*

Se tiene la clase Empleado, de la cual se genera una instancia, que es el objeto que representa al empleado Juan, asimismo se genera otra instancia que es el objeto que representa al empleado Pedro, y así sucesivamente, hasta generar la instancia que es el objeto que representa al empleado Mateo.

**Nota:** La línea indica que hay una relación “instancia de” u “objeto de”; por ejemplo, que Juan es una instancia, objeto u ocurrencia de la clase Empleado, lo mismo que Pedro, Mateo, etcétera.

## 10.3 Métodos y encapsulación

---

### 10.3.1 Métodos

Como ya lo hemos mencionado, un objeto se representa por medio de datos; datos que deben entrar como materia prima y datos que deben calcularse a partir de otros datos. Los métodos son las acciones que implementan el comportamiento o las funciones de un objeto; por ejemplo, leer o dar entrada a los datos, hacer cálculos, imprimir o dar salida de datos.

### 10.3.2 Encapsulación

Como ya se explicó anteriormente, un conjunto de objetos se representa en forma abstracta por una clase, la cual es un tipo de dato, que está compuesto por dos elementos: datos y métodos. La encapsulación significa colocar juntos los datos y los métodos dentro de un objeto. Uno de los principios más importantes de la programación orientada a objetos, es que el programador debe pensar en el código y los datos juntos durante el diseño del programa. Ninguno, ni el código ni los datos existen en un vacío. Los datos dirigen el flujo del código y el código manipula la forma y valores de los datos.

Cuando el código y los datos son entidades separadas, siempre existe el peligro de llamar el método correcto con los datos erróneos o viceversa. Juntar los dos es trabajo del programador. El objeto mantiene en sincronía los datos y los métodos empacados juntos en el mismo. Al diseñar la estructura de una clase, se deben hacer dos cosas:

1. Definir los datos que representarán a los objetos de la clase. Los datos que representan a un objeto son de dos tipos: Un tipo son los datos que entrarán como materia prima, es decir, que se deberán leer a través de dispositivos de entrada. El otro tipo son datos que se deben generar mediante cálculos.
2. Definir los métodos que implementarán las acciones de los objetos. Los métodos son de dos tipos: Un tipo son los métodos que colocan o establecen (setters) valores a los datos del objeto, ya sea a través de leerlos o mediante cálculos. El otro tipo son los métodos que acceden u obtienen (getters) los valores de los datos para utilizarlos, ya sea para hacer cálculos o para darles salida.

¿Cómo identificar los métodos? Se buscan los verbos presentes en el planteamiento del problema, en nuestro caso se debe:

Imprimir como salida: Nombre y Sueldo.

Leer los datos: Nombre, Horas trabajadas y Cuota por hora.

Calcular el sueldo: Sueldo=Horas trabajadas x Cuota por hora.

Típicamente, para establecer los valores para cada uno de los datos de un objeto, se requiere definir y diseñar un método perteneciente al objeto que permita colocar (setter) el valor a cada dato; ya sea que lo lea o que lo calcule. Asimismo, para dar salida al valor de un dato, se requiere definir y diseñar un método que permita acceder (getter) al valor del dato para darlo como salida, esto es, un método para obtener cada dato.

Como muchos aspectos de la programación orientada a objetos, la encapsulación de datos es una disciplina que se debe respetar siempre. Para establecer u obtener los datos de un objeto, se debe hacer mediante métodos del mismo objeto, y no leerlos o accederlos directamente.

Ejemplo:

En el problema de calcular el sueldo de varios empleados, tenemos que cada objeto de empleado se representa mediante los datos:

|           |                            |
|-----------|----------------------------|
| nombreEmp | Nombre del empleado        |
| horasTrab | Número de horas trabajadas |
| cuotaHora | Cuota por hora             |
| sUELDO    | Sueldo del empleado        |

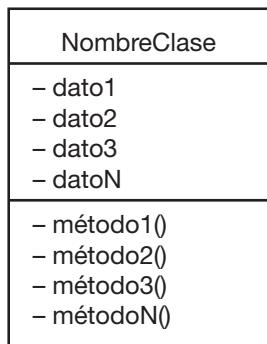
Para cada dato, se debe definir un método para establecer o colocar (setter) el valor que tomará; ya sea que lo tomará a través de leer el dato, haciendo cálculos, u otra forma, y otro método que permita obtener o acceder (getter) al valor para utilizarlo, ya sea para hacer cálculos, para imprimirla o hacer otra cosa con él. En nuestro problema, se requieren los siguientes métodos:

establecerNombreEmp(). Para establecer el valor del dato nombreEmp.  
establecerHorasTrab(). Para establecer el valor del dato horasTrab.  
establecerCuotaHora(). Para establecer el valor del dato cuotaHora.  
calcularSueldo(). Para calcular y establecer el valor del dato sueldo.  
obtenerNombreEmp(). Para acceder e imprimir el valor del dato nombreEmp.  
obtenerSueldo(). Para acceder e imprimir el valor del dato sueldo.

**Nota:** Observe que los métodos setter inician su nombre con establecer o calcular, y los métodos getter inician con obtener. Asimismo, observe que no para todos los datos se define un método getter; en este caso sólo se definieron para los datos que se van a imprimir como salida: nombreEmp y sueldo.

## 10.4 Diseño del diagrama de clases

Una vez que se tienen definidos los datos y los métodos necesarios, se procede a diseñar la estructura de la clase, que al mismo tiempo se convierte en el diagrama de clases; se hace con el siguiente formato:



*En donde:*

- NombreClase Indica el nombre de la clase.
- dato1, dato2, dato3, datoN Son los datos que representarán a cada uno de los objetos de la clase.
- método1(), método2() Son los métodos que establecerán y obtendrán
- método3(), métodoN() los valores de los datos de cada uno de los objetos de esta clase.
- ( ) Indica que es un método.
- Modificador de acceso (visibilidad).

#### 10.4.1 Modificadores de acceso (visibilidad)

Con los símbolos `-`, `+`, `#`, `_` y `*` se indica la visibilidad que tendrá cada dato o método, a éstos se les conoce como modificadores de acceso: Privado (`-`), Protegido (`#`), Público (`+`), Estático (`_`) y Abstracto (`*`). Esto quiere decir desde qué partes son visibles para ser utilizados.

##### Visibilidad de los datos

- **Privado (Private)**  
Los datos que se declaran como privados, sólo pueden ser vistos y utilizados por métodos de la misma clase. Por defecto (default) los datos son privados.
- # **Protegido (Protected)**  
Los datos que se declaran como protegidos, pueden ser vistos y utilizados por métodos de la misma clase y por métodos de subclases derivadas de la clase donde están declarados.
- + **Público (Public)**  
Los datos que se declaran como públicos, pueden ser vistos y utilizados tanto por métodos de la misma clase, como por métodos de otras clases.
- **Estático (Static)**  
Los datos que se declaran como estáticos, son únicos para toda la clase, es decir, no pertenecen a ninguna instancia (objeto) de la clase, pero pueden ser vistos y utilizados por todas las instancias de la clase.

## Visibilidad de los métodos

### - Privado (Private)

Los métodos que se declaran como privados, sólo pueden ser vistos y utilizados por métodos de la misma clase.

### # Protegido (Protected)

Los métodos que se declaran como protegidos, pueden ser vistos y utilizados por métodos de la misma clase y por métodos de subclases derivadas de la clase donde están declarados.

### + Público (Public)

Los métodos que se declaran como públicos, pueden ser vistos y utilizados tanto por métodos de la misma clase, como por métodos de otras clases.

Por defecto (default) los métodos son públicos.

### \_ Estático (Static)

Los métodos que se declaran como estáticos, son únicos para toda la clase, es decir, no pertenecen a ninguna instancia (objeto) de la clase, pero pueden ser vistos y utilizados por métodos de todas las instancias de la clase.

### \* Abstracto (Abstract)

Los métodos que se declaran como abstractos, no tienen implementación; por tanto, deben ser implementados en subclases.

Ejemplo:

| ClaseEjemplo                                                            |
|-------------------------------------------------------------------------|
| - dato1<br>+ dato2<br># dato3<br>_ dato4                                |
| - método1()<br>+ método2()<br># método3()<br>_ método4()<br>* método5() |

*En donde:*

Se tienen los datos:

- dato1      Privado (Private)
- + dato2      Público (Public)
- # dato3      Protegido (Protected)
- \_ dato4      Estático (Static)



Por defecto (default) los datos son privados, es decir, que si no se les define su visibilidad mediante alguno de los modificadores de acceso `-`, `+`, `#` y `_` entonces los datos son privados.

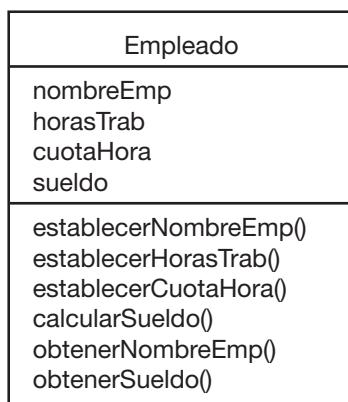
#### Los métodos:

- `- método1` Privado (Private)
- `+ método2` Público (Public)
- `# método3` Protegido (Protected)
- `_ método4` Estático (Static)
- `* método5` Abstracto (Abstract)

Por defecto (default) los métodos son públicos, es decir, que si no se les define su visibilidad mediante alguno de los modificadores de acceso `-`, `+`, `#`, `_` y `*` entonces los métodos son públicos.

Debido a la profundidad con que se está tratando la metodología que se está presentando en este libro, estaremos utilizando sólo datos privados y métodos públicos, es por ello que no se estará indicando el modificador de acceso.

Aplicando el formato en nuestro ejemplo, nos queda el siguiente diagrama de clases:



#### Explicación:

Se tiene el diagrama de clases, el cual consta de una clase.

Nombre de la clase: Empleado

Datos que tiene la clase:

|           |                            |
|-----------|----------------------------|
| nombreEmp | Nombre del empleado        |
| horasTrab | Número de horas trabajadas |
| cuotaHora | Cuota por hora             |
| sueldo    | Sueldo del empleado        |

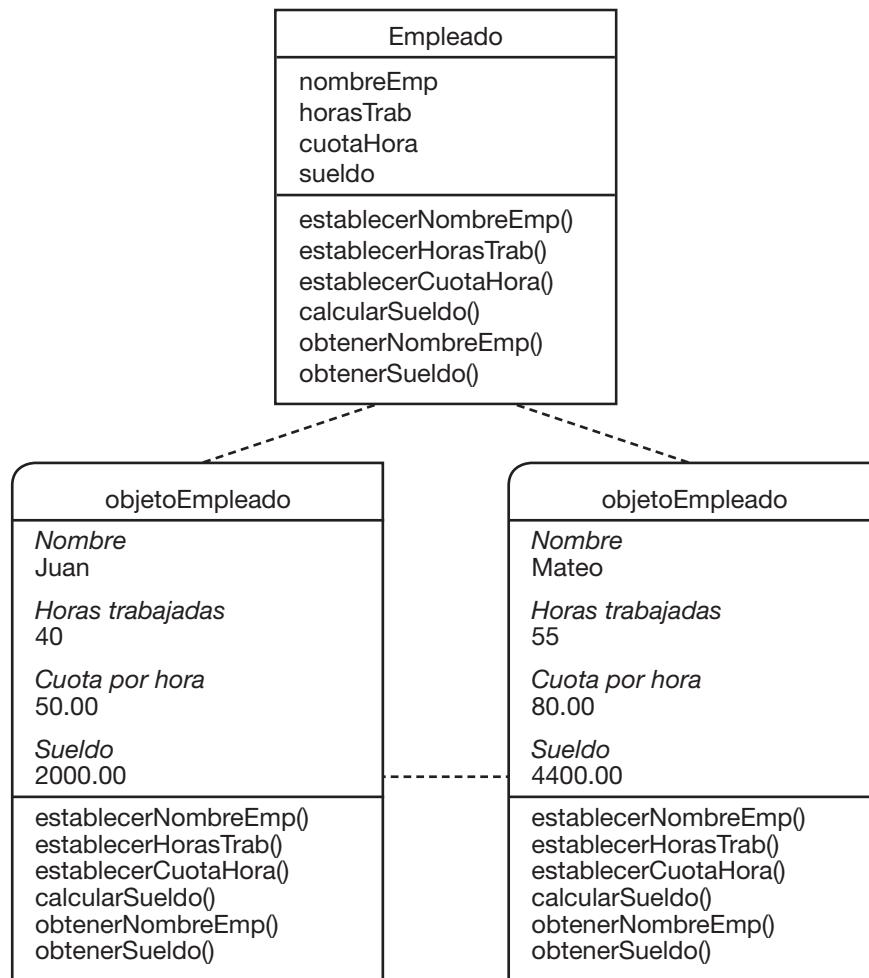
Métodos que tiene la clase:

`establecerNombreEmp()`. Para establecer el valor del dato `nombreEmp`.  
`establecerHorasTrab()`. Para establecer el valor del dato `horasTrab`.

establecerCuotaHora(). Para establecer el valor del dato cuotaHora.  
calcularSueldo(). Para calcular y establecer el valor del dato sueldo.  
obtenerNombreEmp(). Para acceder e imprimir el valor del dato nombreEmp.  
obtenerSueldo(). Para acceder e imprimir el valor del dato sueldo.

## 10.5 Generar instancias de una clase

Una vez que tenemos definida una clase como Empleado, ésta se utiliza para generar instancias u ocurrencias; estas instancias son los objetos que darán vida al funcionamiento de nuestro programa. Esto quiere decir que la clase es como un molde, una plantilla o tipo de dato, que se utiliza para generar objetos específicos, en la siguiente figura se esquematiza.



**Explicación:**

Se tiene la clase Empleado, de la cual se genera una instancia, que es el objetoEmpleado que representa al empleado Juan, y así sucesivamente, hasta generar la instancia que es el objetoEmpleado que representa al empleado Mateo. Observe que cada objetoEmpleado que se crea de la clase Empleado tendrá los datos:

```
nombreEmp
horasTrab
cuotaHora
sueldo
```

También tendrá los métodos:

```
establecerNombreEmp()
establecerHorasTrab()
establecerCuotaHora()
calcularSueldo()
obtenerNombreEmp()
obtenerSueldo()
```

 La línea indica que hay una relación “instancia de” u “objeto de”; por ejemplo, que Juan es una instancia, objeto o ocurrencia de la clase Empleado, lo mismo Mateo, etcétera.

## 10.6 Arquitectura modelo-vista-controlador

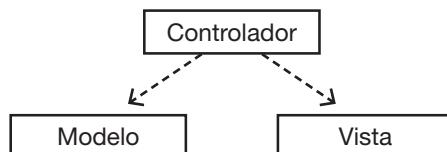
El diagrama de clases que se ha diseñado es lo que constituye el modelo de nuestro programa, sin embargo, este modelo necesita utilizarse en la solución del problema, y para hacerlo, se requiere aplicar la arquitectura modelo-vista-controlador, la cual establece que un programa orientado a objetos está formado por tres partes:

El modelo. Es la clase o conjunto de clases identificadas en el dominio del problema, mismas que representan al objeto u objetos presentes en el problema.

La vista. Es la interfaz de usuario, es decir, lo que el usuario observará en la pantalla, impresora u otro dispositivo de salida de la computadora al operar el programa.

El controlador. Es la parte que permite que el usuario interactúe con la vista para utilizar el modelo que representa y soluciona el problema.

En programas complejos la arquitectura modelo-vista-controlador se ve así:



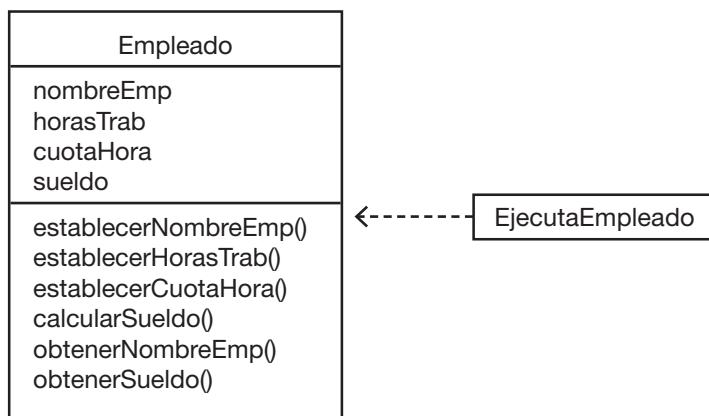
Donde la parte controlador es la que lleva el control del funcionamiento del programa, auxiliándose de la parte vista que seguramente es una sofisticada interfaz gráfica de usuario, y de la parte modelo que es la parte que representa y soluciona el problema.

Sin embargo, en programas no muy complejos o que la vista es sencilla, la parte controlador incluye a la parte vista, y la arquitectura del programa se simplifica a dos partes, la cual se ve así:



Esta arquitectura es la que se utiliza en los problemas que se plantean en este libro, debido a que está dirigido a un nivel donde todavía no se enfrentan problemas muy complejos.

Aplicando esta arquitectura al problema que hemos venido diseñando, el diagrama de clases nos queda de la forma siguiente:



#### Explicación:

Este diagrama consta de dos clases: Una es Empleado, que es el modelo que representa y soluciona el problema planteado. La otra es EjecutaEmpleado, que es la clase controlador, que utiliza el modelo que es la clase Empleado, para controlar la interacción con el usuario y permite que el programa sea operado por el usuario para resolver su problema.

#### ¿Qué hemos hecho hasta ahora?

En este capítulo se han explicado los conceptos básicos de la programación orientada a objetos, y se ha planteado la forma de cómo involucrarlos para diseñar el diagrama de clases, el cual contiene la estructura general del programa (algoritmo) que estamos diseñando, esto es, las clases necesarias para resolver el problema.

#### Diferencia entre la programación estructurada y la orientada a objetos

La mayoría de los conceptos y estructuras de la programación estructurada presentados en los capítulos anteriores, son también parte de la programación orientada a objetos. La diferencia esencial de la programación orientada a objetos con respecto a la programación estructurada, son los conceptos y el enfoque de diseño del diagrama de clases, en lugar del enfoque de diseño descendente (Top



La flecha punteada representa una relación de dependencia en el sentido de que la clase Controlador utiliza a la clase Modelo y a la clase Vista.



A diferencia de todos los demás capítulos de este libro, en este no se presentan problemas resueltos ni problemas por resolver, debido a que la aplicación del diseño de diagramas de clases se hará conjuntamente con los temas que siguen, para así complementar la metodología de diseño de algoritmos orientados a objetos.



Los problemas que se plantean en los capítulos once, doce y trece; serán solucionados con un diagrama de clases similar al presentado en este punto. Sin embargo, es pertinente aclarar que al profundizar en temas orientados a objetos como herencia y polimorfismo, es posible que el diagrama de clases contenga una mayor cantidad de clases.

Down Design). Lo que es totalmente diferente en la POO, es en la arquitectura general de los programas, los cuales ya no están formados por un conjunto de módulos o funciones, como sucedía con la programación estructurada, sino por un conjunto de objetos, generados a partir de las clases del diagrama de clases, donde los objetos contienen un conjunto de métodos (módulos o funciones) y éstos están formados por instrucciones.

Este autor piensa que en el proceso de formación de un estudiante de programación, el que éste aprenda primero la programación estructurada, no debe afectarle, por el contrario, le da más elementos para poder entender y comprender mejor la programación orientada a objetos. Lo que sí es importante es que esté consciente de esta diferencia; y en su momento sepa si está realizando programación estructurada, o bien, si está haciendo programación orientada a objetos.

### ¿Qué sigue?

Para diseñar programas (algoritmos) aplicando la metodología de la programación orientada a objetos que se está presentando, se deben diseñar dos elementos: Uno es el diagrama de clases; el otro elemento es diseñar la lógica de cada una de las clases que integran el diagrama de clases, utilizando la técnica seudocódigo, para así tener completa la metodología de diseño de programas (algoritmos) orientados a objetos. En el siguiente capítulo se explica cómo hacerlo.

## 10.7 Resumen de conceptos que debe dominar

- Objetos
- Clases
- Métodos
- Encapsulación
- Diagrama de clases
- Modificadores de acceso
- Arquitectura modelo-vista-controlador

## 10.8 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

### 10.8.1 Resumen gráfico del capítulo

### 10.8.2 Autoevaluación

### 10.8.3 Programas

### 10.8.4 Power Point para el profesor (\*)

# 11

## Programación orientada a objetos aplicando la estructura de secuenciación

### Contenido

- 11.1 Nuestro primer problema
- 11.2 Diseño de algoritmos OO usando la secuenciación en seudocódigo
- 11.3 Constructores y destructores
- 11.4 Ejercicios resueltos
- 11.5 Ejercicios propuestos
- 11.6 Resumen de conceptos que debe dominar
- 11.7 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 11.7.1 Resumen gráfico del capítulo
  - 11.7.2 Autoevaluación
  - 11.7.3 Programas
  - 11.7.4 Ejercicios resueltos
  - 11.7.5 Power Point para el profesor (\*)

### Objetivos del capítulo

- Aprender cómo diseñar el algoritmo usando la técnica seudocódigo, aplicando la estructura de control de secuenciación en el diseño de algoritmos orientados a objetos de manera de integrar los conceptos de la programación estructurada, estudiados en los primeros 3 capítulos, y la programación orientada a objetos.

## Introducción

Con el estudio del capítulo anterior, usted ya domina los conceptos básicos de la programación orientada a objetos y sabe cómo diseñar el diagrama de clases.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos mediante la programación orientada a objetos aplicando la estructura de secuenciación.

Se explica cómo diseñar el algoritmo usando la técnica seudocódigo, tomando como base el diagrama de clases, diseñado con losconceptos aprendidos en el capítulo anterior, aplicando la estructura de control: secuenciación, en el diseño de algoritmos orientados a objetos. Esto es cómo diseñar la lógica de cada una de las clases definidas en el diagrama de clases, integrando los conceptos de la programación estructurada, estudiados en los primeros siete capítulos, con los conceptos de objetos, clases, métodos y encapsulación de la programación orientada a objetos, en la técnica seudocódigo, lo que significa, una evolución natural de la programación estructurada a la orientada a objetos.

Se expone que los lenguajes orientados a objetos como Java proporcionan por defecto (default) un método constructor y un proceso destructor para cada clase definida.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercte estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que, a veces, aunque haya diferencias, las dos soluciones son correctas.

En el siguiente capítulo se estudia la programación orientada a objetos aplicando las estructuras de selección.

## 11.1 Nuestro primer problema

Para diseñar un programa o algoritmo orientado a objetos se hacen dos cosas: Primero se diseña el diagrama de clases siguiendo los lineamientos explicados en el capítulo anterior; y segundo, se diseña la lógica de cada una de las clases usando la técnica seudocódigo. En este punto aplicaremos la estructura de control de secuenciación en seudocódigo, que estudiamos en el capítulo 3, pero aplicada conjuntamente con el diagrama de clases en el entorno de la programación orientada a objetos.

A continuación se presenta un ejemplo para aplicar los conceptos antes descritos, y además para explicar la forma de cómo se diseña un algoritmo orientado a objetos.

Ejemplo:

Elaborar un algoritmo para calcular e imprimir el sueldo de un empleado.

Siguiendo el proceso de programación se hace lo siguiente:

1. *Definir el problema.*

Elaborar un algoritmo que permita calcular el sueldo de un empleado.

2. *Analizar el problema.*

Se tienen los datos del empleado: Nombre, Número de horas trabajadas y la Cuota que se le paga por hora trabajada. Se requiere imprimir los datos: Nombre del empleado y sueldo que se le debe pagar. El sueldo se calcula multiplicando horas trabajadas por cuota por hora.

Resumiendo, desde la perspectiva de entrada-proceso-salida:

Información que debe imprimir como salida: Nombre y Sueldo.

Datos que debe leer: Nombre, Número de horas trabajadas y Cuota por hora.

Calcular sueldo:  $\text{Sueldo} = \text{Horas trabajadas} \times \text{Cuota por hora}$ .

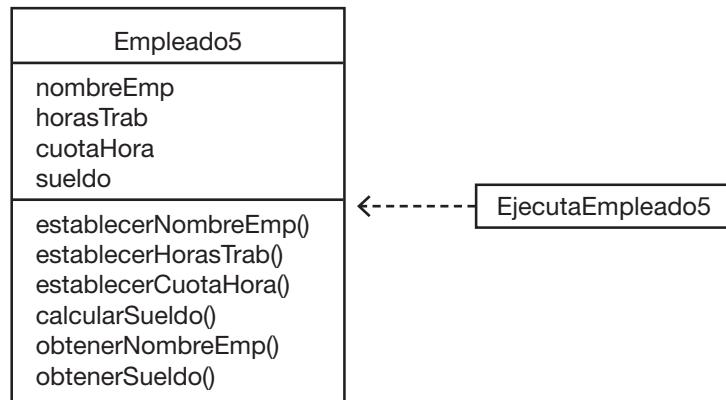
3. *Diseñar el programa.*

Para diseñar el programa, se elabora el algoritmo que soluciona el problema, en dos pasos:

**Primero**

Se diseña el diagrama de clases, aplicando los lineamientos que se explicaron en el capítulo anterior. Nos queda el siguiente diagrama:

Diagrama de clases

**Explicación:**

Este diagrama consta de dos clases: Una es la clase Empleado5, que es el modelo que representa y soluciona el problema planteado, la cual está formada por:

**Los datos:**

|           |                            |
|-----------|----------------------------|
| nombreEmp | Nombre del empleado        |
| horasTrab | Número de horas trabajadas |
| cuotaHora | Cuota por hora             |
| sueldo    | Sueldo del empleado        |

**Los métodos:**

establecerNombreEmp(). Para establecer el valor del dato nombreEmp.  
 establecerHorasTrab(). Para establecer el valor del dato horasTrab.  
 establecerCuotaHora(). Para establecer el valor del dato cuotaHora.  
 calcularSueldo(). Para calcular y establecer el valor del dato sueldo.  
 obtenerNombreEmp(). Para acceder e imprimir el valor del dato nombreEmp.  
 obtenerSueldo(). Para acceder e imprimir el valor del dato sueldo.

La otra es la clase EjecutaEmpleado5, que es la clase controlador, la cual utiliza el modelo, que es la clase Empleado5, para controlar la interacción con el usuario representando y resolviendo su problema.

**Segundo**

Se diseña la lógica de cada una de las clases usando seudocódigo, para armar el algoritmo que resuelve el problema. A continuación se explica cómo elaborar el algoritmo de la solución.

Aquí se utiliza el nombre de clase desde Empleado5, porque Empleado1, Empleado2, Empleado3 y Empleado 4, se usaron en algunas de las clases de programas en Java de capítulos anteriores. Lo mismo sucederá con Alumno, Angulo, Obrero, etcétera.

## 11.2 Diseño de algoritmos OO usando la secuenciación en seudocódigo

En este punto se explica cómo diseñar el algoritmo orientado a objetos usando seudocódigo. Cuando se tiene más de una clase, el formato del algoritmo es el siguiente:

```
Algoritmo ALGO
    Clase NomClase1
        1. Declarar
            Datos
            Objetos
            Variables
        2. Método funcionUno()
            a. Acción a
            b. Acción b
            c. Fin Método funcionUno
        3. Método funcionN()
            a. Acción a
            b. Acción b
            c. Fin Método funcionN
    Fin Clase NomClase1
    Clase NomClase2
        1. Declarar
        2. Método funcionUno()
            a. Acción a
            b. Fin Método funcionUno
        3. Método funcionN()
            a. Acción a
            b. Fin Método funcionN
    Fin Clase NomClase2
    Clase NomClaseN
        1. Declarar
        2. Método principal()
            a. Acción a
            b. Fin Método principal
        3. Método funcionUno()
            a. Acción a
            b. Fin Método funcionUno
        4. Método funcionN()
            a. Acción a
            b. Fin Método funcionN
    Fin Clase NomClaseN
Fin
```

**Explicación:**

Se tiene el esquema de un algoritmo que tiene tres clases, puede haber más clases, cada clase tiene su parte de declaraciones, donde se declaran los datos, y enseguida, los métodos que necesite; aclarando que en un algoritmo que tiene varias clases, sólo en una clase se puede tener Método principal, ya que es ahí donde el programa inicia su funcionamiento, y en el momento en que lo requiera utilizará las otras clases y métodos.

No obstante que un algoritmo puede estar formado por varias o muchas clases, y debido al nivel que está dirigido este libro, estaremos utilizando dos clases, esto es, la clase modelo y la clase controlador, donde la clase controlador incluye a la parte vista.

**Aplicando este formato en nuestro ejemplo**

Tomando como base el diagrama de clases diseñado en el punto anterior, donde se tienen dos clases, la clase Empleado5 y la clase EjecutaEmpleado5. Ahora se procederá a diseñar la lógica de cada una de las clases usando pseudocódigo, quedando el siguiente algoritmo:

```
Algoritmo CALCULAR SUELDO DE UN EMPLEADO
    Clase Empleado5
        1. Declarar
            Datos
                nombreEmp: Cadena
                horasTrab: Entero
                cuotaHora: Real
                sueldo: Real
        2. Método establecerNombreEmp (nom: Cadena)
            a. nombreEmp = nom
            b. Fin Método establecerNombreEmp
        3. Método establecerHorasTrab (horasTrab: Entero)
            a. horasTrab = horasTrab
            b. Fin Método establecerHorasTrab
        4. Método establecerCuotaHora (cuotaHr: Real)
            a. cuotaHora = cuotaHr
            b. Fin Método establecerCuotaHora
        5. Método calcularSueldo()
            a. sueldo = horasTrab * cuotaHora
            b. Fin Método calcularSueldo
        6. Método obtenerNombreEmp () Cadena
            a. return nombreEmp
            b. Fin Método obtenerNombreEmp
        7. Método obtenerSueldo () Real
            a. return sueldo
            b. Fin Método obtenerSueldo
    Fin Clase Empleado5
```

```

Clase EjecutaEmpleado5
1. Método principal()
    a. Declarar
        Variables
            nomEmp: Cadena
            hrsTra: Entero
            cuoHr: Real
    b. Declarar, crear e iniciar objeto
        Empleado5 objEmpleado = new Empleado5()
    c. Solicitar Nombre, número de horas trabajadas
        y cuota por hora
    d. Leer nomEmp, hrsTra, cuoHr
    e. Establecer
        objEmpleado.establecerNombreEmp(nomEmp)
        objEmpleado.establecerHorasTrab(hrsTra)
        objEmpleado.establecerCuotaHora(cuoHr)
    f. Calcular objEmpleado.calcularSueldo()
    g. Imprimir objEmpleado.obtenerNombreEmp()
        objEmpleado.obtenerSueldo()
    h. Fin Método principal
Fin Clase EjecutaEmpleado5
Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Empleado5.java y EjecutaEmpleado5.java



En Java se genera un archivo .java por cada clase.

#### *Explicación:*

Se tiene el encabezado del algoritmo:

Algoritmo CALCULAR SUELDO DE UN EMPLEADO

El cual tiene dos clases: Empleado5 y EjecutaEmpleado5

Clase Empleado5

Inicia Empleado5, que es la clase modelo que representa al objeto empleado que está presente en este problema, la cual está formada por la parte de declaraciones y seis métodos:

1. Declarar

Datos

```

nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real

```





Observe que aunque se están declarando variables que representan datos, este autor prefiere usar declarar datos, porque esta clase es la clase modelo, que se utilizará en la clase ejecuta como molde para crear objetos, y el concepto más preciso en este punto es datos, sin embargo, podría ser equivalente ponerle declarar atributos o declarar variables.

Este paso permite declarar los datos o atributos que representan al objeto empleado, que se creará a partir de esta clase, es decir, los datos: Nombre del empleado, número de horas trabajadas, cuota que se le paga por hora trabajada y sueldo.

2. Método establecerNombreEmp (nom: Cadena)
  - a. nombreEmp = nom
  - b. Fin Método establecerNombreEmp

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato nombreEmp (nombre del empleado). Observe que nom es un parámetro que recibe el valor que se le envía desde donde se llama a este método, y el valor que recibe, lo coloca en el dato nombreEmp.

3. Método establecerHorasTrab (horasTr: Entero)
  - a. horasTrab = horasTr
  - b. Fin Método establecerHorasTrab

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato horasTrab (número de horas trabajadas). Observe que horasTr es un parámetro que recibe el valor que se le envía desde donde se llama a este método, y el valor que recibe, lo coloca en el dato horasTrab.

4. Método establecerCuotaHora (cuotaHr: Real)
  - a. cuotaHora = cuotaHr
  - b. Fin Método establecerCuotaHora

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato cuotaHora (cuota por hora). Observe que cuotaHr es un parámetro que recibe el valor que se le envía desde donde se llama a este método, y el valor que recibe, lo coloca en el dato cuotaHora.

5. Método calcularSueldo()
  - a. sueldo = horasTrab \* cuotaHora
  - b. Fin Método calcularSueldo

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato sueldo. Observe que se multiplica el valor del dato horasTrab por el valor del dato cuotaHora y el valor resultante se coloca en el dato sueldo.

6. Método obtenerNombreEmp() Cadena
  - a. return nombreEmp
  - b. Fin Método obtenerNombreEmp

Este paso es un método (getter) que permite acceder el valor del dato nombreEmp. Observe que lo único que hace es retornar (return) el valor del dato nombreEmp, al punto desde donde se llama a este método. Cadena es el tipo de dato del valor regresa.

7. Método obtenerSueldo() Real
  - a. return sueldo
  - b. Fin Método obtenerSueldo

Este paso es un método (getter) que permite acceder el valor del dato sueldo. Observe que lo único que hace es retornar (return) el valor del dato sueldo, al punto desde donde se llama a este método. Real es el tipo de dato del valor que regresa.

Fin Clase Empleado5

Indica el fin de la clase Empleado5.

Clase EjecutaEmpleado5

Inicia EjecutaEmpleado5, que es la clase controlador que utiliza a la clase Empleado5, para controlar la interacción con el usuario y permite que el programa sea operado por el usuario para resolver su problema. No obstante que puede tener más métodos, en problemas con la magnitud que enfrentaremos en este libro, sólo tiene el método principal, que a continuación se explica:

1. Método principal()

Es el encabezado del método principal e indica el inicio del mismo.

a. Declarar

Indica que se hacen declaraciones.

Variables

nomEmp: Cadena

hrsTra: Entero

cuoHr: Real

Se declaran las variables necesarias para implementar la lógica del algoritmo, en este caso sólo se declaran variables para leer cada uno de los datos a los que hay que dar entrada. Éstas son variables estáticas que se almacenan en el segmento de datos de la memoria; son como las variables normales que se usan en la programación tradicional o estructurada.

nomEmp      Para leer el nombre del empleado.

hrsTra      Para leer el número de horas trabajadas.

cuoHr      Para leer la cuota por hora.

b. Declarar, crear e iniciar objeto

Indica que se declara(n), se genera(n) e inicia(n) objeto(s); en este caso se crea un solo objeto:

Empleado5 objEmpleado = new Empleado5()

*En donde:*

Empleado5      Es la clase modelo que se usa como base para declarar y crear el objeto.

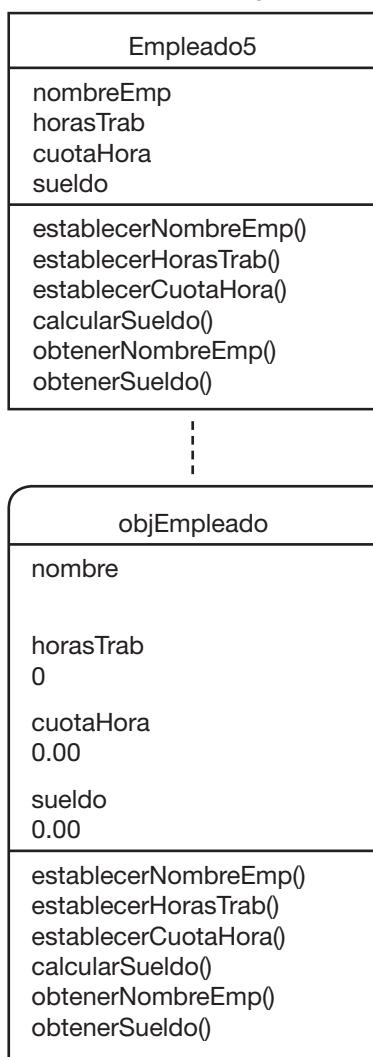
objEmpleado      Es el identificador del objeto.

new      Es una función que crea un nuevo objeto.

`Empleado5 ()`

Es el método constructor que new utiliza para crear el objeto `objEmpleado` tomando como base la clase `Empleado5` y establece sus valores iniciales en la memoria. Para más detalle, véase el punto 11.3 Constructores y destructores.

Gráficamente sucede lo siguiente:



*Explicación:*

Se genera el objeto `objEmpleado` tomando como base la clase `Empleado5`, el objeto creado en la memoria dinámica (Heap) tiene dos elementos. Un elemento son los datos que lo representan: `nombreEmp`, que se inicia en nulo; `horasTrab`,

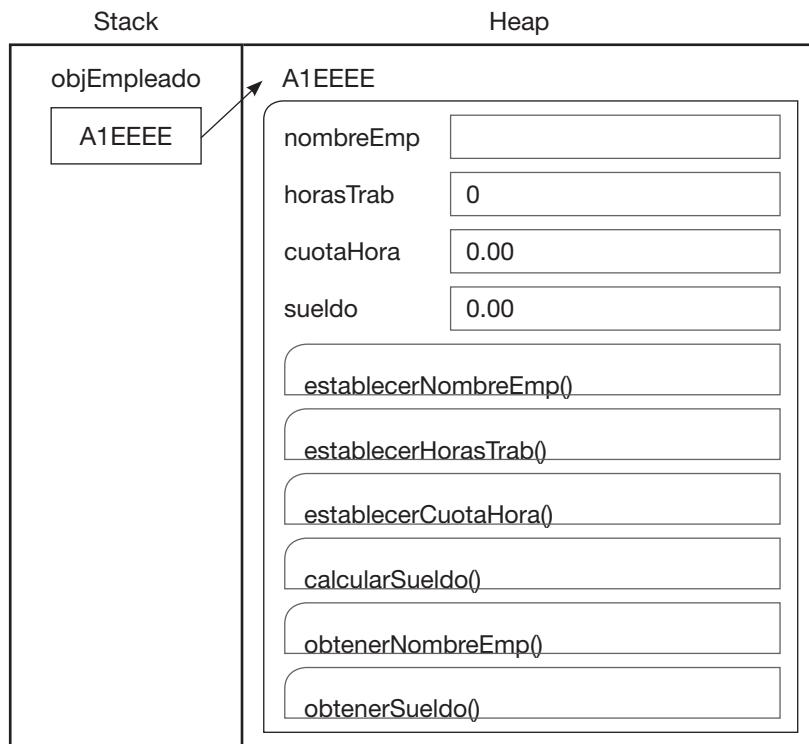
que se inicia en cero; cuotaHora, que se inicia en cero y sueldo, que se inicia en cero. El otro elemento son los métodos: establecerNombreEmp(), establecerHorasTrab(), establecerCuotaHora(), calcularSueldo(), obtenerNombreEmp() y obtenerSueldo(), los cuales permiten acceder a los datos del objeto, ya sea para establecer o para obtener valores.

En la memoria sucede lo siguiente:

Los lenguajes de programación dividen la memoria de la computadora en: Segmento de código (Code), segmento de datos (Data), segmento de pila (Stack) y segmento de variables dinámicas (Heap).

En el segmento de código se almacenan las instrucciones del programa. En el segmento de datos se almacenan las variables estáticas, son las variables normales que se usan en la programación tradicional o estructurada. En el segmento de pila (Stack) se almacenan los datos temporales, y en el segmento de variables dinámicas (Heap) es donde se almacenan los objetos.

Nuestro ejemplo, en la memoria se representa así:



*Explicación:*

En el Stack se crea una referencia (objEmpleado) a una dirección (A1EEEE) en el Heap que es donde se asigna espacio para alojar el objeto creado el cual tiene los datos que lo representan y los métodos que permiten acceder a los datos para establecerlos u obtenerlos.

- c. Solicitar Nombre, número de horas trabajadas y cuota por hora
- d. Leer nomEmp, hrsTra, cuoHr

Se solicita el nombre del empleado y se lee el valor tecleado en nomEmp.

Se solicita el número de horas trabajadas y se lee el valor tecleado en hrsTra.

Se solicita la cuota por hora y se lee el valor tecleado en cuoHr.

Vamos a suponer que se le dio entrada a los siguientes valores:

|        |                         |
|--------|-------------------------|
| nomEmp | Socorro Román Maldonado |
| hrsTra | 40                      |
| cuoHr  | 100.00                  |

- e. Establecer objEmpleado.establecerNombreEmp (nomEmp)
- objEmpleado.establecerHorasTrab (hrsTra)
- objEmpleado.establecerCuotaHora (cuoHr)

En este paso se procede a colocar los valores leídos en los datos del objeto, a continuación se explica cada acción:

`objEmpleado.establecerNombreEmp (nomEmp)`

Se llama al método establecerNombreEmp del objeto objEmpleado, enviando como parámetro nomEmp, es decir, que el valor de nomEmp que es Socorro Román Maldonado, se envía al método establecerNombreEmp, en donde se tiene el parámetro nom que recibe este valor que se le envía desde aquí, y lo coloca en el dato nombreEmp. Observe que nomEmp es una variable que reside en el segmento de datos de la memoria, y nombreEmp reside en el objeto, que a su vez reside en la memoria dinámica (Heap).

`objEmpleado.establecerHorasTrab (hrsTra)`

Se llama al método establecerHorasTrab del objeto objEmpleado, enviando como parámetro hrsTra, es decir, que el valor de hrsTra que es 40, se envía al método establecerHorasTrab, en donde se tiene el parámetro horasTr que recibe este valor que se le envía desde aquí, y lo coloca en el dato horasTrab. Observe que hrsTra es una variable que reside en el segmento de datos de la memoria, y horasTrab reside en el objeto que a su vez reside en la memoria dinámica (Heap).

`objEmpleado.establecerCuotaHora (cuoHr)`

Se llama al método establecerCuotaHora del objeto objEmpleado, enviando como parámetro cuoHr, es decir, que el valor de cuoHr que es 100.00, se envía al método establecerCuotaHora, en donde se tiene el parámetro cuotaHr que recibe este valor que se le envía desde aquí, y lo coloca en el dato cuotaHora. Observe que cuoHr es una variable que reside en el segmento de datos de la memoria, y cuotaHora reside en el objeto, que a su vez reside en la memoria dinámica (Heap).

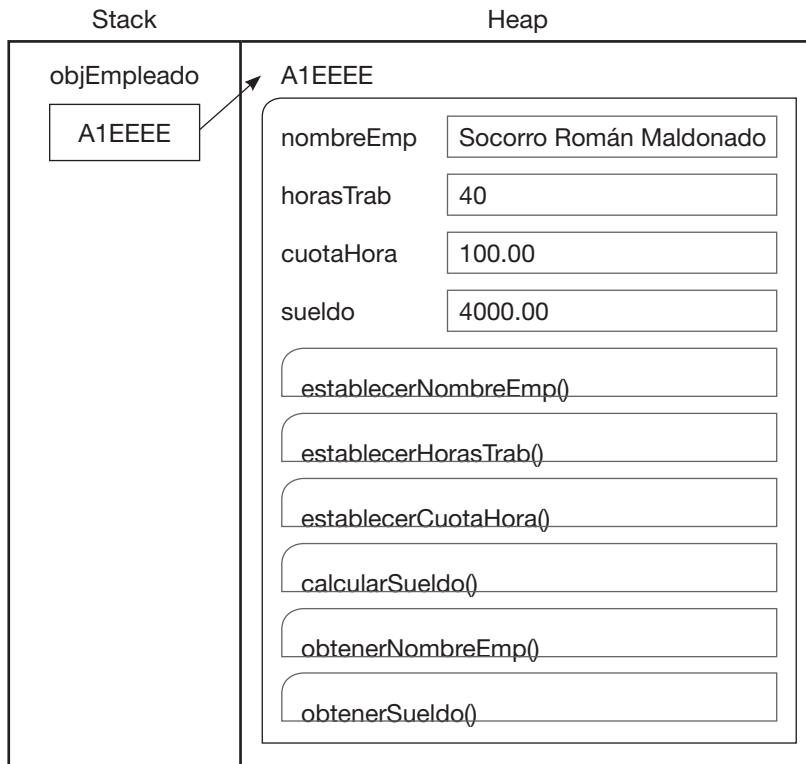
f. Calcular objEmpleado.calcularSueldo()

Se llama al método calcularSueldo del objeto objEmpleado, donde calcula el sueldo: Multiplicando el valor del dato horasTrab por el valor del dato cuotaHora y el valor resultante 4000.00, lo coloca en el dato sueldo. Gráficamente el objeto nos queda como se muestra en la siguiente figura:

| objEmpleado                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------|
| nombreEmp<br>Socorro Román Maldonado                                                                                                 |
| horasTrab<br>40                                                                                                                      |
| cuotaHora<br>100.00                                                                                                                  |
| sueldo<br>4000.00                                                                                                                    |
| establecerNombreEmp()<br>establecerHorasTrab()<br>establecerCuotaHora()<br>calcularSueldo()<br>obtenerNombreEmp()<br>obtenerSueldo() |

Es decir, que los datos del objeto tomaron valores a través de los métodos del mismo objeto.

Nuestro ejemplo, en la memoria ahora queda así:



Observe que los datos del objeto han tomado los valores que les fueron introducidos a través de los métodos correspondientes.

g. Imprimir `objEmpleado.obtenerNombreEmp()`  
`objEmpleado.obtenerSueldo()`

En este paso se procede a imprimir o dar salida a los valores de los datos del objeto, que se requirieron al plantear el problema; enseguida se explican:

`objEmpleado.obtenerNombreEmp()`

Se llama al método `obtenerNombreEmp` del objeto `objEmpleado`, el cual lo que hace es acceder y retornar (return) el valor del dato `nombreEmp`, a este punto para imprimirlo.

`objEmpleado.obtenerSueldo()`

Se llama al método `obtenerSueldo` del objeto `objEmpleado`, el cual lo que hace es acceder y retornar (return) el valor del dato `sueldo`, a este punto para imprimirlo.

h. Fin Método principal

Indica el fin del método principal.

Fin Clase EjecutaEmpleado5

Indica el fin de la clase EjecutaEmpleado5.

Fin

Indica el fin del algoritmo.

Resumiendo:

En la clase controlador EjecutaEmpleado5, en el método principal(), es donde se implementa la secuencia de pasos lógica que controla la solución del problema:

- a. Se declaran variables normales (estáticas) para dar entrada a los datos.
- b. Se declara, crea e inicializa el objeto empleado utilizando el modelo, la clase Empleado5. En el objeto creado, están empaquetados o encapsulados juntos los datos y los métodos; los datos, que representan la estructura del objeto, sólo pueden accederse a través de los métodos del mismo objeto.
- c. Se solicitan los datos.
- d. Se leen los datos en las variables declaradas en este método principal,
- e. Se colocan en los datos del objeto a través de los métodos del mismo objeto: establecerNombreEmp(), establecerHorasTrab(), establecerCuotaHora().
- f. Se hace el cálculo del sueldo llamando al método calcularSueldo().
- g. Se Imprimen los datos de salida, accediendo a los datos del objeto a través de los métodos obtenerNombreEmp() y obtenerSueldo().

Esto es, se establece un proceso interactivo para que el usuario interactúe con la computadora para solucionar su problema, introduciendo los datos que se le solicitan y recibiendo los resultados a través de la parte vista, incluida en esta clase controlador.

## 11.3 Constructores y destructores

Los lenguajes orientados a objetos como Java proporcionan por defecto (default) un método constructor y un proceso destructor para cada clase definida.

### Constructor

El método constructor es ejecutado con new y su función es asignar espacio y crear el objeto en la memoria dinámica (Heap). Asimismo, inicia los datos con los valores que se hayan asignado en la declaración; o bien, si no fueron iniciados con valores en la declaración, se inician con los valores por defecto (default). Los datos numéricos en ceros y los datos cadena (string) en nulos. El constructor tiene el mismo nombre que la clase pero con () al final. En el ejemplo del punto anterior la clase es Empleado5, y su constructor es Empleado5().

Si en nuestro ejemplo quisieramos iniciar los datos del objeto con valores específicos, necesitaríamos el método constructor:

```

2. Método Empleado5()
    a. nombreEmp = "María"
        horasTrab = 40
        cuotaHora = 100.00
        sueldo = 4000.00
    b. Fin Método Empleado5

```

Los datos se están iniciando con los valores especificados, y la numeración de los demás métodos se modificaría.



Aunque los lenguajes orientados a objetos permiten definir el constructor, y en el caso del destructor, algunos lenguajes también permiten definirlo y/o llamarlo, en este libro estaremos usando los conceptos por defecto (default), explicados en los párrafos anteriores.

### Destructor

El proceso destructor hace lo contrario que el constructor, es decir, libera el espacio ocupado por los objetos creados en la memoria dinámica. El proceso destructor, por defecto (default) entra en acción cada determinado tiempo y destruye los objetos creados que ya no tienen referencias, esto es, que ya no se utilizan.



Es pertinente recordar que todo algoritmo orientado a objetos se elabora en dos pasos. Primero, se diseña el diagrama de clases, aplicando los lineamientos que se explicaron en el capítulo 10. Segundo, se diseña la lógica de cada una de las clases usando pseudocódigo, aplicando los lineamientos que se explicaron en los puntos anteriores de este capítulo.

## 11.4 Ejercicios resueltos

En este punto se presentan ejercicios resueltos aplicando la metodología propuesta en el capítulo anterior y en éste, pero también se usan conceptos desarrollados en los primeros tres capítulos; los problemas resueltos en este punto son algunos de los ejercicios resueltos del capítulo 3, es decir, que el mismo problema ya lo resolvimos aplicando la programación estructurada en el capítulo 3, y aquí lo estamos resolviendo aplicando la lógica de la programación orientada a objetos.

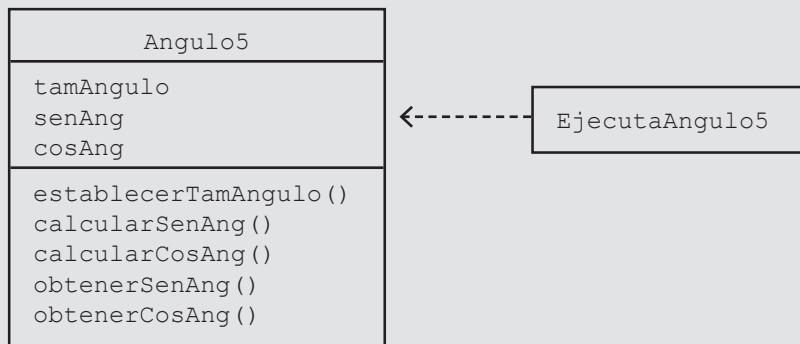
### Ejercicio 11.4.1

Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes, luego que calcule e imprima el seno y coseno.

A continuación se presenta el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*

## Diagrama de clases



## Algoritmo CÁLCULOS LOGARÍTMICOS DE ÁNGULO

Clase Angulo5

1. Declarar
  - Datos
    - tamAngulo, senAng, cosAng: Real
2. Método establecerTamAngulo(ang: Real)
  - a. tamAngulo = ang
  - b. Fin Método establecerTamAngulo
3. Método calcularSenAng()
  - a. senAng = Seno(tamAngulo)
  - b. Fin Método calcularSenAng
4. Método calcularCosAng()
  - a. cosAng = Coseno(tamAngulo)
  - b. Fin Método calcularCosAng
5. Método obtenerSenAng() Real
  - a. return senAng
  - b. Fin Método obtenerSenAng
6. Método obtenerCosAng() Real
  - a. return cosAng
  - b. Fin Método obtenerCosAng

Fin Clase Angulo5

Clase EjecutaAngulo5

1. Método principal()
  - a. Declarar
    - Variabes
    - tamAng: Real
  - b. Declarar, crear e iniciar objeto
    - Angulo5 objAngulo = new Angulo5()
  - c. Solicitar Tamaño del ángulo en radianes
  - d. Leer tamAng
  - e. Establecer

```

        objAngulo, establecerTamAngulo(tamAng)
f. Calcular
    objAngulo.calcularSenAng()
    objAngulo.calcularCosAng()
g. Imprimir
    objAngulo.obtenerSenAng()
    objAngulo.obtenerCosAng()
h. Fin Método principal
Fin Clase EjecutaAngulo5
Fin

```



En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Angulo5.java y EjecutaAngulo5.java

*Explicación:*

El algoritmo tiene dos clases; la Clase Angulo5 y la clase EjecutaAngulo5

En la Clase Angulo5:

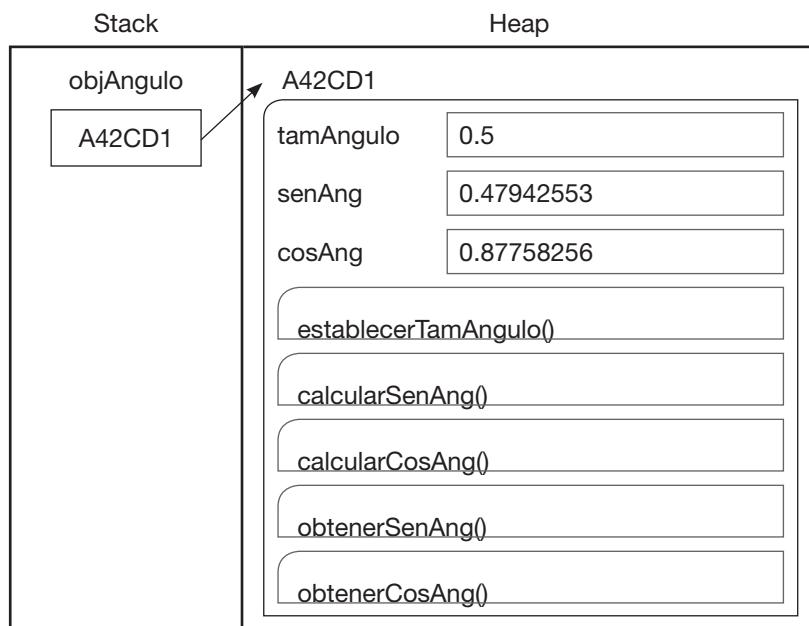
1. Se declaran los datos que representan la estructura de la clase:  
tamAngulo para el tamaño del ángulo  
senAng para el seno  
cosAng para el coseno
  2. Método establecerTamAngulo(ang: Real)  
Recibe en el parámetro ang el valor que luego coloca en el dato tamAngulo
  3. Método calcularSenAng()  
a. Calcula el seno de tamAngulo llamando a la función Seno(tamAngulo)
  4. Método calcularCosAng()  
a. Calcula el coseno de tamAngulo llamando a la función Coseno(tamAngulo)
  5. Método obtenerSenAng() Real  
a. Retorna senAng (seno de tamAngulo)
  6. Método obtenerCosAng() Real  
a. Retorna cosAng (coseno de tamAngulo)
- Fin de la Clase Angulo5

En la Clase EjecutaAngulo5; en el Método principal():

- a. Se declara:  
La variable tamAng para leer el tamaño del ángulo
- b. Se declara el objeto objAngulo, usando como base a la clase Angulo5;  
dicho objeto se crea e inicializa mediante el constructor por defecto Angulo5()

- c. Se solicita el tamaño del ángulo en radianes
  - d. Se lee en tamAng
  - e. Se llama al método establecerTamAngulo(tamAng) del objeto objAngulo; para colocar el valor de tamAng en el dato tamAngulo
  - f. Se llama al método calcularSenAng() del objeto objAngulo; para calcular el seno del dato tamAngulo  
Se llama al método calcularCosAng() del objeto objAngulo; para calcular el coseno del dato tamAngulo
  - g. Se llama al método obtenerSenAng() del objeto objAngulo; para acceder e imprimir el valor del dato senAng  
Se llama al método obtenerCosAng() del objeto objAngulo; para acceder e imprimir el valor del dato cosAng
  - h. Fin del Método
- Fin de la Clase EjecutaAngulo5
- Fin del algoritmo

En la siguiente figura se muestra cómo se ve el objeto en la memoria:



*Explicación:*

En el Stack se crea una referencia objAngulo a la dirección A42CD1 del Heap, donde se almacena el objeto creado, el cual tiene los datos y métodos que ya se explicaron. Observe que los datos del objeto han tomado los valores que les fueron introducidos a través de los métodos correspondientes.

**Ejercicio 11.4.2**

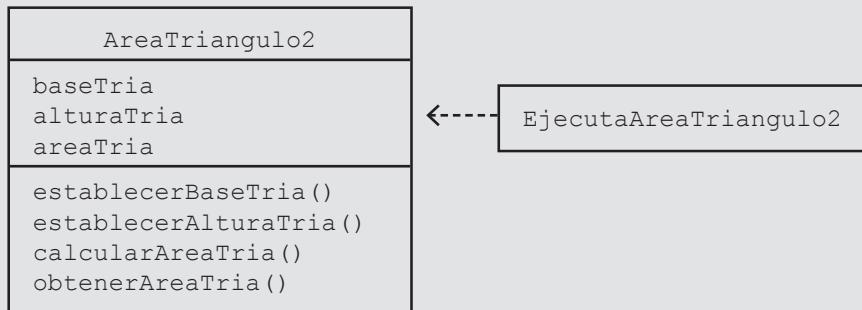
Elaborar un algoritmo para calcular el área de un triángulo. Se requiere imprimir como salida el área del triángulo. Los datos disponibles para leer como entrada son la base y la altura del triángulo.

$$\text{El área se calcula: Área} = \frac{\text{Base} \times \text{Altura}}{2}$$

A continuación se presenta el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*

Diagrama de clases



Algoritmo ÁREA TRIÁNGULO

Clase AreaTriangulo2

1. Declarar
    - Datos
 

```

baseTria: Real
alturaTria: Real
areaTria: Real

```
  2. Método establecerBaseTria(base: Real)
    - a. baseTria = base
    - b. Fin Método establecerBaseTria
  3. Método establecerAlturaTria(altura: Real)
    - a. alturaTria = altura
    - b. Fin Método establecerAlturaTria
  4. Método calcularAreaTria()
    - a. areaTria = (baseTria \* alturaTria) / 2
    - b. Fin Método calcularAreaTria
  5. Método obtenerAreaTria() Real
    - a. return areaTria
    - b. Fin Método obtenerAreaTria
- Fin Clase AreaTriangulo2

```
Clase EjecutaAreaTriangulo2
1. Método principal()
    a. Declarar
        Variables
            basTri, altuTri: Real
    b. Declarar, crear e iniciar objeto
        AreaTriangulo2 objTriangulo =
            new AreaTriangulo2()
    c. Solicitar Base y Altura
    d. Leer basTri, altuTri
    e. Establecer
        objTriangulo.establecerBaseTria(basTri)
        objTriangulo.establecerAlturaTria(altuTri)
    f. Calcular objTriangulo.calcularAreaTria()
    g. Imprimir objTriangulo.obtenerAreaTria()
    h. Fin Método principal
Fin Clase EjecutaAreaTriangulo2
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: AreaTriangulo2.java y EjecutaAreaTriangulo2.java



*Explicación:*

El algoritmo tiene dos clases: la clase AreaTriangulo2 y la clase EjecutaAreaTriangulo2

En la Clase AreaTriangulo2:

1. Se declaran los datos que representan la estructura de la clase:  
baseTria para la base del triángulo  
alturaTria para la altura del triángulo  
areaTria para el área del triángulo
2. Método establecerBaseTria(base: Real)  
Recibe en el parámetro base el valor que luego coloca en el dato baseTria
3. Método establecerAlturaTria(altura: Real)  
Recibe en el parámetro altura el valor que luego coloca en el dato alturaTria
4. Método calcularAreaTria()  
Calcula el área del triángulo
5. Método obtenerAreaTria() Real  
Retorna areaTria (área del triángulo)

Fin de la Clase AreaTriangulo2

En la Clase EjecutaAreaTriangulo2, en el Método principal():

- a. Se declara:  
La variable basTri para leer la base del triángulo  
La variable altuTri para leer la altura del triángulo
- b. Se declara el objeto objTriangulo, usando como base a la clase AreaTriangulo2; dicho objeto se crea e inicializa mediante el constructor por defecto AreaTriangulo2()
- c. Se solicita la base y la altura
- d. Se leen en basTri y altuTri
- e. Se llama al método establecerBaseTria(basTri) del objeto objTriangulo, para colocar el valor de basTri en el dato baseTria  
Se llama al método establecerAlturaTria(altuTri) del objeto objTriangulo, para colocar el valor de altuTri en el dato alturaTria
- f. Se llama al método calcularAreaTria() del objeto objTriangulo, para calcular el área del triángulo
- g. Se llama al método obtenerAreaTria() del objeto objTriangulo, para acceder e imprimir el valor del dato areaTria
- h. Fin del Método

Fin de la Clase EjecutaAreaTriangulo2

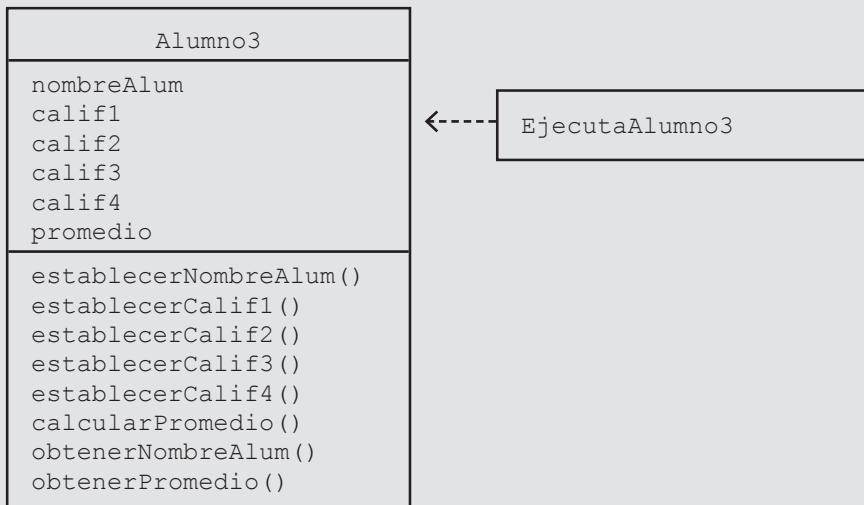
Fin del algoritmo

#### Ejercicio 11.4.3

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4, de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el Nombre y el promedio de las calificaciones. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4.

A continuación se presenta el algoritmo de la solución: (*Primero hágalo usted, después compare la solución.*)

## Diagrama de clases



## Algoritmo CALIFICACIÓN ALUMNO

Clase Alumno3

1. Declarar
  - Datos
 

```

nombreAlum: Cadena
calif1: Real
calif2: Real
calif3: Real
calif4: Real
promedio: Real

```
2. Método establecerNombreAlum(nom: Cadena)
  - a. nombreAlum = nom
  - b. Fin Método establecerNombreAlum
3. Método establecerCalif1(cal1: Real)
  - a. calif1 = cal1
  - b. Fin Método establecerCalif1
4. Método establecerCalif2(ca2: Real)
  - a. calif2 = ca2
  - b. Fin Método establecerCalif2
5. Método establecerCalif3(ca3: Real)
  - a. calif3 = ca3
  - b. Fin Método establecerCalif3
6. Método establecerCalif4(ca4: Real)
  - a. calif4 = ca4
  - b. Fin Método establecerCalif4
7. Método calcularPromedio()
  - a. promedio = (calif1+ calif2+ calif3+ calif4)/4
  - b. Fin Método calcularPromedio

```
8. Método obtenerNombreAlum() Cadena
    a. return nombreAlum
    b. Fin Método obtenerNombreAlum
9. Método obtenerPromedio() Real
    a. return promedio
    b. Fin Método obtenerPromedio
Fin Clase Alumno3
Clase EjecutaAlumno3
1. Método principal()
    a. Declarar
        Variables
        nombre: Cadena
        c1, c2, c3, c4: Real
    b. Declarar, crear e iniciar objeto
        Alumno3 objAlumno = new Alumno3()
    c. Solicitar Nombre del alumno, calificación 1,
        calificación 2, calificación 3 y calificación 4
    d. Leer nombre, c1, c2, c3, c4
    e. Establecer
        objAlumno.establecerNombreAlum(nombre)
        objAlumno.establecerCalif1(c1)
        objAlumno.establecerCalif2(c2)
        objAlumno.establecerCalif3(c3)
        objAlumno.establecerCalif4(c4)
    f. Calcular objAlumno.calcularPromedio()
    g. Imprimir
        objAlumno.obtenerNombreAlum()
        objAlumno.obtenerPromedio()
    h. Fin Método principal
Fin Clase EjecutaAlumno3
Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Alumno3.java y EjecutaAlumno3.java

#### *Explicación:*

El algoritmo tiene dos clases: la Clase Alumno3 y la clase EjecutaAlumno3

En la Clase Alumno3:

1. Se declaran los datos que representan la estructura de la clase:  
nombreAlum para el nombre del alumno  
calif1 para la calificación 1 del alumno  
calif2 para la calificación 2 del alumno

calif3 para la calificación 3 del alumno  
calif4 para la calificación 4 del alumno  
promedio para el promedio del alumno

2. Método establecerNombreAlum(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombreAlum
3. Método establecerCalif1(ca1: Real)  
Recibe en el parámetro ca1 el valor que luego coloca en el dato calif1
4. Método establecerCalif2(ca2: Real)  
Recibe en el parámetro ca2 el valor que luego coloca en el dato calif2
5. Método establecerCalif3(ca3: Real)  
Recibe en el parámetro ca3 el valor que luego coloca en el dato calif3
6. Método establecerCalif4(ca4: Real)  
Recibe en el parámetro ca4 el valor que luego coloca en el dato calif4
7. Método calcularPromedio()  
Calcula el promedio del alumno
8. Método obtenerNombreAlum() Cadena  
Retorna nombreAlum (nombre del alumno)
9. Método obtenerPromedio() Real  
Retorna promedio (promedio del alumno)

Fin de la Clase Alumno3

En la Clase EjecutaAlumno3, en el Método principal():

- a. Se declara:
  - La variable nombre para leer el nombre del alumno
  - La variable c1 para leer la calificación 1 del alumno
  - La variable c2 para leer la calificación 2 del alumno
  - La variable c3 para leer la calificación 3 del alumno
  - La variable c4 para leer la calificación 4 del alumno
- b. Se declara el objeto objAlumno, usando como base a la clase Alumno3; dicho objeto se crea e inicializa mediante el constructor por defecto Alumno3()
- c. Se solicitan Nombre, calificación 1, calificación 2, calificación 3 y calificación 4
- d. Se leen en nombre, c1, c2, c3, c4
- e. Se llama al método establecerNombreAlum(nombre) del objeto objAlumno, para colocar el valor de nombre en el dato nombreAlum  
Se llama al método establecerCalif1(c1) del objeto objAlumno, para colocar el valor de c1 en el dato calif1  
Se llama al método establecerCalif2(c2) del objeto objAlumno, para colocar el valor de c2 en el dato calif2  
Se llama al método establecerCalif3(c3) del objeto objAlumno, para colocar el valor de c3 en el dato calif3  
Se llama al método establecerCalif4(c4) del objeto objAlumno, para colocar el valor de c4 en el dato calif4

- f. Se llama al método calcularPromedio() del objeto objAlumno, para calcular el promedio.
  - g. Se llama al método obtenerNombreAlum() del objeto objAlumno; para acceder e imprimir el valor del dato nombreAlum  
Se llama al método obtenerPromedio() del objeto objAlumno, para acceder e imprimir el valor del dato promedio
  - h. Fin del Método
- Fin de la Clase EjecutaAlumno3
- Fin del algoritmo

La tabla 11.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 11 de la Web del libro.

**Tabla 11.1**

| Ejercicio        | Descripción                                      |
|------------------|--------------------------------------------------|
| Ejercicio 11.4.4 | Calcula el precio de venta de un artículo        |
| Ejercicio 11.4.5 | Calcula la hipotenusa de un triángulo rectángulo |
| Ejercicio 11.4.6 | Convierte grados a radianes y viceversa          |

## 11.5 Ejercicios propuestos

Como ejercicios propuestos para este punto, se recomiendan, del capítulo 3, algunos de los ejercicios resueltos que no fueron incluidos en este punto; además, todos los ejercicios propuestos en dicho capítulo.

## 11.6 Resumen de conceptos que debe dominar

- Cómo diseñar algoritmos orientados a objetos aplicando la estructura de secuenciación en seudocódigo
- Constructores y destructores

## 11.7 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

### 11.7.1 Resumen gráfico del capítulo

### 11.7.2 Autoevaluación

### 11.7.3 Programas

### 11.7.4 Ejercicios resueltos

### 11.7.5 Power Point para el profesor (\*)

# 12

## Programación orientada a objetos aplicando las estructuras de selección

### Contenido

- 12.1 Diseño de algoritmos OO usando la selección doble (if then else)
- 12.2 Diseño de algoritmos OO usando la selección simple (if then)
- 12.3 Diseño de algoritmos OO usando la selección múltiple (switch)
- 12.4 Ejercicios resueltos
- 12.5 Ejercicios propuestos
- 12.6 Resumen de conceptos que debe dominar
- 12.7 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 12.7.1 Resumen gráfico del capítulo
  - 12.7.2 Autoevaluación
  - 12.7.3 Programas
  - 12.7.4 Ejercicios resueltos
  - 12.7.5 Power Point para el profesor (\*)

### Objetivo del capítulo

- Aprender cómo diseñar algoritmos orientados a objetos aplicando las estructuras de selección: if then else, if then y switch.

## Introducción

Con el estudio del capítulo anterior, usted ya domina los conceptos básicos de la programación orientada a objetos aplicando la estructura de secuenciación y sabe cómo diseñar algoritmos usando dicha estructura.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos mediante la programación orientada a objetos aplicando las estructuras de selección.

Se explica cómo diseñar algoritmos orientados a objetos usando las estructuras de control selección que estudiamos en el capítulo 4, pero ahora inmersas en la arquitectura orientada a objetos, que se estudió en los dos capítulos anteriores.

Recordemos que la selección doble (if-then-else), sirve para plantear situaciones en la que se tienen dos alternativas de acción, y se debe escoger una. La selección simple (if-then), sirve para cuando se tiene una alternativa de acción condicionada. La selección múltiple (switch), sirve para cuando se tienen múltiples alternativas de acción, de las cuales se debe escoger una.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que, a veces, aunque haya diferencias, las dos soluciones son correctas.

En el siguiente capítulo se estudia la programación orientada a objetos aplicando las estructuras de repetición.

## 12.1 Diseño de algoritmos OO usando la selección doble (if then else)

En este capítulo se utilizarán las estructuras de selección en pseudocódigo, que se estudiaron en el capítulo 4, pero ahora aplicadas conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

En este punto se aplica la selección doble (if then else).

Ejemplo:

Calcular el Sueldo de un empleado.

*Información a imprimir como salida:*

Nombre y Sueldo

*Datos disponibles para leer:*

Nombre, número de horas trabajadas y cuota por hora

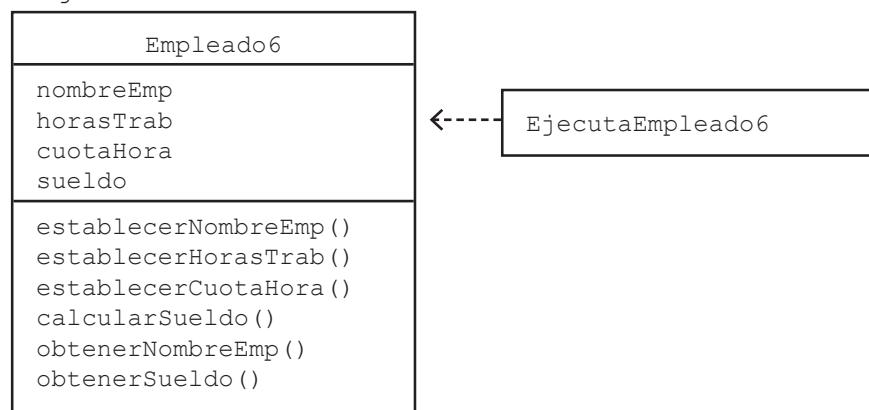
*Proceso:*

El Sueldo se calcula de la forma siguiente:

Si el número de horas trabajadas es mayor que 40, el excedente de 40 se paga al doble de la cuota por hora. En caso de no ser mayor que 40 se paga a la cuota por hora normal.

A continuación se presenta el diagrama de clases de la solución:

Diagrama de clases



*Explicación:*

Este diagrama consta de dos clases: una es la clase Empleado6 que es el modelo que representa y soluciona el problema planteado, la cual está formada por:

**Los datos:**

|           |                            |
|-----------|----------------------------|
| nombreEmp | Nombre del empleado        |
| horasTrab | Número de horas trabajadas |
| cuotaHora | Cuota por hora             |
| sueldo    | Sueldo del empleado        |

**Los métodos:**

|                       |                                                      |
|-----------------------|------------------------------------------------------|
| establecerNombreEmp() | Para establecer el valor del dato nombreEmp.         |
| establecerHorasTrab() | Para establecer el valor del dato horasTrab.         |
| establecerCuotaHora() | Para establecer el valor del dato cuotaHora.         |
| calcularSueldo()      | Para calcular y establecer el valor del dato sueldo. |
| obtenerNombreEmp()    | Para acceder e imprimir el valor del dato nombreEmp. |
| obtenerSueldo()       | Para acceder e imprimir el valor del dato sueldo.    |

La otra es la clase EjecutaEmpleado6, que es la clase controlador, la cual utiliza el modelo, que es la clase Empleado6, para controlar la interacción con el usuario representando y resolviendo su problema.

A continuación se presenta el algoritmo de la solución en pseudocódigo:

```

Algoritmo CALCULAR SUELDO DOBLE DE UN EMPLEADO
    Clase Empleado6
        1. Declarar
            Datos
                nombreEmp: Cadena
                horasTrab: Entero
                cuotaHora: Real
                sueldo: Real
        2. Método establecerNombreEmp (nom: Cadena)
            a. nombreEmp = nom
            b. Fin Método establecerNombreEmp
        3. Método establecerHorasTrab (horasTr: Entero)
            a. horasTrab = horasTr
            b. Fin Método establecerHorasTrab
        4. Método establecerCuotaHora (cuotaHr: Real)
            a. cuotaHora = cuotaHr
            b. Fin Método establecerCuotaHora
        5. Método calcularSueldo()
            a. if horasTrab <= 40 then
                1. sueldo = horasTrab * cuotaHora

```

```
b. else
    1. sueldo = (40*cuotaHora) +
               ((horasTrab-40)*(cuotaHora*2))
c. endif
d. Fin Método calcularSueldo
6. Método obtenerNombreEmp() Cadena
   a. return nombreEmp
   b. Fin Método obtenerNombreEmp
7. Método obtenerSueldo() Real
   a. return sueldo
   b. Fin Método obtenerSueldo
Fin Clase Empleado6
Clase EjecutaEmpleado6
1. Método principal()
   a. Declarar
      Variables
      nomEmp: Cadena
      hrsTra: Entero
      cuoHr: Real
   b. Declarar, crear e iniciar objeto
      Empleado6 objEmpleado = new Empleado6()
   c. Solicitar Nombre, número de horas trabajadas
      y cuota por hora
   d. Leer nomEmp, hrsTra, cuoHr
   e. Establecer
      objEmpleado.establecerNombreEmp(nomEmp)
      objEmpleado.establecerHorasTrab(hrsTra)
      objEmpleado.establecerCuotaHora(cuoHr)
   f. Calcular objEmpleado.calcularSueldo()
   g. Imprimir objEmpleado.obtenerNombreEmp()
      objEmpleado.obtenerSueldo()
   h. Fin Método principal
Fin Clase EjecutaEmpleado6
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Empleado6.java y EjecutaEmpleado6.java



*Explicación:*

El algoritmo tiene dos clases: la Clase Empleado6 y la clase EjecutaEmpleado6.

En la Clase Empleado6:

1. Se declaran los datos que representan la estructura de la clase:  
nombreEmp para el nombre del empleado  
horasTrab para el número de horas trabajadas  
cuotaHora para la cuota por hora  
sueldo para el sueldo del empleado
  2. Método establecerNombreEmp(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp
  3. Método establecerHorasTrab(horasTr: Entero)  
Recibe en el parámetro horasTr el valor que luego coloca en el dato horas-Trab
  4. Método establecerCuotaHora(cuotaHr: Real)  
Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuota-Hora
  5. Método calcularSueldo()
    - a. Si horasTrab <= 40 Entonces
      1. Se calcula el sueldo en forma simple
    - b. Si no
      1. Se calcula el sueldo; 40 horas a la cuota simple y el excedente de 40, al doble de la cuota
    - c. Fin del if
    - d. Fin del método calcularSueldo()
  6. Método obtenerNombreEmp() Cadena  
Retorna nombreEmp
  7. Método obtenerSueldo() Real  
Retorna sueldo
- Fin de la Clase Empleado6

En la Clase EjecutaEmpleado6, en el Método principal():

- a. Se declara:  
La variable nomEmp para leer el nombre del empleado  
La variable hrsTra para leer el número de horas trabajadas  
La variable cuoHr para leer la cuota por hora
- b. Se declara el objeto objEmpleado, usando como base a la clase Empleado6; dicho objeto se crea e inicializa mediante el constructor por defecto Empleado6()
- c. Se solicitan Nombre, número de horas trabajadas y cuota por hora
- d. Se leen en nomEmp, hrsTra, cuoHr
- e. Se llama al método establecerNombreEmp(nomEmp) del objeto objEmpleado, para colocar el valor de nomEmp en el dato nombreEmp

- Se llama al método establecerHorasTrab(hrsTra) del objeto objEmpleado, para colocar el valor de hrsTra en el dato horasTrab
- Se llama al método establecerCuotaHora(cuoHr) del objeto objEmpleado, para colocar el valor de cuoHr en el dato cuotaHora
- f. Se llama al método calcularSueldo() del objeto objEmpleado, para calcular el sueldo
- g. Se llama al método obtenerNombreEmp() del objeto objEmpleado, para acceder e imprimir el valor del dato nombreEmp
- Se llama al método obtenerSueldo() del objeto objEmpleado, para acceder e imprimir el valor del dato sueldo
- h. Fin del Método
- Fin de la Clase EjecutaEmpleado6
- Fin del algoritmo

Práctica: En este momento se recomienda practicar haciendo algunos ejercicios resueltos y propuestos para aplicar el if-then-else.

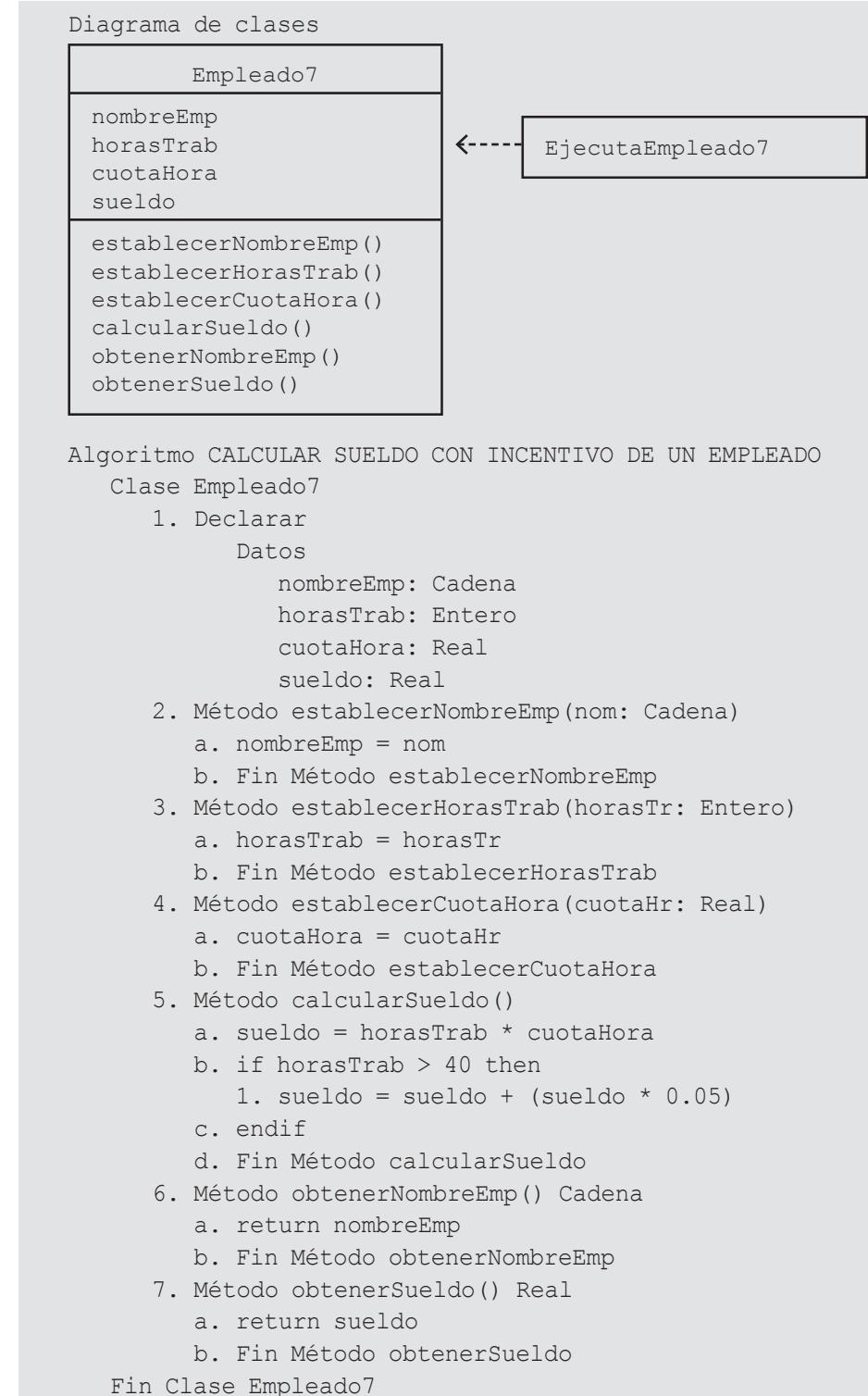
## 12.2 Diseño de algoritmos OO usando la selección simple (if then)

En este punto se aplica la selección simple (if then).

Ejemplo:

Siguiendo con el mismo problema de calcular el sueldo de un empleado, ahora se otorga un incentivo del 5% si el empleado trabajó más de 40 horas, esto se agrega independientemente del cálculo del sueldo.

A continuación se presenta el algoritmo de la solución:



```
Clase EjecutaEmpleado7
1. Método principal()
    a. Declarar
        Variables
            nomEmp: Cadena
            hrsTra: Entero
            cuoHr: Real
    b. Declarar, crear e iniciar objeto
        Empleado7 objEmpleado = new Empleado7()
    c. Solicitar Nombre, número de horas trabajadas
        y cuota por hora
    d. Leer nomEmp, hrsTra, cuoHr
    e. Establecer
        objEmpleado.establecerNombreEmp(nomEmp)
        objEmpleado.establecerHorasTrab(hrsTra)
        objEmpleado.establecerCuotaHora(cuoHr)
    f. Calcular objEmpleado.calcularSueldo()
    g. Imprimir objEmpleado.obtenerNombreEmp()
        objEmpleado.obtenerSueldo()
    h. Fin Método principal
Fin Clase EjecutaEmpleado7
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Empleado7.java y EjecutaEmpleado7.java



*Explicación:*

Este algoritmo es casi igual al del punto anterior, lo único diferente es en el método calcularSueldo(), en el cual ahora se hace el cálculo así:

5. Método calcularSueldo()
  - a. Se calcula el sueldo en forma normal (horasTrab \* cuotaHora)
  - b. Si horasTrab > 40 Entonces
    1. Al sueldo se le agrega el 5% del mismo sueldo
  - c. Fin del if
  - d. Fin del método

Práctica: En este momento se recomienda practicar haciendo algunos ejercicios resueltos y propuestos para aplicar el if-then.

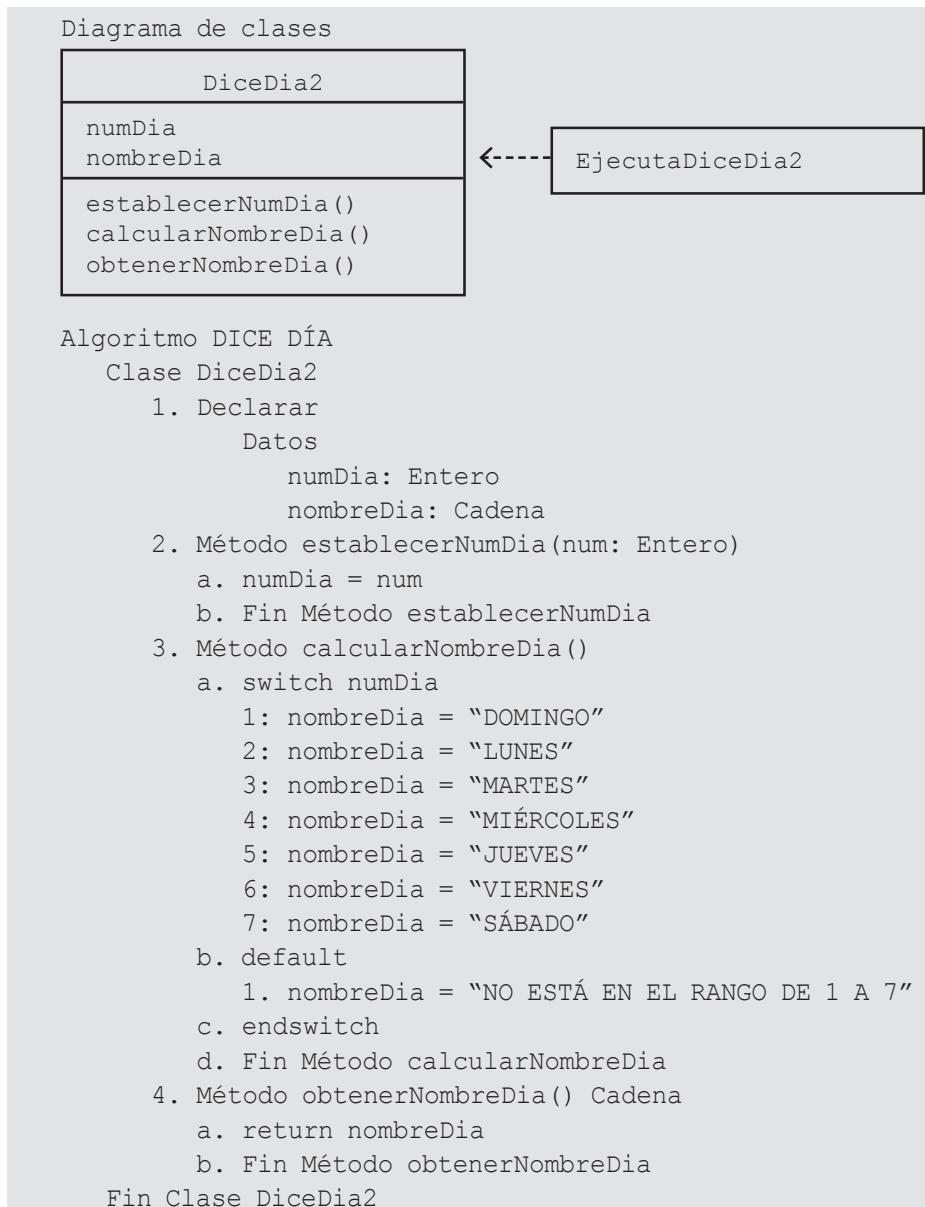
## 12.3 Diseño de algoritmos OO usando la selección múltiple (switch)

En este punto se aplica la selección múltiple (switch).

Ejemplo:

Elaborar un algoritmo que lea el número de día (un valor entre 1 y 7), e imprima domingo si es 1, lunes si es 2, ..., sábado si es 7.

A continuación se presenta el algoritmo de la solución:



```

Clase EjecutaDiceDia2
  1. Método principal()
    a. Declarar
      Variables
      nDia: Entero
    b. Declarar, crear e iniciar objeto
      DiceDia2 objDia = new DiceDia2()
    c. Solicitar Número de día
    d. Leer nDia
    e. Establecer objDia.establecerNumDia(nDia)
    f. Calcular objDia.calcularNombreDia()
    g. Imprimir objDia.obtenerNombreDia()
    h. Fin Método principal
  Fin Clase EjecutaDiceDia2
Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: DiceDia2.java y EjecutaDiceDia2.java



#### *Explicación:*

El algoritmo tiene dos clases: la Clase DiceDia2 y la clase EjecutaDiceDia2

#### En la Clase DiceDia2:

1. Se declaran los datos que representan la estructura de la clase:
  - numDia para el número de día
  - nombreDia para el nombre del día
2. Método establecerNumDia(num: Entero)
 

Recibe en el parámetro num el valor que luego coloca en el dato numDia
3. Método calcularNombreDia()
 

Establece el nombre del día:

  - a. switch numDia
    - Si numDia ==1 Entonces: nombreDia = “DOMINGO”
    - Si numDia ==2 Entonces: nombreDia = “LUNES”
    - Si numDia ==3 Entonces: nombreDia = “MARTES”
    - Si numDia ==4 Entonces: nombreDia = “MIÉRCOLES”
    - Si numDia ==5 Entonces: nombreDia = “JUEVES”
    - Si numDia ==6 Entonces: nombreDia = “VIERNES”
    - Si numDia ==7 Entonces: nombreDia = “SÁBADO”
  - b. Si no es ninguno, Entonces:
    1. nombreDia = “NO ESTÁ EN EL RANGO DE 1 A 7”

c. Fin del switch  
d. Fin del método  
4. Método obtenerNombreDia() Cadena  
Retorna nombreDia (nombre del dia)  
Fin de la Clase DiceDia2

En la Clase EjecutaDiceDia2, en el Método principal():

- a. Se declara:  
nDia para leer el número de día
  - b. Se declara el objeto objDia, usando como base a la clase DiceDia2; dicho objeto se crea e inicializa mediante el constructor por defecto DiceDia2()
  - c. Se solicita el número de día
  - d. Se lee en nDia
  - e. Se llama al método establecerNumDia(nDia) del objeto objDia, para colocar el valor de nDia en el dato numDia
  - f. Se llama al método calcularNombreDia() del objeto objDia, para calcular el nombre del día nombreDia
  - g. Se llama al método obtenerNombreDia() del objeto objDia, para acceder e imprimir el valor del dato nombreDia
  - h. Fin del Método
- Fin de la Clase EjecutaDiceDia2  
Fin del algoritmo

Práctica: En este momento se recomienda practicar haciendo el ejercicio resuelto y algunos propuestos para aplicar el switch.

## 12.4 Ejercicios resueltos

---

En este punto se presentan ejercicios resueltos aplicando la metodología propuesta en el capítulo anterior y en éste, pero también se usan conceptos desarrollados en los primeros cuatro capítulos; los problemas resueltos en este punto son algunos de los ejercicios resueltos del capítulo 4, es decir, que el mismo problema ya lo resolvimos aplicando la programación estructurada en el capítulo 4, y aquí lo estamos resolviendo aplicando la lógica de la programación orientada a objetos.

Se le recomienda que primero haga usted el algoritmo, y después compare su solución con la del libro. En esta parte hay ejercicios donde se utilizan las tres estructuras de selección. En los puntos anteriores, al estudiar cada estructura de selección se recomienda venir a esta parte y practicar con ejercicios correspondientes a la estructura que esté estudiando, de manera que a esta parte va a venir en varias ocasiones.

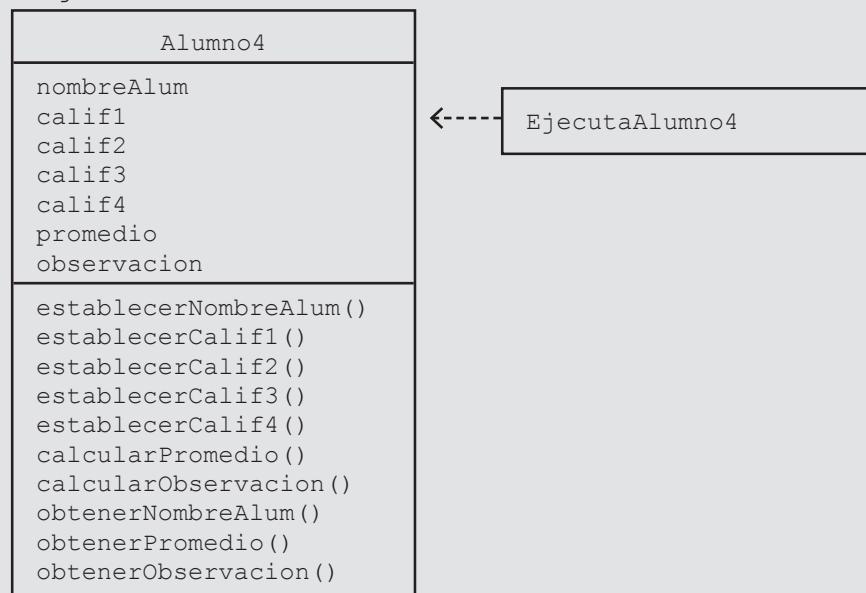
**Ejercicio 12.4.1**

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4, de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el Nombre, el promedio de las calificaciones y un comentario de “Aprobado” si obtiene 60 o más, o “Reprobado” en caso contrario. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4. Usando if-then-else.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted, después compare la solución.)

Diagrama de clases

**Algoritmo CALCULA PROMEDIO DE UN ALUMNO**

Clase Alumno4

1. Declarar

Datos

    nombreAlum: Cadena  
    calif1: Real  
    calif2: Real  
    calif3: Real  
    calif4: Real  
    promedio: Real  
    observacion: Cadena

```
2. Método establecerNombreAlum(nom: Cadena)
   a. nombreAlum = nom
   b. Fin Método establecerNombreAlum
3. Método establecerCalif1(cal1: Real)
   a. calif1 = cal1
   b. Fin Método establecerCalif1
4. Método establecerCalif2(cal2: Real)
   a. calif2 = cal2
   b. Fin Método establecerCalif2
5. Método establecerCalif3(cal3: Real)
   a. calif3 = cal3
   b. Fin Método establecerCalif3
6. Método establecerCalif4(cal4: Real)
   a. calif4 = cal4
   b. Fin Método establecerCalif4
7. Método calcularPromedio()
   a. promedio = (calif1+ calif2+ calif3+ calif4)/4
   b. Fin Método calcularPromedio
8. Método calcularObservacion()
   a. if promedio >= 60 then
      1. observacion = "Aprobado"
   b. else
      1. observacion = "Reprobado"
   c. endif
   d. Fin Método calcularObservacion
9. Método obtenerNombreAlum() Cadena
   a. return nombreAlum
   b. Fin Método obtenerNombreAlum
10. Método obtenerPromedio() Real
    a. return promedio
    b. Fin Método obtenerPromedio
11. Método obtenerObservacion() Cadena
    a. return observacion
    b. Fin Método obtenerObservacion
Fin Clase Alumno4
Clase EjecutaAlumno4
1. Método principal()
   a. Declarar
      Variables
      nombre: Cadena
      c1, c2, c3, c4: Real
   b. Declarar, crear e iniciar objeto
      Alumno4 objAlumno = new Alumno4()
```

```
c. Solicitar Nombre del alumno, calificación 1,  
    calificación 2, calificación 3 y calificación 4  
d. Leer nombre, c1, c2, c3, c4  
e. Establecer  
    objAlumno.establecerNombreAlum(nombre)  
    objAlumno.establecerCalif1(c1)  
    objAlumno.establecerCalif2(c2)  
    objAlumno.establecerCalif3(c3)  
    objAlumno.establecerCalif4(c4)  
f. Calcular objAlumno.calcularPromedio()  
    objAlumno.calcularObservacion()  
g. Imprimir objAlumno.obtenerNombreAlum()  
    objAlumno.obtenerPromedio()  
    objAlumno.obtenerObservacion()  
h. Fin Método principal  
Fin Clase EjecutaAlumno4  
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Alumno4.java y EjecutaAlumno4.java



*Explicación:*

El algoritmo tiene dos clases: la Clase Alumno4 y la clase EjecutaAlumno4

En la Clase Alumno4:

1. Se declaran los datos que representan la estructura de la clase:  
nombreAlum para el nombre del alumno  
calif1 para la calificación 1 del alumno  
calif2 para la calificación 2 del alumno  
calif3 para la calificación 3 del alumno  
calif4 para la calificación 4 del alumno  
promedio para el promedio del alumno  
observación en el que se colocará Aprobado o Reprobado
2. Método establecerNombreAlum(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombreAlum.
3. Método establecerCalif1(ca1: Real)  
Recibe en el parámetro ca1 el valor que luego coloca en el dato calif1
4. Método establecerCalif2(ca2: Real)  
Recibe en el parámetro ca2 el valor que luego coloca en el dato calif2

5. Método establecerCalif3(ca3: Real)  
Recibe en el parámetro ca3 el valor que luego coloca en el dato calif3
  6. Método establecerCalif4(ca4: Real)  
Recibe en el parámetro ca4 el valor que luego coloca en el dato calif4
  7. Método calcularPromedio()  
Calcula el promedio del alumno
  8. Método calcularObservacion()
    - a. Si promedio >= 60 Entonces
      1. Coloca "Aprobado" en observacion
    - b. Si no
      1. Coloca "Reprobado" en observacion
    - c. Fin del if
    - d. Fin del Método
  9. Método obtenerNombreAlum() Cadena  
Retorna nombreAlum (nombre del alumno)
  10. Método obtenerPromedio() Real  
Retorna promedio (promedio del alumno)
  11. Método obtenerObservacion() Cadena  
Retorna observación
- Fin de la Clase Alumno4

En la Clase EjecutaAlumno4, en el Método principal():

- a. Se declara:
  - La variable nombre para leer el nombre del alumno
  - La variable c1 para leer la calificación 1 del alumno
  - La variable c2 para leer la calificación 2 del alumno
  - La variable c3 para leer la calificación 3 del alumno
  - La variable c4 para leer la calificación 4 del alumno
- b. Se declara el objeto objAlumno, usando como base a la clase Alumno4; dicho objeto se crea e inicializa mediante el constructor por defecto Alumno4()
- c. Se solicitan Nombre del alumno, calificación 1, calificación 2, calificación 3 y calificación 4
- d. Se leen en nombre, c1, c2, c3, c4
- e. Se llama al método establecerNombreAlum(nombre) del objeto objAlumno, para colocar el valor de nombre en el dato nombreAlum  
Se llama al método establecerCalif1(c1) del objeto objAlumno, para colocar el valor de c1 en el dato calif1  
Se llama al método establecerCalif2(c2) del objeto objAlumno, para colocar el valor de c2 en el dato calif2

Se llama al método establecerCalif3(c3) del objeto objAlumno, para colocar el valor de c3 en el dato calif3

Se llama al método establecerCalif4(c4) del objeto objAlumno, para colocar el valor de c4 en el dato calif4

- f. Se llama al método calcularPromedio() del objeto objAlumno, para calcular el promedio
- g. Se llama al método calcularObservacion() del objeto objAlumno, para calcular la observacion
- h. Se llama al método obtenerNombreAlum() del objeto objAlumno, para acceder e imprimir el valor del dato nombreAlum
- i. Se llama al método obtenerPromedio() del objeto objAlumno, para acceder e imprimir el valor del dato promedio
- j. Se llama al método obtenerObservacion() del objeto objAlumno, para acceder e imprimir el valor del dato observacion

h. Fin del Método

Fin de la Clase EjecutaAlumno4

Fin del algoritmo

#### Ejercicio 12.4.2

Elaborar un algoritmo similar al anterior de CALCULAR SUELDO DOBLE DE UN EMPLEADO, pero ahora tomando en cuenta que se tiene otra alternativa: las horas que exceden de 50, se pagan al triple de la cuota por hora. Usando if-then-else.

A continuación se presenta el algoritmo:

(Primero hágalo usted, después compare la solución.)

La solución es muy similar al algoritmo CALCULAR SUELDO DOBLE DE UN EMPLEADO; el diagrama de clases es igual, la clase EjecutaEmpleado7 es igual, en la clase Empleado7, es donde cambia, el método calcularSueldo() queda así:

```
5. Método calcularSueldo()  
    a. if horasTrab <= 40 then  
        1. sueldo = horasTrab * cuotaHora  
    b. else  
        1. if horasTrab <= 50 then  
            a. sueldo=(40*cotaHora)+  
                ((horasTrab-40)*(cotaHora*2))  
        2. else  
            a. sueldo=(40*cotaHora)+(10*cotaHora*2)  
                + ((horasTrab-50)*(cotaHora*3))  
        3. endif  
    c. endif  
    d. Fin Método calcularSueldo
```

*Explicación:*

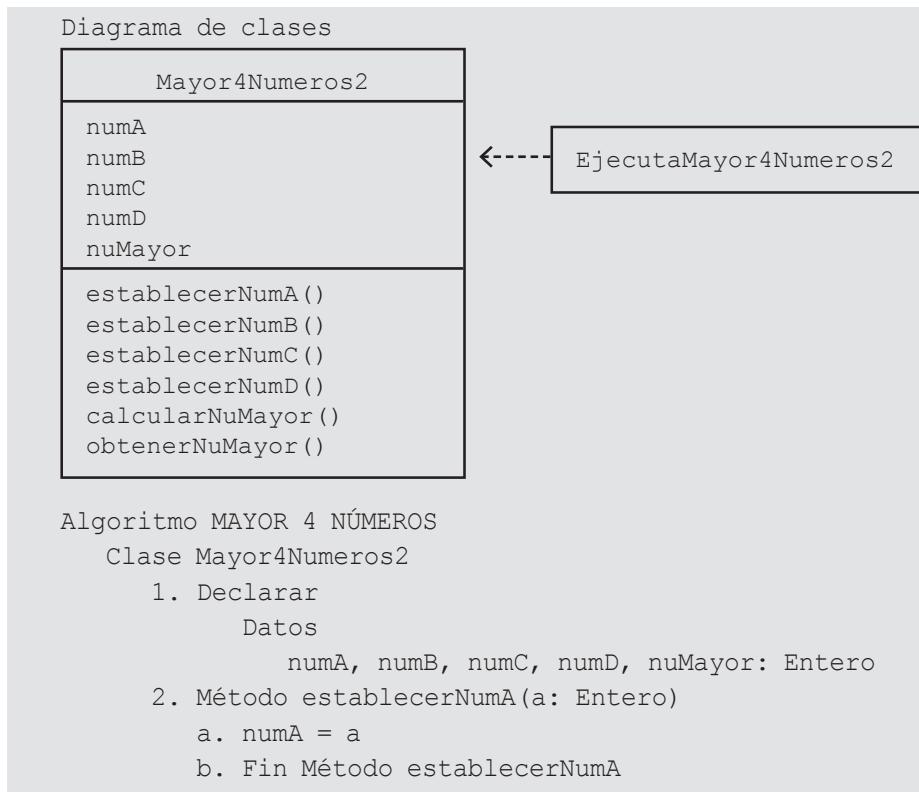
5. Método calcularSueldo()
  - a. Si horasTrab <= 40 Entonces
    1. Calcula el sueldo de la forma simple
  - b. Si no
    1. Si horasTrab <= 50 Entonces
      - a. Calcula el sueldo de la forma doble
    2. Si no
      - a. Calcula el sueldo de la forma triple
    3. Fin del if
  - c. Fin del if
  - d. Fin Método

**Ejercicio 12.4.3**

Elaborar un algoritmo que lea cuatro números y calcule e imprima el mayor. Se supone que son números diferentes. Utilizar if-then-else y AND.

A continuación se presenta el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*



```
3. Método establecerNumB(b: Entero)
   a. numB = b
   b. Fin Método establecerNumB
4. Método establecerNumC(c: Entero)
   a. numC = c
   b. Fin Método establecerNumC
5. Método establecerNumD(d: Entero)
   a. numD = d
   b. Fin Método establecerNumD
6. Método calcularNuMayor()
   a. if (numA > numB) AND (numA > numC)
      AND (numA > numD) then
      1. nuMayor = numA
   b. else
      1. if (numB > numC) AND (numB > numD) then
         a. nuMayor = numb
      2. else
         a. if numC > numD then
            1. nuMayor = numC
         b. else
            1. nuMayor = numD
         c. endif
      3. endif
   c. endif
   d. Fin Método calcularNuMayor
7. Método obtenerNuMayor() Entero
   a. return nuMayor
   b. Fin Método obtenerNuMayor
Fin Clase Mayor4Numeros2
Clase EjecutaMayor4Numeros2
1. Método principal()
   a. Declarar
      Variables
      n1, n2, n3, n4: Entero
   b. Declarar, crear e iniciar objeto
      Mayor4Numeros2 objMayor4Numeros =
      new Mayor4Numeros2()
   c. Solicitar Número 1, Número 2, Número 3 y
      Número 4
   d. Leer n1, n2, n3, n4
   e. Establecer objMayor4Numeros.establecerNumA(n1)
      objMayor4Numeros.establecerNumB(n2)
      objMayor4Numeros.establecerNumC(n3)
      objMayor4Numeros.establecerNumD(n4)
```

```
f. Calcular objMayor4Numeros.calcularNuMayor()  
g. Imprimir objMayor4Numeros.obtenerNuMayor()  
h. Fin Método principal
```

```
Fin Clase EjecutaMayor4Numeros2
```

```
Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Mayor4Numeros2.java y EjecutaMayor4Numeros2.java

*Explicación:*

El algoritmo tiene dos clases: la Clase Mayor4Numeros2 y la clase EjecutaMayor4Numeros2

En la Clase Mayor4Numeros2:

1. Se declaran los datos que representan la estructura de la clase:  
numA, numB, numC, numD para cada uno de los 4 números  
nuMayor para número mayor
2. Método establecerNumA(a: Entero)  
Recibe en el parámetro a el valor que luego coloca en el dato numA
3. Método establecerNumB(b: Entero)  
Recibe en el parámetro b el valor que luego coloca en el dato numB
4. Método establecerNumC(c: Entero)  
Recibe en el parámetro c el valor que luego coloca en el dato numC
5. Método establecerNumD(d: Entero)  
Recibe en el parámetro d el valor que luego coloca en el dato numD
6. Método calcularNuMayor()  
Calcula el mayor en nuMayor
  - a. Si (numA > numB) y (numA > numC) y (numA > numD) Entonces
    1. Coloca numB en nuMayor
  - b. Si no
    1. Si (numB > numC) y (numB > numD) Entonces
      - a. Coloca numB en nuMayor
    2. Si no
      - a. Si numC > numD Entonces
        1. Coloca numC en nuMayor
      - b. Si no
        1. Coloca numD en nuMayor

- c. Fin del if
  - d. Fin Método
7. Método obtenerNuMayor() Entero
- a. Retorna nuMayor
- Fin de la Clase Mayor4Numeros2

En la Clase EjecutaMayor4Numeros2, en el Método principal():

- a. Se declara:  
Las variables n1, n2, n3, n4 para leer los 4 números
- b. Se declara el objeto objMayor4Numeros, usando como base a la clase Mayor4Numeros2; dicho objeto se crea e inicializa mediante el constructor por defecto Mayor4Numeros2()
- c. Se solicitan Número 1, Número 2, Número 3 y Número 4
- d. Se leen en n1, n2, n3, n4
- e. Se llama al método establecerNumA(n1) del objeto objMayor4Numeros, para colocar el valor de n1 en el dato numA  
Se llama al método establecerNumB(n2) del objeto objMayor4Numeros, para colocar el valor de n2 en el dato numB  
Se llama al método establecerNumC(n3) del objeto objMayor4Numeros, para colocar el valor de n3 en el dato numC  
Se llama al método establecerNumD(n4) del objeto objMayor4Numeros, para colocar el valor de n4 en el dato numD
- f. Se llama al método calcularNuMayor() del objeto objMayor4Numeros, para calcular el mayor
- g. Se llama al método obtenerNuMayor() del objeto objMayor4Numeros, para acceder e imprimir el valor del dato nuMayor
- h. Fin del Método

Fin de la Clase EjecutaMayor4Numeros2

Fin del algoritmo

La tabla 12.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 12 de la Web del libro.

**Tabla 12.1**

| Ejercicio        | Descripción                                   |
|------------------|-----------------------------------------------|
| Ejercicio 12.4.4 | Obtiene el mayor de 5 números                 |
| Ejercicio 12.4.5 | Imprime el tipo de ángulo que es              |
| Ejercicio 12.4.6 | Realiza cálculos con la segunda ley de Newton |
| Ejercicio 12.4.7 | Realiza cálculos de un cliente                |



## 12.5 Ejercicios propuestos

Como ejercicios propuestos para este capítulo, se recomiendan, del capítulo 4, algunos de los ejercicios resueltos que no fueron incluidos en este capítulo; además todos los ejercicios propuestos en dicho capítulo.

## 12.6 Resumen de conceptos que debe dominar

- Cómo diseñar algoritmos orientados a objetos aplicando las estructuras de selección: if then else, if then y switch.

## 12.7 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

### 12.7.1 Resumen gráfico del capítulo

### 12.7.2 Autoevaluación

### 12.7.3 Programas

### 12.7.4 Ejercicios resueltos

### 12.7.5 Power Point para el profesor (\*)

# 13

## Programación orientada a objetos aplicando las estructuras de repetición

### Contenido

- 13.1 Diseño de algoritmos OO usando la repetición do...while
- 13.2 Contadores y acumuladores
  - 13.2.1 Ejercicios resueltos para do...while
- 13.3 Ejercicios propuestos para do...while
- 13.4 Diseño de algoritmos OO usando la repetición for
  - 13.4.1 Ejercicios resueltos para for
- 13.5 Ejercicios propuestos para for
- 13.6 Diseño de algoritmos OO usando la repetición while
  - 13.6.1 Ejercicios resueltos para while
- 13.7 Ejercicios propuestos para while
- 13.8 Resumen de conceptos que debe dominar
- 13.9 Contenido de la página Web de apoyo  
El material marcado con asterisco (\*) sólo está disponible para docentes.
  - 13.9.1 Resumen gráfico del capítulo
  - 13.9.2 Autoevaluación
  - 13.9.3 Programas
  - 13.9.4 Ejercicios resueltos
  - 13.9.5 Power Point para el profesor (\*)

### Objetivo del capítulo

- Aplicar las estructuras de repetición: do...while, for y while; en el diseño de algoritmos orientados a objetos.

## Introducción

Con el estudio del capítulo anterior, usted ya domina los conceptos básicos de la programación orientada a objetos aplicando las estructuras de selección y sabe cómo diseñar algoritmos usando dichas estructuras.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos mediante la programación orientada a objetos aplicando las estructuras de repetición.

Se explica cómo diseñar algoritmos orientados a objetos usando las estructuras de control de repetición que estudiamos en el capítulo 5, pero ahora inmersas en la arquitectura orientada a objetos, que se estudió en los capítulos 10 y 11.

Recordemos que la repetición do...while sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición; en un rango de 1 a n veces. La repetición for sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán un número de veces conocido de antemano, y la repetición while sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición; en un rango de cero a n veces.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejerzte estudiando los problemas planteados en los ejercicios resueltos y propuestos; al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro; analice las diferencias y vea sus errores, al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal, usted debe ir aprendiendo a analizar las diferencias y a comprender que, a veces, aunque haya diferencias, las dos soluciones son correctas.

Este libro termina en este capítulo, por lo tanto, es pertinente aclarar que el tema orientado a objetos, tiene otros conceptos que, por el enfoque y nivel de este libro, no se han tratado, como son la aplicación de arreglos en la arquitectura orientada a objetos, herencia y polimorfismo. Si le interesa profundizar más en el tema orientado a objetos, se le recomienda la lectura del libro Metodología de la programación orientada a objetos, publicado por esta misma editorial y de este mismo autor.

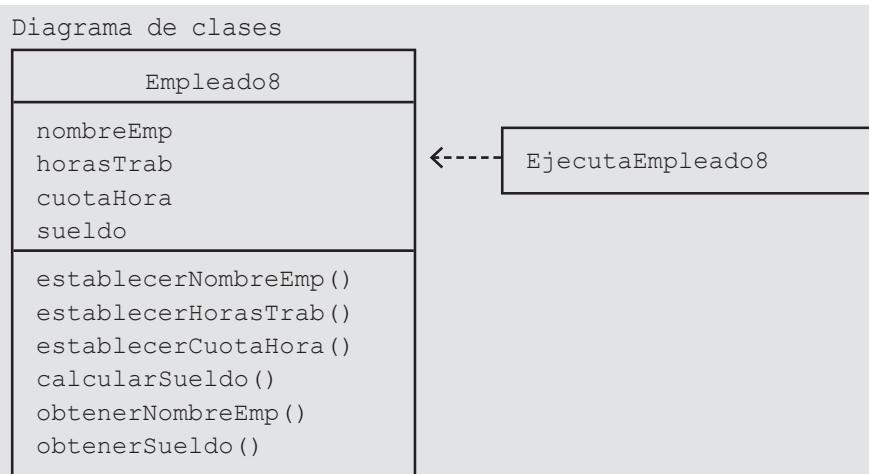
### 13.1 Diseño de algoritmos OO usando la repetición do...while

En este punto se utilizará la estructura de repetición do...while en seudocódigo, misma que se estudió en el punto 5.1 del capítulo 5, pero ahora aplicada conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejemplo:

Elaborar un algoritmo que calcule e imprima el sueldo de varios empleados, cada empleado se tratará en forma similar al primer problema que planteamos en el capítulo 3 cuando estudiamos la secuenciación.

A continuación se presenta el algoritmo de la solución:



Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado8

1. Declarar
  - Datos
    - nombreEmp: Cadena
    - horasTrab: Entero
    - cuotaHora: Real
    - sueldo: Real
2. Método establecerNombreEmp (nom: Cadena)
  - a. nombreEmp = nom
  - b. Fin Método establecerNombreEmp
3. Método establecerHorasTrab (horasTr: Entero)
  - a. horasTrab = horasTr
  - b. Fin Método establecerHorasTrab

```
4. Método establecerCuotaHora(cuotaHr: Real)
   a. cuotaHora = cuotaHr
   b. Fin Método establecerCuotaHora
5. Método calcularSueldo()
   a. sueldo = horasTrab * cuotaHora
   b. Fin Método calcularSueldo
6. Método obtenerNombreEmp() Cadena
   a. return nombreEmp
   b. Fin Método obtenerNombreEmp
7. Método obtenerSueldo() Real
   a. return sueldo
   b. Fin Método obtenerSueldo
Fin Clase Empleado8
Clase EjecutaEmpleado8
1. Método principal()
   a. Declarar
      Variables
      nomEmp: Cadena
      hrsTra: Entero
      cuoHr: Real
      desea: Carácter
   b. do
      1. Declarar, crear e iniciar objeto
         Empleado8 objEmpleado = new Empleado8()
      2. Solicitar Nombre, número de horas
         trabajadas y cuota por hora
      3. Leer nomEmp, hrsTra, cuoHr
      4. Establecer
         objEmpleado.establecerNombreEmp(nomEmp)
         objEmpleado.establecerHorasTrab(hrsTra)
         objEmpleado.establecerCuotaHora(cuoHr)
      5. Calcular objEmpleado.calcularSueldo()
      6. Imprimir objEmpleado.obtenerNombreEmp()
         objEmpleado.obtenerSueldo()
      7. Preguntar "¿Desea procesar otro
         empleado(S/N)?""
      8. Leer desea
   c. while desea == 'S'
   d. Fin Método principal
Fin Clase EjecutaEmpleado8
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Empleado8.java y EjecutaEmpleado8.java



*Explicación:*

El algoritmo tiene dos clases; la Clase Empleado8 y la clase EjecutaEmpleado8.

En la Clase Empleado8:

1. Se declaran los datos que representan la estructura de la clase:  
nombreEmp para el nombre del empleado  
horasTrab para las horas trabajadas del empleado  
cuotaHora para la cuota por hora  
sueldo para el sueldo del empleado
2. Método establecerNombreEmp(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp
3. Método establecerHorasTrab(horasTr: Entero)  
Recibe en el parámetro horasTr el valor que luego coloca en el dato horas-Trab
4. Método establecerCuotaHora(cuotaHr: Real)  
Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuota-Hora
5. Método calcularSueldo()  
Calcula el sueldo del empleado
6. Método obtenerNombreEmp() Cadena  
Retorna nombreEmp (nombre del empleado)
7. Método obtenerSueldo() Real  
Retorna sueldo

Fin de la Clase Empleado8

En la Clase EjecutaEmpleado8, en el Método principal():

- a. Se declara:  
La variable nomEmp para leer el nombre del empleado  
La variable hrsTra para leer las horas trabajadas  
La variable cuoHr para leer la cuota por hora  
La variable desea para controlar el ciclo repetitivo
- b. Inicia ciclo do
  1. Se declara el objeto objEmpleado, usando como base a la clase Empleado8; dicho objeto se crea e inicializa mediante el constructor por defecto Empleado8()  
Observe que cada vez que entra al ciclo crea un nuevo objeto empleado

2. Se Solicitan el Nombre, número de horas trabajadas y cuota por hora
  3. Se leen en nomEmp, hrsTra, cuoHr
  4. Se llama al método establecerNombreEmp(nomEmp) del objeto objEmpleado, para colocar el valor de nomEmp en el dato nombreEmp  
Se llama al método establecerHorasTrab(hrsTra) del objeto objEmpleado, para colocar el valor de hrsTra en el dato horasTrab  
Se llama al método establecerCuotaHora(cuoHr) del objeto objEmpleado, para colocar el valor de cuoHr en el dato cuotaHora
  5. Se llama al método calcularSueldo() del objeto objEmpleado, para calcular el sueldo
  6. Se llama al método obtenerNombreEmp() del objeto objEmpleado, para acceder e imprimir el valor del dato nombreEmp  
Se llama al método obtenerSueldo() del objeto objEmpleado, para acceder e imprimir el valor del dato sueldo
  7. Se pregunta “¿Desea procesar otro empleado(S/N)?”
  8. Se lee en desea la respuesta
- c. Fin del ciclo (while desea == ‘S’). Si se cumple regresa al do; si no, se sale del ciclo
- d. Fin del Método
- Fin de la Clase EjecutaEmpleado8
- Fin del algoritmo

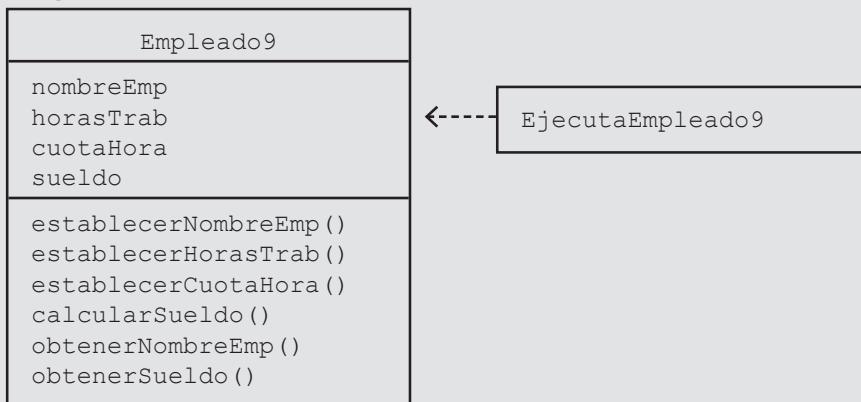
## 13.2 Contadores y acumuladores

Utilizando el concepto de contadores y acumuladores, en el problema anterior, ahora se requiere elaborar un algoritmo que permita procesar los empleados e imprima un reporte que tenga el siguiente formato:

| REPORTE DE EMPLEADOS   |            |
|------------------------|------------|
| NOMBRE                 | SUELDO     |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| ---                    | ---        |
| ---                    | ---        |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| TOTAL 999 EMPLEADOS    | 999,999.99 |

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado9

1. Declarar
    - Datos
 

```

nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real
          
```
  2. Método establecerNombreEmp(nom: Cadena)
    - a. nombreEmp = nom
    - b. Fin Método establecerNombreEmp
  3. Método establecerHorasTrab(horasTr: Entero)
    - a. horasTrab = horasTr
    - b. Fin Método establecerHorasTrab
  4. Método establecerCuotaHora(cuotaHr: Real)
    - a. cuotaHora = cuotaHr
    - b. Fin Método establecerCuotaHora
  5. Método calcularSueldo()
    - a. sueldo = horasTrab \* cuotaHora
    - b. Fin Método calcularSueldo
  6. Método obtenerNombreEmp() Cadena
    - a. return nombreEmp
    - b. Fin Método obtenerNombreEmp
  7. Método obtenerSueldo() Real
    - a. return sueldo
    - b. Fin Método obtenerSueldo
- Fin Clase Empleado9

```

Clase EjecutaEmpleado9
    1. Método principal()
        a. Declarar
            Variables
                nomEmp: Cadena
                hrsTra, totEmpleados: Entero
                cuoHr, totSueldos: Real
                desea: Carácter
        b. Imprimir encabezado
        c. totEmpleados = 0
            totSueldos = 0
        d. do
            1. Declarar, crear e iniciar objeto
                Empleado9 objEmpleado = new Empleado9()
            2. Solicitar Nombre, número de horas
                trabajadas y cuota por hora
            3. Leer nomEmp, hrsTra, cuoHr
            4. Establecer
                objEmpleado.establecerNombreEmp(nomEmp)
                objEmpleado.establecerHorasTrab(hrsTra)
                objEmpleado.establecerCuotaHora(cuoHr)
            5. Calcular objEmpleado.calcularSueldo()
            6. Imprimir objEmpleado.obtenerNombreEmp()
                objEmpleado.obtenerSueldo()
            7. totEmpleados = totEmpleados + 1
                totSueldos = totSueldos +
                objEmpleado.obtenerSueldo()
            8. Preguntar "¿Desea procesar otro
                empleado(S/N) ?"
            9. Leer desea
        e. while desea == 'S'
        f. Imprimir totEmpleados, totSueldos
        g. Fin Método principal
    Fin Clase EjecutaEmpleado9
    Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
 Programa en Java: Empleado9.java y EjecutaEmpleado9.java

*Explicación:*

El algoritmo tiene dos clases: la Clase Empleado9 y la clase EjecutaEmpleado9

En la Clase Empleado9:

1. Se declaran los datos que representan la estructura de la clase:  
nombreEmp para el nombre del empleado  
horasTrab para las horas trabajadas del empleado  
cuotaHora para la cuota por hora  
sueldo para el sueldo del empleado
  2. Método establecerNombreEmp(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp
  3. Método establecerHorasTrab(horasTr: Entero)  
Recibe en el parámetro horasTr el valor que luego coloca en el dato horas-Trab
  4. Método establecerCuotaHora(cuotaHr: Real)  
Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuota-Hora
  5. Método calcularSueldo()  
Calcula el sueldo del empleado
  6. Método obtenerNombreEmp() Cadena  
Retorna nombreEmp (nombre del empleado)
  7. Método obtenerSueldo() Real  
Retorna sueldo
- Fin de la Clase Empleado9

En la Clase EjecutaEmpleado9, en el Método principal():

- a. Se declara:  
La variable nomEmp para leer el nombre del empleado  
La variable hrsTra para leer las horas trabajadas  
La variable cuoHr para leer la cuota por hora  
La variable totEmpleados para contar el total de empleados  
La variable totSueldos para calcular el total de sueldos de todos los empleados  
La variable desea para controlar el ciclo repetitivo
- b. Imprime el encabezado
- c. Inicia totEmpleados en 0, y totSueldos en 0
- d. Inicia ciclo do
  1. Se declara el objeto objEmpleado, usando como base a la clase Empleado9; dicho objeto se crea e inicializa mediante el constructor por defecto Empleado9()  
Observe que cada vez que entra al ciclo crea un nuevo objeto empleado

2. Se Solicitan el Nombre, número de horas trabajadas y cuota por hora
  3. Se leen en nomEmp, hrsTra, cuoHr
  4. Se llama al método establecerNombreEmp(nomEmp) del objeto objEmpleado, para colocar el valor de nomEmp en el dato nombreEmp  
Se llama al método establecerHorasTrab(hrsTra) del objeto objEmpleado, para colocar el valor de hrsTra en el dato horasTrab  
Se llama al método establecerCuotaHora(cuoHr) del objeto objEmpleado, para colocar el valor de cuoHr en el dato cuotaHora
  5. Se llama al método calcularSueldo() del objeto objEmpleado, para calcular el sueldo
  6. Se llama al método obtenerNombreEmp() del objeto objEmpleado, para acceder e imprimir el valor del dato nombreEmp  
Se llama al método obtenerSueldo() del objeto objEmpleado, para acceder e imprimir el valor del dato sueldo
  7. Se incrementa totEmpleados en 1  
Se incrementa totSueldos con el sueldo del empleado; mismo que se accede llamando al método obtenerSueldo() del objeto objEmpleado
  8. Se pregunta “¿Desea procesar otro empleado(S/N)?”
  9. Se lee en desea la respuesta
- e. Fin del ciclo (while desea == ‘S’). Si se cumple regresa al do; si no, se sale del ciclo
- f. Imprime totEmpleados y totSueldos
- g. Fin del Método
- Fin de la Clase EjecutaEmpleado9
- Fin del algoritmo

### 13.2.1 Ejercicios resueltos para do...while

En este punto se presentan algunos de los ejercicios resueltos en el punto 5.1 del capítulo 5. Es decir, que el mismo problema ya lo resolvimos aplicando la programación estructurada en el capítulo 5, y aquí lo estamos resolviendo aplicando la metodología de la programación orientada a objetos.

Cabe aclarar que por la naturaleza de la repetición do...while, los problemas que se plantean en este capítulo tienen una orientación más administrativa, porque esta estructura es más natural para resolver problemas de esta índole. Sin embargo, en el punto siguiente (for), donde los problemas tienen una orientación más hacia ingeniería, utilizaremos el do...while para resolver ese tipo de problemas.

### Ejercicio 13.2.1.1

Una empresa vende hojas de hielo seco, con las condiciones siguientes:

- Si el cliente es tipo 1 se le descuenta el 5%
- Si el cliente es tipo 2 se le descuenta el 8%
- Si el cliente es tipo 3 se le descuenta el 12%
- Si el cliente es tipo 4 se le descuenta el 15%

Cuando un cliente realiza una compra se generan los datos siguientes:

- Nombre del cliente
- Tipo de cliente (1,2,3,4)
- Cantidad de hojas compradas
- Precio por hoja

Elaborar un algoritmo que permita procesar varios clientes e imprima el reporte:

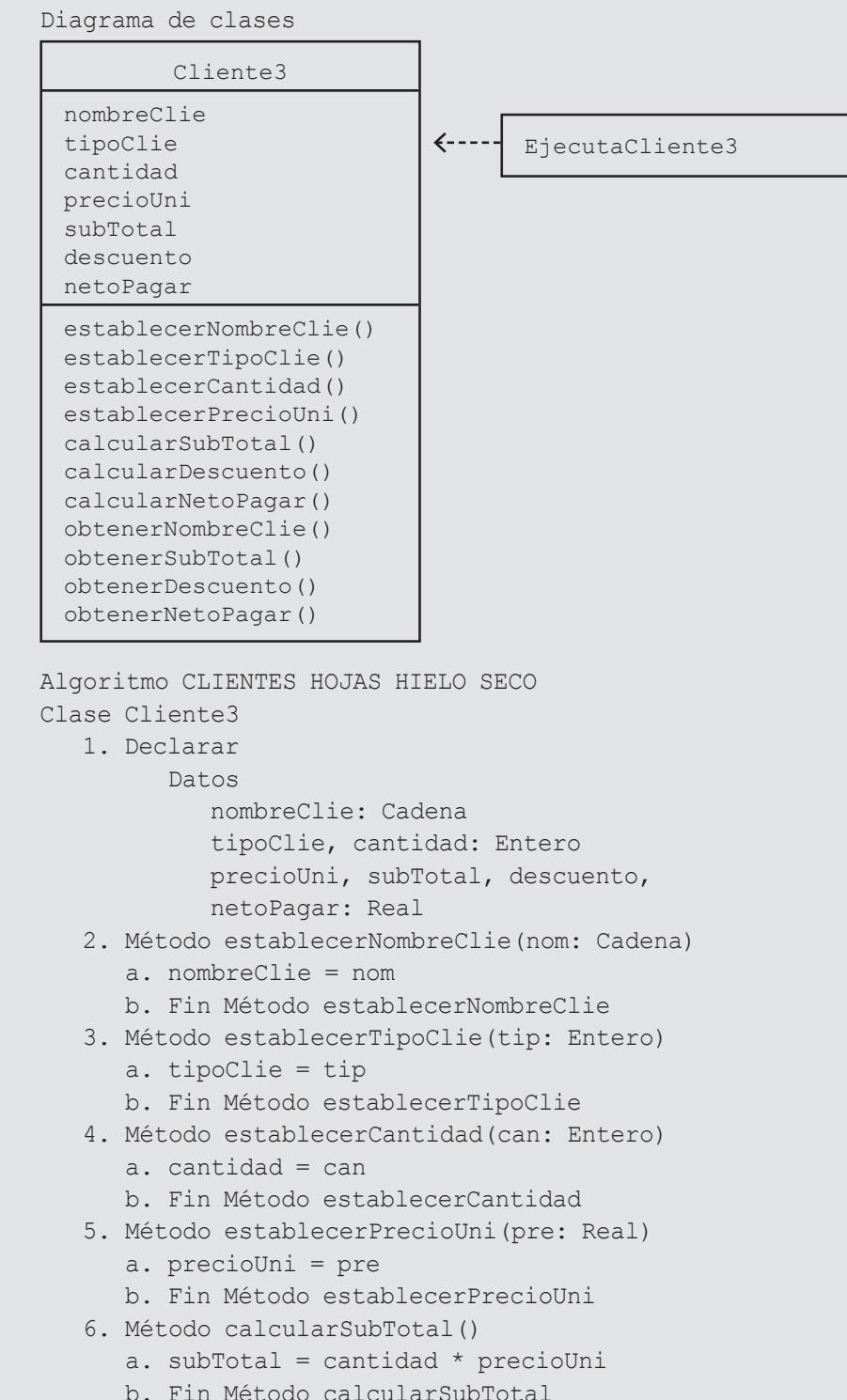
| REPORTE DE CLIENTES  |            |            |              |
|----------------------|------------|------------|--------------|
| NOMBRE               | SUB.TOTAL  | DESCUENTO  | NETO A PAGAR |
| XXXXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99    |
| XXXXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99    |
| - - -                | - - -      | - - -      | - - -        |
| - - -                | - - -      | - - -      | - - -        |
| XXXXXXXXXXXXXXXXXXXX | 99,999.99  | 99,999.99  | 99,999.99    |
| TOTAL 999 CLIENTES   | 999,999.99 | 999,999.99 | 999,999.99   |

Cálculos:

- Subtotal = Cantidad de hojas X Precio por hoja
- Descuento es el porcentaje correspondiente del Subtotal a pagar
- Neto a pagar = Sub total – Descuento

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted, después compare la solución.)



```
7. Método calcularDescuento()
    a. switch tipoClie
        1: descuento = subtotal * 0.05
        2: descuento = subtotal * 0.08
        3: descuento = subtotal * 0.12
        4: descuento = subtotal * 0.15
    b. endswitch
    c. Fin Método calcularDescuento
8. Método calcularNetoPagar()
    a. netoPagar = subtotal - descuento
    b. Fin Método calcularNetoPagar
9. Método obtenerNombreClie() Cadena
    a. return nombreClie
    b. Fin Método obtenerNombreClie
10. Método obtenerSubTotal() Real
    a. return subtotal
    b. Fin Método obtenerSubTotal
11. Método obtenerDescuento() Real
    a. return descuento
    b. Fin Método obtenerDescuento
12. Método obtenerNetoPagar() Real
    a. return netoPagar
    b. Fin Método obtenerNetoPagar
Fin Clase Cliente3
Clase EjecutaCliente3
1. Método principal()
    a. Declarar
        Variables
            nomCli: Cadena
            tiCli, cant, totClientes: Entero
            preUni, totSubTot, totDescuento,
            totNeto: Real,
            desea: Carácter
    b. Imprimir encabezado
    c. totClientes = 0
        totSubTot = 0
        totDescuento = 0
        totNeto = 0
    d. do
        1. Declarar, crear e iniciar objeto
            Cliente3 objCliente = new Cliente3()
        2. Solicitar Nombre, Tipo cliente,
            Cantidad, Precio unitario
```

```
3. Leer nomCli, tiCli, cant, preUni
4. Establecer
    objCliente.establecerNombreClie(nomCli)
    objCliente.establecerTipoClie(tiCli)
    objCliente.establecerCantidad(cant)
    objCliente.establecerPrecioUni(preUni)
5. Calcular objCliente.calcularSubTotal()
    objCliente.calcularDescuento()
    objCliente.calcularNetoPagar()
6. Imprimir objCliente.obtenerNombreClie()
    objCliente.obtenerSubTotal()
    objCliente.obtenerDescuento()
    objCliente.obtenerNetoPagar()
7. totClientes = totClientes + 1
    totSubTot = totSubTot +
        objCliente.obtenerSubTotal()
    totDescuento = totDescuento +
        objCliente.obtenerDescuento()
    totNeto = totNeto +
        objCliente.obtenerNetoPagar()
8. Preguntar "¿Desea procesar otro
    cliente (S/N) ?"
9. Leer desea
e. while desea == 'S'
f. Imprimir totClientes, totSubTot,
    totDescuento, totNeto
g. Fin Método principal
Fin Clase EjecutaCliente3
Fin
```



En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Cliente3.java y EjecutaCliente3.java

#### *Explicación:*

El algoritmo tiene dos clases: la Clase Cliente3 y la clase EjecutaCliente3

En la Clase Cliente3:

1. Se declaran los datos que representan la estructura de la clase:  
nombreClie para el nombre del cliente  
tipoClie para el tipo de cliente  
cantidad para la cantidad de hojas compradas

- precioUni para el precio unitario
  - subTotal para el subtotal
  - descuento para el descuento
  - netoPagar para el neto a pagar
2. Método establecerNombreClie(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombreClie
  3. Método establecerTipoClie(tipo: Entero)  
Recibe en el parámetro tipo el valor que luego coloca en el dato tipoClie
  4. Método establecerCantidad(can: Entero)  
Recibe en el parámetro can el valor que luego coloca en el dato cantidad
  5. Método establecerPrecioUni(pre: Real)  
Recibe en el parámetro pre el valor que luego coloca en el dato precioUni
  6. Método calcularSubTotal()  
Calcula el subTotal a pagar
  7. Método calcularDescuento()  
Calcula el descuento
    - a. switch tipoClie
      - Si tipoClie==1: descuento = subTotal \* 0.05
      - Si tipoClie==2: descuento = subTotal \* 0.08
      - Si tipoClie==3: descuento = subTotal \* 0.12
      - Si tipoClie==4: descuento = subTotal \* 0.15
    - b. endswitch
    - c. Fin Método
  8. Método calcularNetoPagar()  
Calcula el netoPagar a pagar
  9. Método obtenerNombreClie() Cadena  
Retorna nombreClie
  10. Método obtenerSubTotal() Real  
Retorna subTotal
  11. Método obtenerDescuento() Real  
Retorna descuento
  12. Método obtenerNetoPagar() Real  
Retorna netoPagar
- Fin de la Clase Cliente3

En la Clase EjecutaCliente3, en el Método principal():

- a. Se declara:

La variable nomCli para leer el nombre del cliente  
La variable tiCli para leer el tipo de cliente  
La variable cant para leer la cantidad de hojas compradas  
La variable preUni para leer el precio unitario por hoja  
La variable totClientes para contar el total de clientes  
La variable totSubTot calcular el total de los subtotales  
La variable totDescuento calcular el total de los descuentos  
La variable totNeto calcular el total de los netos a pagar  
La variable desea para controlar el ciclo repetitivo

- b. Imprime el encabezado

- c. Inicia los totalizadores en 0

- d. Inicia ciclo do

1. Se declara el objeto objCliente, usando como base a la clase Cliente3; dicho objeto se crea e inicializa mediante el constructor por defecto Cliente3()  
Observe que cada vez que entra al ciclo crea un nuevo objeto cliente
2. Solicita Nombre, Tipo cliente, Cantidad, Precio unitario
3. Lee en nomCli, tiCli, cant, preUni
4. Se llama al método establecerNombreClie(nomCli) del objeto objCliente, para colocar el valor de nomCli en el dato nombreClie  
Se llama al método establecerTipoClie(tiCli) del objeto objCliente, para colocar el valor de tiCli en el dato tipoClie  
Se llama al método establecerCantidad(cant) del objeto objCliente, para colocar el valor de cant en el dato cantidad  
Se llama al método establecerPrecioUni(preUni) del objeto objCliente, para colocar el valor de preUni en el dato precioUni
5. Se llama al método calcularSubTotal() del objeto objCliente, para calcular el subtotal  
Se llama al método calcularDescuento() del objeto objCliente, para calcular el descuento  
Se llama al método calcularNetoPagar() del objeto objCliente, para calcular el neto a pagar
6. Se llama al método obtenerNombreClie() del objeto objCliente, para acceder e imprimir el valor del dato nombreClie  
Se llama al método obtenerSubTotal() del objeto objCliente, para acceder e imprimir el valor del dato subTotal  
Se llama al método obtenerDescuento() del objeto objCliente, para acceder e imprimir el valor del dato descuento

Se llama al método obtenerNetoPagar() del objeto objCliente, para acceder e imprimir el valor del dato netoPagar

7. Incrementa totClientes en 1

Se incrementa totSubTot con el subTotal del cliente; mismo que se accede llamando al método obtenerSubTotal() del objeto objCliente

Se incrementa totDescuento con el descuento del cliente; mismo que se accede llamando al método obtenerDescuento() del objeto objCliente

Se incrementa totNeto con el netoPagar del cliente; mismo que se accede llamando al método obtenerNetoPagar() del objeto objCliente

8. Pregunta “¿Desea procesar otro cliente(S/N)?”

9. Se lee en deseja la respuesta

e. Fin del ciclo (while deseja == ‘S’). Si se cumple regresa al do; si no, se sale del ciclo.

f. Imprime totClientes, totSubTot, totDescuento, totNeto

g. Fin Método

Fin de la Clase EjecutaCliente3

Fin del algoritmo

### Ejercicio 13.2.1.2

Elaborar un algoritmo que proporcione el siguiente reporte:

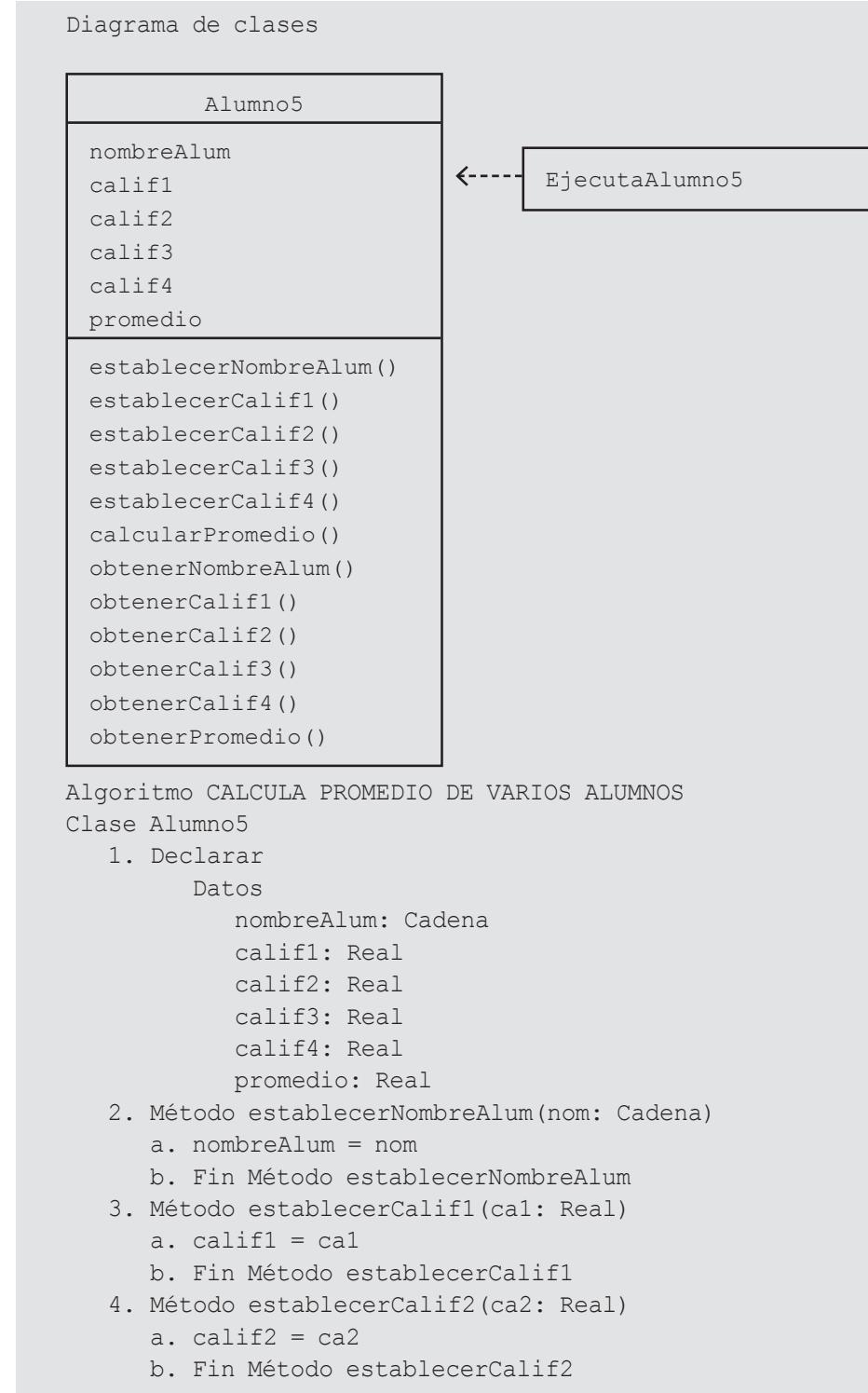
| NOMBRE                    | ANÁLISIS DE CALIFICACIONES |       |       |       | PROMEDIO |
|---------------------------|----------------------------|-------|-------|-------|----------|
|                           | CAL.1                      | CAL.2 | CAL.3 | CAL.4 |          |
| XXXXXXXXXXXXXXXXXXXX      | 99.99                      | 99.99 | 99.99 | 99.99 | 99.99    |
| XXXXXXXXXXXXXXXXXXXX      | 99.99                      | 99.99 | 99.99 | 99.99 | 99.99    |
| - - -                     | - - -                      | - - - | - - - | - - - | - - -    |
| XXXXXXXXXXXXXXXXXXXX      | 99.99                      | 99.99 | 99.99 | 99.99 |          |
| 99.99 PROMEDIOS GENERALES | 99.99                      | 99.99 | 99.99 | 99.99 | 99.99    |

A partir de que se tiene el NOMBRE, calificación 1,2,3 y 4 de varios alumnos. El promedio se obtiene sumando las cuatro calificaciones y dividiendo el resultado entre cuatro.

El promedio general de cada calificación se calcula sumando las calificaciones de todos los alumnos y dividiendo el resultado entre el número de alumnos.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted, después compare la solución.)



```
5. Método establecerCalif3(ca3: Real)
   a. calif3 = ca3
   b. Fin Método establecerCalif3
6. Método establecerCalif4(ca4: Real)
   a. calif4 = ca4
   b. Fin Método establecerCalif4
7. Método calcularPromedio()
   a. promedio = (calif1+ calif2+ calif3+ calif4)/4
   b. Fin Método calcularPromedio
8. Método obtenerNombreAlum() Cadena
   a. return nombreAlum
   b. Fin Método obtenerNombreAlum
9. Método obtenerCalif1() Real
   a. return calif1
   b. Fin Método obtenerCalif1
10. Método obtenerCalif2() Real
    a. return calif2
    b. Fin Método obtenerCalif2
11. Método obtenerCalif3() Real
    a. return calif3
    b. Fin Método obtenerCalif3
12. Método obtenerCalif4() Real
    a. return calif4
    b. Fin Método obtenerCalif4
13. Método obtenerPromedio() Real
    a. return promedio
    b. Fin Método obtenerPromedio
Fin Clase Alumno5
Clase EjecutaAlumno5
1. Método principal()
   a. Declarar
      Variables
         nombre: Cadena
         totAlumnos: Entero
         c1, c2, c3, c4, promCall1, promCal2,
         promCal3, promCal4, promProm, totCall1,
         totCal2, totCal3, totCal4, totProm: Real
         desea: Carácter
   b. Imprimir encabezado
   c. totAlumnos = 0
      totCall1 = 0, totCal2 = 0, totCal3 = 0,
      totCal4 = 0, totProm = 0
   d. do
```

```
1. Declarar, crear e iniciar objeto  
Alumno5 objAlumno = new Alumno5()  
2. Solicitar Nombre del alumno, calificación 1,  
calificación 2, calificación 3 y calificación 4  
3. Leer nombre, c1, c2, c3, c4  
4. Establecer  
    objAlumno.establecerNombreAlum(nombre)  
    objAlumno.establecerCalif1(c1)  
    objAlumno.establecerCalif2(c2)  
    objAlumno.establecerCalif3(c3)  
    objAlumno.establecerCalif4(c4)  
5. Calcular objAlumno.calcularPromedio()  
6. Imprimir objAlumno.obtenerNombreAlum()  
    objAlumno.obtenerCalif1()  
    objAlumno.obtenerCalif2()  
    objAlumno.obtenerCalif3()  
    objAlumno.obtenerCalif4()  
    objAlumno.obtenerPromedio()  
7. totAlumnos = totAlumnos + 1  
8. totCall1 = totCall1  
    + objAlumno.obtenerCalif1()  
totCal2 = totCal2  
    + objAlumno.obtenerCalif2()  
totCal3 = totCal3  
    + objAlumno.obtenerCalif3()  
totCal4 = totCal4  
    + objAlumno.obtenerCalif4()  
totProm = totProm  
    + objAlumno.obtenerPromedio()  
9. Preguntar "¿Desea procesar otro alumno  
(S/N)?"  
10. Leer desea  
e. while desea == 'S'  
f. promCall1 = totCall1 / totAlumnos  
    promCal2 = totCal2 / totAlumnos  
    promCal3 = totCal3 / totAlumnos  
    promCal4 = totCal4 / totAlumnos  
    promProm = totProm / totAlumnos  
g. Imprimir promCall1, promCal2, promCal3,  
    promCal4, promProm  
h. Fin Método principal  
Fin Clase EjecutaAlumno5  
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Alumno5.java y EjecutaAlumno5.java



*Explicación:*

El algoritmo tiene dos clases: la Clase Alumno5 y la clase EjecutaAlumno5

En la Clase Alumno5:

1. Se declaran los datos que representan la estructura de la clase:

nombreAlum para el nombre del alumno  
calif1 para la calificación 1 del alumno  
calif2 para la calificación 2 del alumno  
calif3 para la calificación 3 del alumno  
calif4 para la calificación 4 del alumno  
promedio para el promedio del alumno

2. Método establecerNombreAlum(nom: Cadena)

Recibe en el parámetro nom el valor que luego coloca en el dato nombreAlum

3. Método establecerCalif1(ca1: Real)

Recibe en el parámetro ca1 el valor que luego coloca en el dato calif1

4. Método establecerCalif2(ca2: Real)

Recibe en el parámetro ca2 el valor que luego coloca en el dato calif2

5. Método establecerCalif3(ca3: Real)

Recibe en el parámetro ca3 el valor que luego coloca en el dato calif3

6. Método establecerCalif4(ca4: Real)

Recibe en el parámetro ca4 el valor que luego coloca en el dato calif4

7. Método calcularPromedio()

Calcula el promedio del alumno

8. Método obtenerNombreAlum() Cadena

Retorna nombreAlum (nombre del alumno)

9. Método obtenerCalif1() Real

Retorna calif1

10. Método obtenerCalif2() Real

Retorna calif2

11. Método obtenerCalif3() Real

Retorna calif3

12. Método obtenerCalif4() Real

Retorna calif4

13. Método obtenerPromedio() Real

Retorna promedio (promedio del alumno)

Fin de la Clase Alumno5

En la Clase EjecutaAlumno5, en el Método principal():

- a. Se declara:

La variable nombre para leer el nombre del alumno  
La variable c1 para leer la calificación 1 del alumno  
La variable c2 para leer la calificación 2 del alumno  
La variable c3 para leer la calificación 3 del alumno  
La variable c4 para leer la calificación 4 del alumno  
totAlumnos para contar el total de alumnos  
promCal1, promCal2, promCal3, promCal4, promProm, para los promedios  
totCal1, totCal2, totCal3, totCal4, totProm, para los totalizadores  
La variable desea para controlar el ciclo repetitivo

- b. Imprime el encabezado  
c. Inicia los totalizadores en 0  
d. Inicia ciclo do

1. Se declara el objeto objAlumno, usando como base a la clase Alumno5;  
dicho objeto se crea e inicializa mediante el constructor por defecto  
Alumno5()

Observe que cada vez que entra al ciclo crea un nuevo objeto alumno

2. Se solicitan Nombre del alumno, calificación 1,

calificación 2, calificación 3 y calificación 4

3. Se leen en nombre, c1, c2, c3, c4

4. Se llama al método establecerNombreAlum(nombre) del objeto objAlumno,  
para colocar el valor de nombre en el dato nombreAlum

Se llama al método establecerCalif1(c1) del objeto objAlumno, para  
colocar el valor de c1 en el dato calif1

Se llama al método establecerCalif2(c2) del objeto objAlumno, para  
colocar el valor de c2 en el dato calif2

Se llama al método establecerCalif3(c3) del objeto objAlumno, para  
colocar el valor de c3 en el dato calif3

Se llama al método establecerCalif4(c4) del objeto objAlumno, para  
colocar el valor de c4 en el dato calif4

5. Se llama al método calcularPromedio() del objeto objAlumno, para cal-  
cular el promedio

6. Se llama al método obtenerNombreAlum() del objeto objAlumno, para  
acceder e imprimir el valor del dato nombreAlum

Se llama al método obtenerCalif1() del objeto objAlumno, para acceder e  
imprimir el valor del dato calif1

Se llama al método obtenerCalif2() del objeto objAlumno para acceder e  
imprimir el valor del dato calif2

Se llama al método obtenerCalif3() del objeto objAlumno, para acceder e  
imprimir el valor del dato calif3

Se llama al método obtenerCalif4() del objeto objAlumno, para acceder e  
imprimir el valor del dato calif4

Se llama al método obtenerPromedio() del objeto objAlumno, para acceder e imprimir el valor del dato promedio

7. Incrementa totAlumnos en 1
8. Se incrementa totCal1 con calif1; que se accede llamando al método obtenerCalif1() del objeto objAlumno
- Se incrementa totCal2 con calif2; que se accede llamando al método obtenerCalif2() del objeto objAlumno
- Se incrementa totCal3 con calif3; que se accede llamando al método obtenerCalif3() del objeto objAlumno
- Se incrementa totCal4 con calif4; que se accede llamando al método obtenerCalif4() del objeto objAlumno
- Se incrementa totProm con promedio; que se accede llamando al método obtenerPromedio() del objeto objAlumno
9. Pregunta “¿Desea procesar otro alumno(S/N)?”
10. Se lee en desea la respuesta

e. Fin del ciclo (while desea == 'S'). Si se cumple regresa al do; si no, se sale del ciclo

f. Calcula los promedios promCal1, promCal2, promCal3, promCal4 y promProm

g. Imprime promCal1, promCal2, promCal3, promCal4, promProm

h. Fin Método

Fin de la Clase EjecutaAlumno5

Fin del algoritmo

La tabla 13.1 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 13 de la Web del libro.

**Tabla 13.1**

| Ejercicio          | Descripción                                            |
|--------------------|--------------------------------------------------------|
| Ejercicio 13.1.2.3 | Procesa artículos con inflación                        |
| Ejercicio 13.1.2.4 | Procesa la producción de varios obreros con dos ciclos |



### 13.3 Ejercicios propuestos para do...while

Como ejercicios propuestos para este punto se recomiendan del capítulo 5, del punto de la repetición do...while (5.1), algunos de los ejercicios resueltos que no fueron incluidos en este punto; además todos los ejercicios propuestos en dicho punto.

### 13.4 Diseño de algoritmos OO usando la repetición for

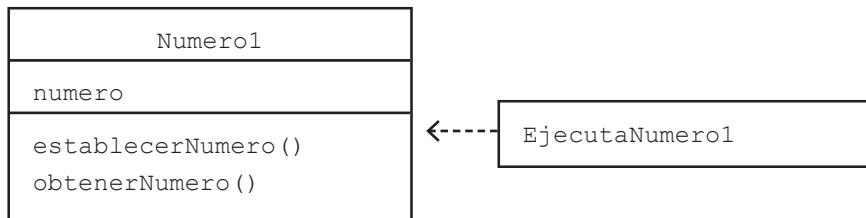
En este punto se utilizará la estructura de repetición for en seudocódigo, misma que se estudió en el punto 5.2 del capítulo 5, pero ahora aplicada conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejemplo:

Elaborar un algoritmo que lea 20 números y que calcule e imprima el promedio de dichos números.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



#### Algoritmo PROMEDIO DE 20 NÚMEROS

Clase Numero1

1. Declarar

Datos

    numero: Entero

2. Método establecerNumero(nu: Entero)

    a. numero = nu

    b. Fin Método establecerNumero

3. Método obtenerNumero()

    a. return numero

    b. Fin Método obtenerNumero

Fin Clase Numero1

Clase EjecutaNumero1

1. Método principal()

    a. Declarar

        Variables

            i, num, sumatoria: Entero

            promedio: Real

    b. sumatoria = 0

    c. for i=1; i<=20; i++

        1. Declarar, crear e iniciar objeto

            Numero1 objNumero = new Numero1()

        2. Solicitar Número

        3. Leer num

```
4. Establecer objNumero.establecerNumero (num)
5. sumatoria = sumatoria + objNumero.obtenerNumero()
d. endfor
e. promedio = sumatoria / 20
f. Imprimir promedio
g. Fin Método principal
Fin Clase EjecutaNumero1
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Numero1.java y EjecutaNumero1.java



*Explicación:*

El algoritmo tiene dos clases: la Clase Numero1 y la clase EjecutaNumero1

En la Clase Numero1:

1. Se declaran los datos que representan la estructura de la clase:  
numero para el número
  2. Método establecerNumero(nu: Entero)  
Recibe en el parámetro nu el valor que luego coloca en el dato numero
  3. Método obtenerNumero()  
Retorna numero
- Fin de la Clase Numero1

En la Clase EjecutaNumero1, en el Método principal():

- a. Se declara:  
La variable num para leer el número  
La variable i para controlar el ciclo repetitivo for  
La variable sumatoria para calcular la sumatoria de los números  
La variable promedio para calcular el promedio
- b. Inicia sumatoria en 0
- c. Inicia ciclo for desde i=1 hasta 20 con incrementos de 1
  1. Se declara el objeto objNumero, usando como base a la clase Numero1; dicho objeto se crea e inicializa mediante el constructor por defecto Numero1()  
Observe que cada vez que entra al ciclo crea un nuevo objeto numero
  2. Solicita el número
  3. Lee en num
  4. Se llama al método establecerNumero(num) del objeto objNumero, para colocar el valor de num en el dato numero
  5. Se incrementa sumatoria con numero, que se accede llamando al método obtenerNumero() del objeto objNumero

- d. Fin del ciclo for
  - e. Calcula promedio
  - f. Imprime promedio
  - g. Fin Método
- Fin de la Clase EjecutaNumero1  
Fin del algoritmo

### 13.4.1 Ejercicios resueltos para for

En este punto se presentan algunos de los ejercicios resueltos en el punto 5.2 del capítulo 5. Es decir, que el mismo problema ya lo resolvimos aplicando la programación estructurada en el capítulo 5, y aquí lo estamos resolviendo aplicando la metodología de la programación orientada a objetos.

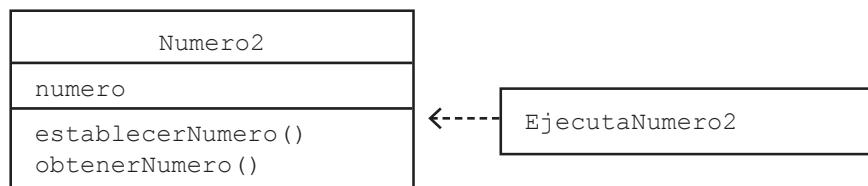
#### Ejercicio 13.4.1.1

Elaborar un algoritmo que solicite la cantidad de números a procesar y lea la respuesta en N; luego que lea los N números y calcule e imprima el promedio de dichos números.

A continuación se presenta el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*

Diagrama de clases



```

Algoritmo PROMEDIO DE N NÚMEROS
Clase Numero2
1. Declarar
    Datos
    numero: Entero
2. Método establecerNumero(nu: Entero)
    a. numero = nu
    b. Fin Método establecerNumero
3. Método obtenerNumero() Entero
    a. return numero
    b. Fin Método obtenerNumero
Fin Clase Numero2
Clase EjecutaNumero2
1. Método principal()

```

```

a. Declarar
    Variables
        i, n, num, sumatoria: Entero
        promedio: Real
    b. Solicitar cantidad de números a procesar
    c. Leer n
    d. sumatoria = 0
    e. for i=1; i<=n; i++
        1. Declarar, crear e iniciar objeto
            Numero2 objNumero = new Numero2()
        2. Solicitar Número
        3. Leer num
        4. Establecer objNumero.establecerNumero(num)
        5. sumatoria = sumatoria + objNumero.obtenerNumero()
    f. endfor
    g. promedio = sumatoria / n
    h. Imprimir promedio
    i. Fin Método principal
Fin Clase EjecutaNumero2
Fin

```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Numero2.java y EjecutaNumero2.java



*Explicación:*

El algoritmo tiene dos clases: la Clase Numero2 y la clase EjecutaNumero2

En la Clase Numero2:

1. Se declaran los datos que representan la estructura de la clase:  
numero para el número
  2. Método establecerNumero(nu: Entero)  
Recibe en el parámetro nu el valor que luego coloca en el dato numero
  3. Método obtenerNumero() Entero  
Retorna numero
- Fin de la Clase Numero2

En la Clase EjecutaNumero2, en el Método principal():

- a. Se declara:
  - La variable num para leer el número
  - La variable i para controlar el ciclo repetitivo for
  - La variable sumatoria para calcular la sumatoria de los números
  - La variable promedio para calcular el promedio
  - La variable n para leer la cantidad de números

- b. Sigue la cantidad de números a procesar
  - c. Lee en n
  - d. Inicia sumatoria en 0
  - e. Inicia ciclo for desde i=1 hasta n con incrementos de 1
    - 1. Se declara el objeto objNumero, usando como base a la clase Numero2; dicho objeto se crea e inicializa mediante el constructor por defecto Numero2()
    - Observe que cada vez que entra al ciclo crea un nuevo objeto numero
    - 2. Sigue el número
    - 3. Lee en num
    - 4. Se llama al método establecerNumero(num) del objeto objNumero, para colocar el valor de num en el dato numero
    - 5. Se incrementa sumatoria con numero; que se accede llamando al método obtenerNumero() del objeto objNumero
  - f. Fin del ciclo for
  - g. Calcula promedio
  - h. Imprime promedio
  - i. Fin Método
- Fin de la Clase EjecutaNumero2
- Fin del algoritmo

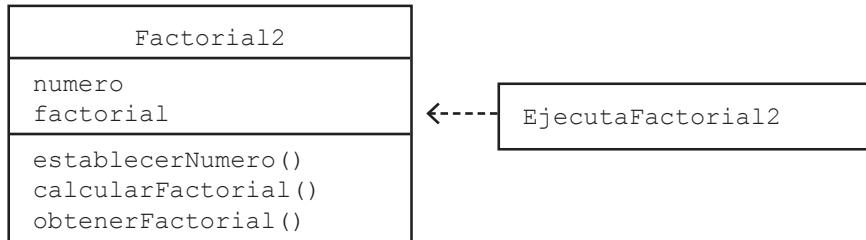
#### Ejercicio 13.4.1.2

Elabore un algoritmo que lea un valor N, entero y positivo, y que le calcule e imprima su factorial. Por ejemplo, si se lee el 5, su factorial es el producto de  $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$ . El factorial de 0 es 1.

A continuación se presenta el algoritmo de la solución:

*(Primero hágalo usted, después compare la solución.)*

Diagrama de clases



Algoritmo FACTORIAL  
 Clase Factorial2  
 1. Declarar  
 Datos  
 numero: Entero

```
factorial: Entero
2. Método establecerNúmero(nu: Entero)
    a. numero = nu
    b. Fin Método establecerNúmero
3. Método calcularFactorial()
    a. Declarar
        Variables
        i: Entero
    b. if numero == 0 then
        1. factorial = 1
    c. else
        1. factorial = 1
        2. for i=numero; i>=1; i--
            a. factorial = factorial * i
        3. endfor
    d. endif
    e. Fin Método calcularFactorial
4. Método obtenerFactorial() Entero
    a. return factorial
    b. Fin Método obtenerFactorial
Fin Clase Factorial2
Clase EjecutaFactorial2
1. Método principal()
    a. Declarar
        Variables
        num: Entero
    b. Declarar, crear e iniciar objeto
        Factorial2 objFactorial = new Factorial2()
    c. Solicitar Número
    d. Leer num
    e. Establecer objFactorial.establecerNúmero(num)
    f. Calcular objFactorial.calcularFactorial()
    g. Imprimir objFactorial.obtenerFactorial()
    h. Fin Método principal
Fin Clase EjecutaFactorial2
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Factorial2.java y EjecutaFactorial2.java



*Explicación:*

El algoritmo tiene dos clases: la Clase Factorial2 y la clase EjecutaFactorial2.

En la Clase Factorial2:

1. Se declaran los datos que representan la estructura de la clase:  
numero para el número  
factorial para el factorial
  2. Método establecerNúmero(nu: Entero)  
Recibe en el parámetro nu el valor que luego coloca en el dato numero
  3. Método calcularFactorial()
    - a. Declarar  
Variables  
i: Entero
    - b. Si numero == 0 Entonces
      1. factorial = 1
    - c. Si no
      1. Inicia factorial en 1
      2. Inicia ciclo for desde i=numero hasta 1 con decrementos de 1
        - a. Calcula factorial = factorial \* i
      3. Fin del for
    - d. Fin del if
    - e. Fin Método
  4. Método obtenerFactorial()  
Retorna factorial
- Fin de la Clase Factorial2

En la Clase EjecutaFactorial2, en el Método principal():

- a. Se declara:  
La variable num para leer el número
  - b. Se declara el objeto objFactorial, usando como base a la clase Factorial2;  
dicho objeto se crea e inicializa mediante el constructor por defecto Factorial2()
  - c. Solicita el número
  - d. Lee en num
  - e. Se llama al método establecerNúmero(num) del objeto objFactorial, para colocar el valor de num en el dato numero
  - f. Se llama al método calcularFactorial() del objeto objFactorial, para calcular el factorial
  - g. Se llama al método obtenerFactorial() del objeto objFactorial, para acceder e imprimir el valor del dato factorial
  - h. Fin Método
- Fin de la Clase EjecutaFactorial2
- Fin del algoritmo

La tabla 13.2 muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 13 de la Web del libro.

**Tabla 13.2**

| Ejercicio          | Descripción                                |
|--------------------|--------------------------------------------|
| Ejercicio 13.2.1.3 | Calcula el factorial a N números           |
| Ejercicio 13.2.1.4 | Procesa obreros con dos do...while         |
| Ejercicio 13.2.1.5 | Procesa obreros con un do...while y un for |
| Ejercicio 13.2.1.6 | Procesa obreros con un for y un do...while |



## 13.5 Ejercicios propuestos para for

Como ejercicios propuestos para este punto se recomiendan del capítulo 5, del punto de la repetición for (5.2), algunos de los ejercicios resueltos que no fueron incluidos en este punto, además todos los ejercicios propuestos en dicho punto.

## 13.6 Diseño de algoritmos OO usando la repetición while

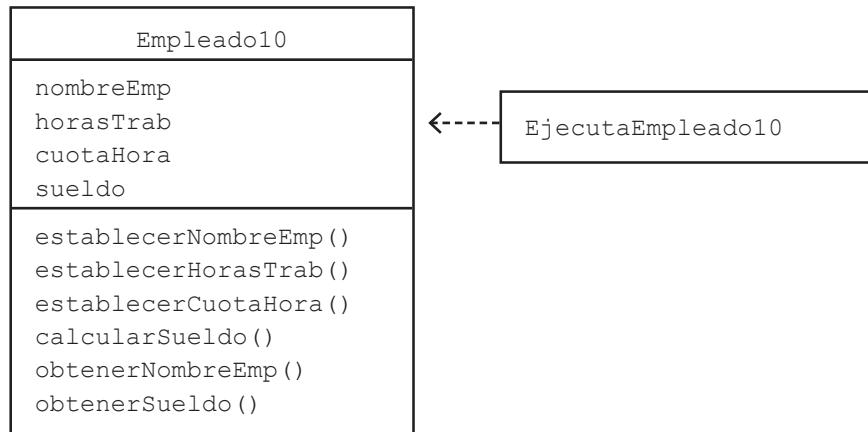
En este punto se utilizará la estructura de repetición while en seudocódigo, misma que se estudió en el punto 5.3 del capítulo 5, pero ahora aplicada conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejemplo:

Elaborar un algoritmo que permita procesar varios empleados, igual al primer ejemplo del primer punto (do...while) de este capítulo. Por cada empleado se leen los datos: Nombre del empleado, número de horas trabajadas y cuota por hora, y se imprime el nombre y sueldo.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS  
Clase Empleado10

1. Declarar
 

Datos

    - nombreEmp: Cadena
    - horasTrab: Entero
    - cuotaHora: Real
    - sueldo: Real
  2. Método establecerNombreEmp (nom: Cadena)
    - a. nombreEmp = nom
    - b. Fin Método establecerNombreEmp
  3. Método establecerHorasTrab (horasTr: Entero)
    - a. horasTrab = horasTr
    - b. Fin Método establecerHorasTrab
  4. Método establecerCuotaHora (cuotaHr: Real)
    - a. cuotaHora = cuotaHr
    - b. Fin Método establecerCuotaHora
  5. Método calcularSueldo ()
    - a. sueldo = horasTrab \* cuotaHora
    - b. Fin Método calcularSueldo
  6. Método obtenerNombreEmp () Cadena
    - a. return nombreEmp
    - b. Fin Método obtenerNombreEmp
  7. Método obtenerSueldo () Real
    - a. return sueldo
    - b. Fin Método obtenerSueldo
- Fin Clase Empleado10

```
Clase EjecutaEmpleado10
1. Método principal()
   a. Declarar
      Variables
         nomEmp: Cadena
         hrsTra: Entero
         cuoHr: Real
         desea: Carácter
   b. Preguntar "¿Desea procesar empleado (S/N) ?"
   c. Leer desea
   d. while desea == 'S'
      1. Declarar, crear e iniciar objeto
         Empleado10 objEmpleado = new Empleado10()
      2. Solicitar Nombre, número de horas
         trabajadas y cuota por hora
      3. Leer nomEmp, hrsTra, cuoHr
      4. Establecer
         objEmpleado.establecerNombreEmp(nomEmp)
         objEmpleado.establecerHorasTrab(hrsTra)
         objEmpleado.establecerCuotaHora(cuoHr)
      5. Calcular objEmpleado.calcularSueldo()
      6. Imprimir objEmpleado.obtenerNombreEmp()
         objEmpleado.obtenerSueldo()
      7. Preguntar "¿Desea procesar otro
         empleado (S/N) ?"
      8. Leer desea
   e. endwhile
   f. Fin Método principal
Fin Clase EjecutaEmpleado10
Fin
```

En la zona de descarga de la Web del libro, están disponibles:

Programa en Java: Empleado10.java y EjecutaEmpleado10.java



#### *Explicación:*

El algoritmo tiene dos clases: la Clase Empleado10 y la clase EjecutaEmpleado10

En la Clase Empleado10:

1. Se declaran los datos que representan la estructura de la clase:  
nombreEmp para el nombre del empleado  
horasTrab para las horas trabajadas del empleado  
cuotaHora para la cuota por hora  
sueldo para el sueldo del empleado

2. Método establecerNombreEmp(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp
  3. Método establecerHorasTrab(horasTr: Entero)  
Recibe en el parámetro horasTr el valor que luego coloca en el dato horasTrab
  4. Método establecerCuotaHora(cuotaHr: Real)  
Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuotaHora
  5. Método calcularSueldo()  
Calcula el sueldo del empleado
  6. Método obtenerNombreEmp() Cadena  
Retorna nombreEmp (nombre del empleado)
  7. Método obtenerSueldo() Real  
Retorna sueldo
- Fin de la Clase Empleado10

En la Clase EjecutaEmpleado10, en el Método principal():

- a. Se declara:
  - La variable nomEmp para leer el nombre del empleado
  - La variable hrsTra para leer las horas trabajadas
  - La variable cuoHr para leer la cuota por hora
  - La variable desea para controlar el ciclo repetitivo
- b. Pregunta “¿Desea procesar empleado (S/N)?”
- c. Lee en desea la respuesta
- d. Inicia ciclo while mientras desea == ‘S’
  1. Se declara el objeto objEmpleado, usando como base a la clase Empleado10; dicho objeto se crea e inicializa mediante el constructor por defecto Empleado10()  
Observe que cada vez que entra al ciclo crea un nuevo objeto empleado
  2. Se Solicitan el Nombre, número de horas trabajadas y cuota por hora
  3. Se leen en nomEmp, hrsTra, cuoHr
  4. Se llama al método establecerNombreEmp(nomEmp) del objeto objEmpleado, para colocar el valor de nomEmp en el dato nombreEmp  
Se llama al método establecerHorasTrab(hrsTra) del objeto objEmpleado, para colocar el valor de hrsTra en el dato horasTrab  
Se llama al método establecerCuotaHora(cuoHr) del objeto objEmpleado, para colocar el valor de cuoHr en el dato cuotaHora
  5. Se llama al método calcularSueldo() del objeto objEmpleado, para calcular el sueldo

6. Se llama al método obtenerNombreEmp() del objeto objEmpleado, para acceder e imprimir el valor del dato nombreEmp  
Se llama al método obtenerSueldo() del objeto objEmpleadom para acceder e imprimir el valor del dato sueldo
  7. Se pregunta “¿Desea procesar otro empleado(S/N)?”
  8. Se lee en desea la respuesta
- e. Fin del while
- f. Fin Método
- Fin de la Clase EjecutaEmpleado10
- Fin del algoritmo

### 13.6.1 Ejercicios resueltos para while

En este punto se presentan algunos de los ejercicios resueltos en el punto 5.3 del capítulo 5, es decir, que el mismo problema ya lo resolvimos aplicando la programación estructurada en el capítulo 5, y aquí lo estamos resolviendo aplicando la metodología de la programación orientada a objetos.

#### Ejercicio 13.6.1.1

El sueldo que perciben los vendedores de una empresa automotriz está integrado de la manera siguiente: El salario mínimo, más \$100.00 por cada automóvil vendido, más el 2% del valor de los automóviles vendidos.

Datos que se tienen por cada vendedor:

Nombre del Vendedor: XXXXXXXXXXXXXXXXXXXXXXXXX

precio automóvil: 999,999.99

precio automóvil: 999,999.99

precio automóvil: 999,999.99

.....

Nombre del Vendedor: XXXXXXXXXXXXXXXXXXXXXXXXX

precio automóvil: 999,999.99

precio automóvil: 999,999.99

precio automóvil: 999,999.99

.....

Nombre del Vendedor: XXXXXXXXXXXXXXXXXXXXXXXXX

precio automóvil: 999,999.99

precio automóvil: 999,999.99

Como se puede apreciar, se tienen varios vendedores, por cada vendedor, se tiene el nombre y el precio de cada automóvil que vendió en la quincena; es posible que algunos vendedores no hayan realizado venta alguna, en tal caso, sólo se tendrá el nombre.

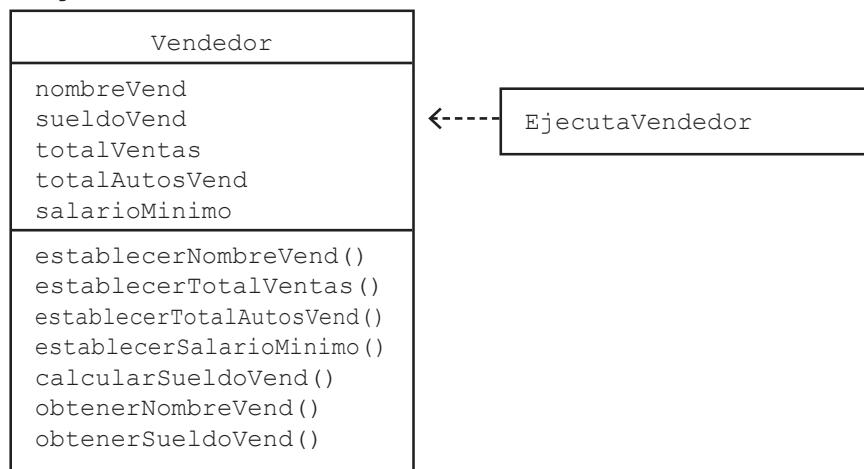
Elaborar un algoritmo que permita leer los datos e imprimir el siguiente reporte:

| NÓMINA QUINCENAL       |            |
|------------------------|------------|
| NOMBRE                 | SUELDO     |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| ---                    | ---        |
| XXXXXXXXXXXXXXXXXXXXXX | 99,999.99  |
| TOTALES 999            | 999,999.99 |

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted, después compare la solución.)

Diagrama de clases



#### Algoritmo VENDEDORES DE AUTOMÓVILES

Clase Vendedor

1. Declarar

Datos

nombreVend: Cadena

totalAutosVend: Entero

sueldoVend, totalVentas, salarioMinimo:  
Real

2. Método establecerNombreVend(nom: Cadena)

a. nombreVend = nom

b. Fin Método establecerNombreVend

3. Método establecerTotalVentas(toVen: Real)

a. totalVentas = toVen

b. Fin Método establecerTotalVentas

4. Método establecerTotalAutosVend(toAuVen: Entero)

a. totalAutosVend = toAuVen

b. Fin Método establecerTotalAutosVend

```
5. Método establecerSalarioMinimo(saMi: Real)
   a. salarioMinimo = saMi
   b. Fin Método establecerSalarioMinimo
5. Método establecerSalarioMinimo(saMi: Real)
   a. salarioMinimo = saMi
   b. Fin Método establecerSalarioMinimo
6. Método calcularSueldoVend()
   a. sueldoVend=salarioMinimo+(totalAutosVe
      nd*100.00)+(totalVentas*0.02)
   b. Fin Método calcularSueldoVend
7. Método obtenerNombreVend() Cadena
   a. return nombreVend
   b. Fin Método obtenerNombreVend
8. Método obtenerSueldoVend() Real
   a. return sueldoVend
   b. Fin Método obtenerSueldoVend
Fin Clase Vendedor
Clase EjecutaVendedor
1. Método principal()
   a. Declarar
      Variables
         nombre: Cadena
         desea, otro: Carácter
         totAutos, totVend: Entero
         precioAuto, salMin, sueldo,
         totSueldos, totVendido: Real
   b. Solicitar el Salario mínimo
   c. Leer salMin
   d. Imprimir Encabezado
   e. totSueldos = 0
      totVend = 0
   f. do
      1. Declarar, crear e iniciar objeto
         Vendedor objVendedor = new Vendedor()
      2. Solicitar el Nombre del vendedor
      3. Leer nombre
      4. totAutos = 0
         totVendido = 0
      5. Preguntar "¿Hay automóvil vendido (S/N)?"
      6. Leer otro
      7. while otro == 'S'
         a. Solicitar el precio del automóvil
         b. Leer precioAuto
         c. totAutos = totAutos + 1
            totVendido = totVendido + precioAuto
```

```

d. Preguntar "¿Hay otro auto vendido (S/N)?"  

e. Leer otro  

8. endwhile  

9. Establecer  

    objVendedor.establecerNombreVend(nombre)  

    objVendedor.establecerTotalVentas(totVendido  

)  

    objVendedor.establecerTotalAutosVend(totAutos)  

    objVendedor.establecerSalarioMinimo(salMin)  

10.Calcular objVendedor.calcularSueldoVend()  

11.Imprimir objVendedor.obtenerNombreVend()  

    objVendedor.obtenerSueldoVend()  

12.totVend = totVend + 1  

    totSueldos = totSueldos +  

        objVendedor.obtenerSueldoVend()  

13.Preguntar "¿Hay otro vendedor (S/N)?"  

14.Leer desea  

g. while desea == 'S'  

h. Imprimir totVend, totSueldos  

i. Fin Método principal  

Fin Clase EjecutaVendedor  

Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
 Programa en Java: Vendedor.java y EjecutaVendedor.java

#### *Explicación:*

El algoritmo tiene dos clases: la Clase Vendedor y la clase EjecutaVendedor.

En la Clase Vendedor:

1. Se declaran los datos que representan la estructura de la clase:  
 nombreVend para el nombre del vendedor  
 sueldoVend para el sueldo del vendedor  
 totalVentas para la cantidad vendida  
 totalAutosVend para los automóviles vendidos  
 salarioMinimo para el salario mínimo
2. Método establecerNombreVend(nom: Cadena)  
 Recibe en el parámetro nom el valor que luego coloca en el dato nombre-Vend
3. Método establecerTotalVentas(toVen: Real)  
 Recibe en el parámetro toVen el valor que luego coloca en el dato totalVentas

4. Método establecerTotalAutosVend (toAuVen: Entero)  
Recibe en el parámetro toAuVen el valor que luego coloca en el dato tota-  
lAutosVend.
  5. Método establecerSalarioMinimo(saMi: Real)  
Recibe en el parámetro saMi el valor que luego coloca en el dato salarioMi-  
nimo.
  6. Método calcularSueldoVend ()  
Calcula el sueldo sumando el salario mínimo, más 100.00 por cada auto-  
móvil vendido, más el 2% del total vendido en dinero
  7. Método obtenerNombreVend() Cadena  
Retorna nombreVend
  8. Método obtenerSueldoVend () Real  
Retorna sueldoVend
- Fin de la Clase Vendedor

En la Clase EjecutaVendedor, en el Método principal():

- a. Se declara:
  - La variable nombre para leer el nombre del vendedor
  - La variable precioAuto para leer el precio de cada automóvil vendido
  - La variable salMin para leer el salario mínimo
  - La variable totVend para contar el total de vendedores
  - La variable totAutos para calcular el total de automóviles vendidos
  - La variable sueldo para calcular el sueldo del vendedor
  - La variable totSueldos para calcular el total de los sueldos de los vendedo-  
res
  - La variable totVendido para calcular el total vendido por cada vendedor
  - Las variables desea y otro para controlar los ciclos repetitivos
- b. Solicita el Salario mínimo quincenal
- c. Lee en salMin
- d. Imprime el encabezado
- e. Inicia totSueldos y totVend en 0
- f. Inicia ciclo do
  - 1. Se declara el objeto objVendedor, usando como base a la clase  
Vendedor; dicho objeto se crea e inicializa mediante el constructor por  
defecto Vendedor()  
Observe que cada vez que entra al ciclo crea un nuevo objeto vendedor
  - 2. Solicita el Nombre del vendedor
  - 3. Lee en nombre
  - 4. Inicia totAutos y totVendido en 0
  - 5. Pregunta “¿Hay automóvil vendido (S/N)?”

6. Lee en otro la respuesta
  7. Inicia ciclo while mientras otro == 'S'
    - a. Sigue el precio del automóvil
    - b. Lee en precioAuto
    - c. Incrementa totAutos en 1  
Incrementa totVendido con precioAuto
    - d. Pregunta “¿Hay otro automóvil vendido (S/N)?”
    - e. Lee en otro la respuesta
  8. Fin del while
  9. Se llama al método establecerNombreVend(nombre) del objeto objVendedor, para colocar el valor de nombre en el dato nombreVend  
Se llama al método establecerTotalAutosVend (totAutos) del objeto objVendedor, para colocar el valor de totAutos en el dato totalAutosVend  
Se llama al método establecerTotalVentas(totVendido) del objeto objVendedor, para colocar el valor de totVendido en el dato totalVentas  
Se llama al método establecerSalarioMinimo(salMin) del objeto objVendedor, para colocar el valor de salMin en el dato salarioMinimo
  10. Se llama al método calcularSueldoVend() del objeto objVendedor, para calcular el sueldo
  11. Se llama al método obtenerNombreVend() del objeto objVendedor, para acceder e imprimir el valor del dato nombreVend  
Se llama al método obtenerSueldoVend() del objeto objVendedor, para acceder e imprimir el valor del dato sueldoVend
  12. Se incrementa totVend en 1  
Se incrementa totSueldos con el sueldo del empleado; mismo que se accede llamando al método obtenerSueldoVend() del objeto objVendedor
  13. Pregunta “¿Hay otro vendedor (S/N)?”
  14. Lee en desea la respuesta
- g. Fin del ciclo (while desea == 'S'). Si se cumple regresa al do; si no, se sale del ciclo
  - h. Imprime totVend, totSueldos
  - i. Fin Método
- Fin de la Clase EjecutaVendedor
- Fin del algoritmo

#### Ejercicio 13.6.1.2

La Comisión Nacional del Agua Delegación Sonora, lleva un registro de las lluvias que se presentan en todas las poblaciones del estado, de manera que se tienen los siguientes datos:

Población 1: X-----X

Lluvia: ----

Lluvia: ----

.

Lluvia: ----

Población 2: X-----X

Lluvia: ----

Lluvia: ----

.

Lluvia: ----

Población N: X-----X

Lluvia: ----

Lluvia: ----

.

Lluvia: ----

Por cada población aparece el dato lluvia tantas veces como lluvias haya habido, este dato está en milímetros cúbicos, pudiera darse el caso de que en alguna población no traiga ningún dato de lluvia, esto quiere decir que no llovió.

Elaborar un algoritmo que lea estos datos e imprima el reporte:

| REPORTE DE LLUVIAS                     |              |              |
|----------------------------------------|--------------|--------------|
| POBLACIÓN                              | TOTAL LLUVIA | NIVEL LLUVIA |
| XXXXXXXXXXXXXXXXXXXX                   | 999.99       | NULA         |
| XXXXXXXXXXXXXXXXXXXX                   | 999.99       | BAJA         |
|                                        |              | REGULAR      |
| XXXXXXXXXXXXXXXXXXXX                   | 999.99       | ALTA         |
| TOTAL 999 POBLACIONES                  | 999.99       |              |
| TOTAL POBLACIONES DONDE NO LLOVIÓ: 999 |              |              |

Cálculos:

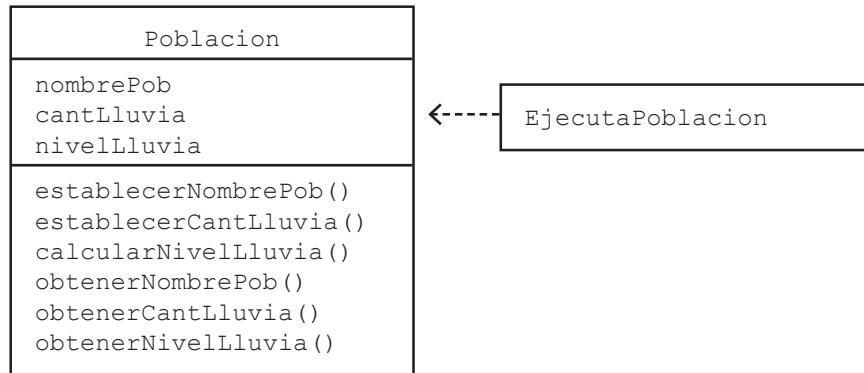
TOTAL LLUVIA es el total de lluvia que se presentó en una población, se calcula sumando la cantidad de milímetros cúbicos de todas las lluvias.

Se piden dos totales: Total de poblaciones procesadas y el total del total de lluvia en todas las poblaciones. También se pide la cantidad de poblaciones donde no se presentó lluvia alguna.

El NIVEL LLUVIA es una observación:  
 NULA si no llovió.  
 BAJA si TOTAL LLUVIA es menor a 100.  
 REGULAR si TOTAL LLUVIA está entre 100 y 250.  
 ALTA si TOTAL LLUVIA es mayor a 250.

A continuación se presenta el algoritmo de la solución:  
*(Primero hágalo usted, después compare la solución.)*

Diagrama de clases



Algoritmo LLUVIAS  
 Clase Población  
 1. Declarar  
 Datos  
 nombrePob: Cadena  
 cantLluvia: Real  
 nivelLluvia: Cadena  
 2. Método establecerNombrePob(nom: Cadena)  
 a. nombrePob = nom  
 b. Fin Método establecerNombrePob  
 3. Método establecerCantLluvia(cant: Real)  
 a. cantLluvia = cant  
 b. Fin Método establecerCantLluvia  
 4. Método calcularNivelLluvia()  
 a. if cantLluvia == 0 then  
 1. nivelLluvia = "NULA"  
 b. endif  
 c. if (cantLluvia > 0)AND(cantLluvia < 100) then  
 1. nivelLluvia = "BAJA"  
 d. endif  
 e. if (cantLluvia >= 100)AND(cantLluvia <= 250)then  
 1. nivelLluvia = "REGULAR"

```
f. endif
g. if cantLluvia > 250 then
    1. nivelLluvia = "ALTA"
h. endif
i. Fin Método calcularNivelLluvia
5. Método obtenerNombrePob() Cadena
    a. return nombrePob
    b. Fin Método obtenerNombrePob
6. Método obtenerCantLluvia() Real
    a. return cantLluvia
    b. Fin Método obtenerCantLluvia
7. Método obtenerNivelLluvia() Cadena
    a. return nivelLluvia
    b. Fin Método obtenerNivelLluvia
Fin Clase Población
Clase EjecutaPoblacion
1. Método principal()
    a. Declarar
        Variables
            poblacion: Cadena
            otro, hay: Carácter
            totPobra, totPobNoLluvia: Entero
            lluvia, totLluvia, toTotLluvia: Real
    b. Imprimir encabezado
    c. totPobra = 0
        totPobNoLluvia = 0
        toTotLluvia = 0
    d. Preguntar "¿Hay población (S/N)?""
    e. Leer hay
    f. while hay == 'S'
        1. Declarar, crear e iniciar objeto
            Poblacion objPoblacion = new Poblacion()
        2. Solicitar POBLACIÓN
        3. Leer poblacion
        4. totLluvia = 0
        5. Preguntar "¿Hay lluvia (S/N)?""
        6. Leer otro
        7. while otro == 'S'
            a. Solicitar Lluvia en milímetros cúbicos
            b. Leer lluvia
            c. totLluvia = totLluvia + lluvia
            d. Preguntar "¿Hay otra lluvia (S/N)?""
            e. Leer otro
```

```

8. endwhile
9. Establecer
    objPoblacion.establecerNombrePob(nombre)
    objPoblacion.establecerCantLluvia(totLluvia)
10.Calcular objPoblacion.calcularNivelLluvia()
11.Imprimir objPoblacion.obtenerNombrePob()
    objPoblacion.obtenerCantLluvia()
    objPoblacion.obtenerNivelLluvia()
12.totPoba = totPoba + 1
    toTotLluvia = toTotLluvia
        +objPoblacion.obtenerCantLluvia()
13.if totLluvia == 0 then
    a. totPobNoLluvia = totPobNoLluvia + 1
14.endif
15.Preguntar "¿Hay otra población (S/N)?""
16.Leer hay
g. endwhile
h. Imprimir totPoba, toTotLluvia, totPobNoLluvia
i. Fin Método principal
    Fin Clase EjecutaPoblacion
Fin

```



En la zona de descarga de la Web del libro, están disponibles:  
Programa en Java: Poblacion.java y EjecutaPoblacion.java

#### *Explicación:*

El algoritmo tiene dos clases: la Clase Poblacion y la clase EjecutaPoblacion

En la Clase Poblacion:

1. Se declaran los datos que representan la estructura de la clase:  
nombrePob para el nombre de la poblacion  
cantLluvia para la cantidad de lluvia  
nivelLluvia para el nivel de lluvia
2. Método establecerNombrePob(nom: Cadena)  
Recibe en el parámetro nom el valor que luego coloca en el dato nombrePob
3. Método establecerCantLluvia(cant: Real)  
Recibe en el parámetro cant el valor que luego coloca en el dato cantLluvia
4. Método calcularNivelLluvia()  
a. Si cantLluvia == 0 entonces
  1. nivelLluvia = "NULA"

b. Fin del if  
c. Si (cantLluvia > 0)AND(cantLluvia < 100) entonces  
    1. nivelLluvia = "BAJA"  
d. Fin del if  
e. Si (cantLluvia >= 100)AND(cantLluvia <= 250) entonces  
    1. nivelLluvia = "REGULAR"  
f. Fin del if  
g. Si cantLluvia > 250 entonces  
    1. nivelLluvia = "ALTA"  
h. Fin del if  
i. Fin Método  
5. Método obtenerNombrePob() Cadena  
Retorna nombrePob  
6. Método obtenerCantLluvia() Real  
Retorna cantLluvia  
7. Método obtenerNivelLluvia() Cadena  
Retorna nivelLluvia  
Fin de la Clase Poblacion

En la Clase EjecutaPoblacion, en el Método principal():

- a. Se declara:  
    La variable poblacion para leer el nombre de la población  
    La variable lluvia para leer la cantidad de cada lluvia que hubo  
    La variable totPoba para contar el total de poblaciones  
    La variable totPobNoLluvia para contar el total de poblaciones donde no llovió  
    La variable totLluvia para calcular el total de lluvia de cada población  
    La variable toTotLluvia para calcular el total de lluvia de todas las poblaciones  
    Las variables otro y hay para controlar los ciclos repetitivos
- b. Imprime el encabezado
- c. Inicia totPoba, totPobNoLluvia y toTotLluvia en 0
- d. Pregunta "¿Hay población (S/N)?"
- e. Lee en hay la respuesta
- f. Inicia ciclo while mientras hay == 'S'
  1. Se declara el objeto objPoblacion, usando como base a la clase Poblacion; dicho objeto se crea e inicializa mediante el constructor por defecto Poblacion()  
    Observe que cada vez que entra al ciclo crea un nuevo objeto población
  2. Solicita POBLACIÓN
  3. Lee en poblacion

4. Inicia totLluvia en 0
  5. Pregunta “¿Hay lluvia (S/N)?”
  6. Lee en otro la respuesta
  7. Inicia ciclo while mientras otro == ‘S’
    - a. Solicita Lluvia en milímetros cúbicos
    - b. Lee en lluvia
    - c. Incrementa totLluvia con lluvia
    - d. Pregunta “¿Hay otra lluvia (S/N)?”
    - e. Lee en otro la respuesta
  8. Fin del while
  9. Se llama al método establecerNombrePob(nombre) del objeto objPoblacion, para colocar el valor de nombre en el dato nombrePob  
Se llama al método establecerCantLluvia(totLluvia) del objeto objPoblacion, para colocar el valor de totLluvia en el dato cantLluvia
  10. Se llama al método calcularLluvia() del objeto objPoblacion, para calcular el nivel de lluvia
  11. Se llama al método obtenerNombrePob() del objeto objPoblacion, para acceder e imprimir el valor del dato nombrePob  
Se llama al método obtenerCantLluvia() del objeto objPoblacion, para acceder e imprimir el valor del dato cantLluvia  
Se llama al método obtenerNivelLluvia() del objeto objPoblacion, para acceder e imprimir el valor del dato nivelLluvia
  12. Incrementa totPoba en 1  
Incrementa toTotLluvia con totLluvia
  13. Si totLluvia == 0 Entonces
    1. Incrementa totPobNoLluvia en 1
  14. Fin del if
  15. Pregunta “¿Hay otra población (S/N)?”
  16. Lee en hay la respuesta
  - g. Fin del while
  - h. Imprime totPoba, toTotLluvia, totPobNoLluvia
  - i. Fin Método
- Fin de la Clase EjecutaPoblacion
- Fin del algoritmo

En la zona de descarga del capítulo 13 de la Web del libro, está disponible el ejercicio resuelto.



| Ejercicio          | Descripción                                    |
|--------------------|------------------------------------------------|
| Ejercicio 13.6.1.3 | Procesa la producción de estaciones de trabajo |

## 13.7 Ejercicios propuestos para while

Como ejercicios propuestos para este punto, se recomiendan del capítulo 5, del punto de la repetición while (5.3), algunos de los ejercicios resueltos que no fueron incluidos en este punto; además todos los ejercicios propuestos en dicho punto.

## 13.8 Resumen de conceptos que debe dominar

- Diseño de algoritmos orientados a objetos usando las estructuras de repetición:
  - do...while, for y while.
  - 
  -

## 13.9 Contenido de la página Web de apoyo



El material marcado con asterisco (\*) sólo está disponible para docentes.

### 13.9.1 Resumen gráfico del capítulo

### 13.9.2 Autoevaluación

### 13.9.3 Programas

### 13.9.4 Ejercicios resueltos

### 13.9.5 Power Point para el profesor (\*)

# A

## Apéndice A: Algoritmos sin usar etiquetas

Aunque el autor de esta obra considera que es mejor utilizar etiquetas porque ayudan a desarrollar disciplina y orden en los estudiantes (virtudes que los maestros debemos estimular), sobre todo en la etapa de formación de los aprendices de programación, en este apéndice se presentan algunos de los algoritmos elaborados en los capítulos del 3 al 13, sólo que aquí se muestra cómo quedan sin utilizar el concepto de etiquetas, con la finalidad de que el lector pueda observar la diferencia y opte por utilizar la forma que le parezca mejor; esto, una vez que haya aprendido a programar.

**Nota:** A algunos maestros no les gusta usar etiquetas para enumerar los pasos de los algoritmos; sin embargo, estimado maestro, si Usted está utilizando este libro como texto, es conveniente que utilice los ejemplos como están planteados en el libro, es decir, con etiquetas; porque si Usted explica los algoritmos sin etiquetas, y luego cuando los alumnos estudien el libro y vean los algoritmos con etiquetas, seguramente les causará confusión, en consecuencia, la utilidad del libro podría ser limitada.

A continuación se presentan algunos algoritmos sin usar etiquetas.

#### Ejercicio 3.4.1 (Capítulo 3)

```
Algoritmo CÁLCULOS LOGARÍTMICOS DE ÁNGULO
Declarar
    Variables
        tamAngulo, senAng, cosAng: Real
    Solicitar Tamaño del ángulo en radianes
    Leer tamAngulo
    Calcular senAng = Seno(tamAngulo)
        cosAng = Coseno(tamAngulo)
    Imprimir senAng, cosAng
Fin
```

#### Ejercicio 4.1.4.1 (Capítulo 4)

```
Algoritmo CALCULA PROMEDIO DE UN ALUMNO
Declarar
    Variables
        nombreAlum: Cadena
        calif1, calif2, calif3, calif4, promedio: Real
        observacion: Cadena
    Solicitar Nombre del alumno, calificación 1,2,3 y 4
    Leer nombreAlum, calif1, calif2, calif3, calif4
    Calcular promedio = (calif1+calif2+calif3+calif4) / 4
    if promedio >= 60 then
        observación = "Aprobado"
    else
        observación = "Reprobado"
    endif
    Imprimir nombreAlum, promedio, observación
Fin
```

#### Ejercicio 5.3.3.1 (Capítulo 5)

```
Algoritmo VENDEDORES DE AUTOMÓVILES
Declarar
    Variables
        nombreVend: Cadena
        desea, otro: Carácter
        totAutos, totVend: Entero
        precioAuto, salMin, sueldo,
```

```

        totSueldos, totVendido: Real
        Solicitar el Salario mínimo
        Leer salMin
        Imprimir Encabezado
        totSueldos = 0
        totVend = 0
        do
            Solicitar el Nombre del vendedor
            Leer nombreVend
            totAutos = 0
            totVendido = 0
            Preguntar "¿Hay automóvil vendido (S/N) ?"
            Leer otro
            while otro == 'S'
                Solicitar el precio del automóvil
                Leer precioAuto
                totAutos = totAutos + 1
                totVendido = totVendido + precioAuto
                Preguntar "¿Hay otro automóvil vendido (S/N) ?"
                Leer otro
            endwhile
            sueldo = salMin+(totAutos*100)+(totVendido*0.02)
            Imprimir nombreVend, sueldo
            totVend = totVend + 1
            totSueldos = totSueldos + sueldo
            Preguntar "¿Hay otro vendedor (S/N) ?"
            Leer desea
        while desea == 'S'
        Imprimir totVend, totSueldos
        Fin
    
```

#### Ejercicio 13.1.2.1 (Capítulo 13)

```

Algoritmo CLIENTES HOJAS HIELO SECO
Clase Cliente3
    Declarar
        Datos
            nombreClie: Cadena
            tipoClie, cantidad: Entero
            precioUni, subTotal, descuento, netoPagar: Real
        Método establecerNombreClie(nom: Cadena)
            nombreClie = nom
        Fin Método establecerNombreClie
    
```

```
Método establecerTipoClie(tip: Entero)
    tipoClie = tip
    Fin Método establecerTipoClie
Método establecerCantidad(can: Entero)
    cantidad = can
    Fin Método establecerCantidad
Método establecerPrecioUni(pre: Real)
    precioUni = pre
    Fin Método establecerPrecioUni
Método calcularSubTotal()
    subTotal = cantidad * precioUni
    Fin Método calcularSubTotal
Método calcularDescuento()
    switch tipoClie
        1: descuento = subTotal * 0.05
        2: descuento = subTotal * 0.08
        3: descuento = subTotal * 0.12
        4: descuento = subTotal * 0.15
    endswitch
    Fin Método calcularDescuento
Método calcularNetoPagar()
    netoPagar = subTotal - descuento
    Fin Método calcularNetoPagar
Método obtenerNombreClie() Cadena
    return nombreClie
    Fin Método obtenerNombreClie
Método obtenerSubTotal() Real
    return subTotal
    Fin Método obtenerSubTotal
Método obtenerDescuento() Real
    return descuento
    Fin Método obtenerDescuento
Método obtenerNetoPagar() Real
    return netoPagar
    Fin Método obtenerNetoPagar
Fin Clase Cliente3
Clase EjecutaCliente3
    Método principal()
        Declarar
            Variables
                nomCli: Cadena
                ticcli, cant, totClientes: Entero
```

```

        preUni, totSubTot, totDescuento,
        totNeto: Real
        desea: Carácter
        Imprimir encabezado
        totClientes = 0
        totSubTot = 0
        totDescuento = 0
        totNeto = 0
        do
            Declarar, crear e iniciar objeto
            Cliente3 objCliente = new Cliente3()
            Solicitar Nombre, Tipo cliente,
            Cantidad, Precio unitario
            Leer nomCli, tiCli, cant, preUni
            Establecer
                objCliente.establecerNombreClie(nomCli)
                objCliente.establecerTipoClie(tiCli)
                objCliente.establecerCantidad(cant)
                objCliente.establecerPrecioUni(preUni)
            Calcular objCliente.calcularSubTotal()
                objCliente.calcularDescuento()
                objCliente.calcularNetoPagar()
            Imprimir objCliente.obtenerNombreClie()
                objCliente.obtenerSubTotal()
                objCliente.obtenerDescuento()
                objCliente.obtenerNetoPagar()
            totClientes = totClientes + 1
            totSubTot = totSubTot
                + objCliente.obtenerSubTotal()
            totDescuento = totDescuento
                + objCliente.obtenerDescuento()
            totNeto = totNeto
                + objCliente.obtenerNetoPagar()
            Preguntar "¿Desea procesar otro cliente(S/N) ?"
            Leer desea
            while desea == 'S'
            Imprimir totClientes, totSubTot,
                totDescuento, totNeto
            Fin Método principal
            Fin Clase EjecutaCliente3
        Fin
    
```

# B

## Apéndice B: Diagramas de flujo

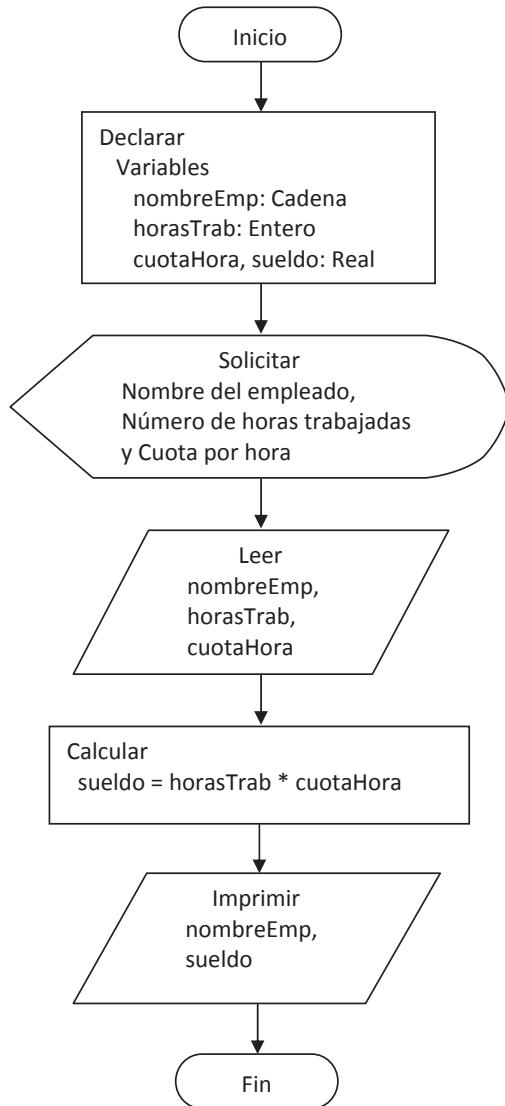
En este apéndice se presentan algunos algoritmos que han sido desarrollados en pseudocódigo en la parte de la programación estructurada del libro (capítulos 3 al 8), que es donde se presenta la metodología original de este autor; pero aquí se utilizan los lineamientos y simbología de los diagramas de flujo conjuntamente con los conceptos del capítulo dos de la metodología original. El propósito es que el lector, alumno o maestro que ya conozca y domine los diagramas de flujo, y que, se le dificulte el aprendizaje de la metodología original de este libro en pseudocódigo, pueda hacer una comparación entre los algoritmos hechos en ambos métodos, a efecto de facilitarle el aprendizaje de la metodología en pseudocódigo.

### Advertencia:

Este autor piensa que los diagramas de flujo resultan didácticos por su enfoque gráfico en la solución de problemas pequeños, sin embargo, cuando los problemas son de mayor magnitud, no son prácticos y se convierten en una técnica realmente difícil de usar y de comprender. Además de que no soportan las estructuras do...while y while en forma natural; y lo que hacemos es simularlas. Este autor no está de acuerdo con el uso de los diagramas de flujo para la enseñanza de la metodología de la programación, es por ello que desarrolló la metodología con pseudocódigo; sin embargo, debido a que hemos observado que muchos maestros que enseñan a programar utilizan los diagramas de flujo, se han incluido en esta edición para que sirva como enlace en el cambio que deben experimentar los maestros que han sido formados y actualmente utilizan los diagramas de flujo, hacia el uso de otra técnica más moderna como lo es el pseudocódigo que presenta este autor.

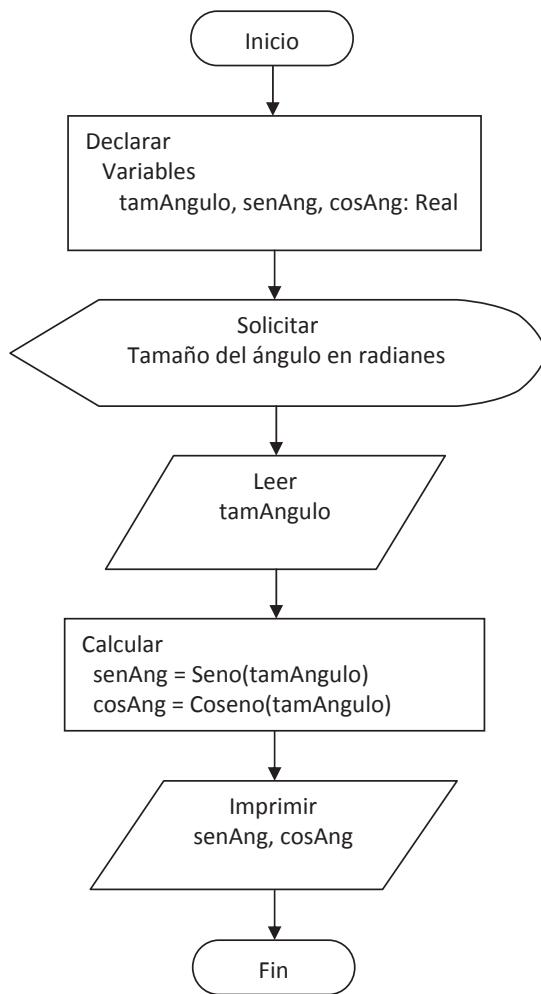
Nuestro primer algoritmo del capítulo 3 (página 53).

### Algoritmo CALCULA SUELDO DE UN EMPLEADO



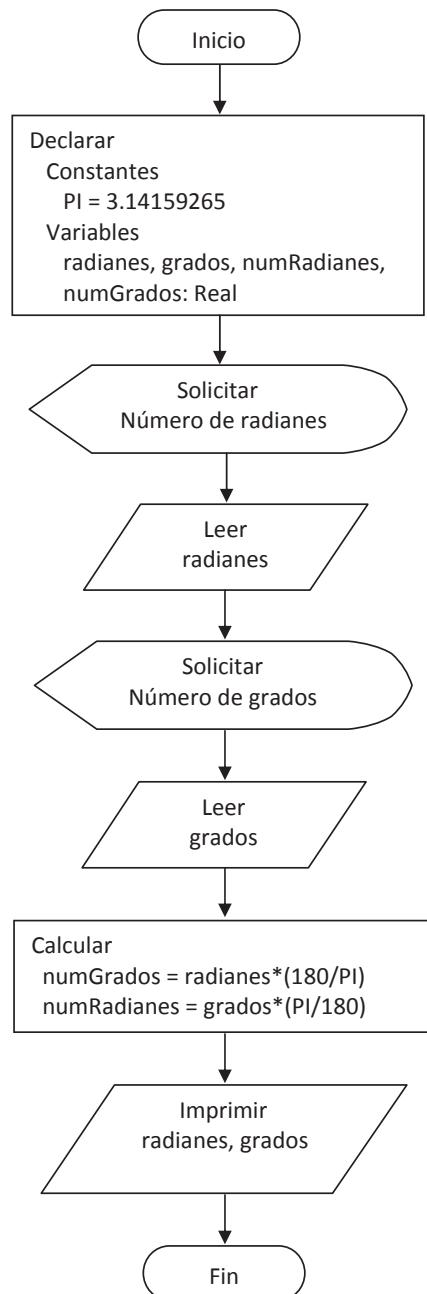
Ejercicio 3.4.1 (página 59).

### Algoritmo CÁLCULOS LOGARÍTMICOS DE ÁNGULO



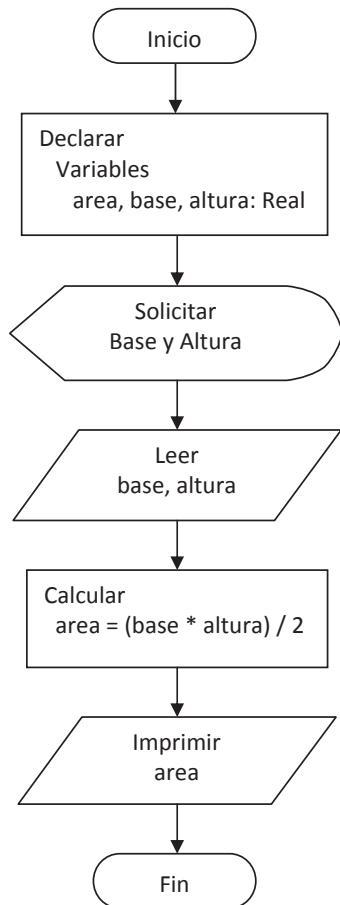
Ejercicio 3.4.2 (página 60).

### Algoritmo CONVIERTE RADIANES A GRADOS Y GRADOS A RADIANES



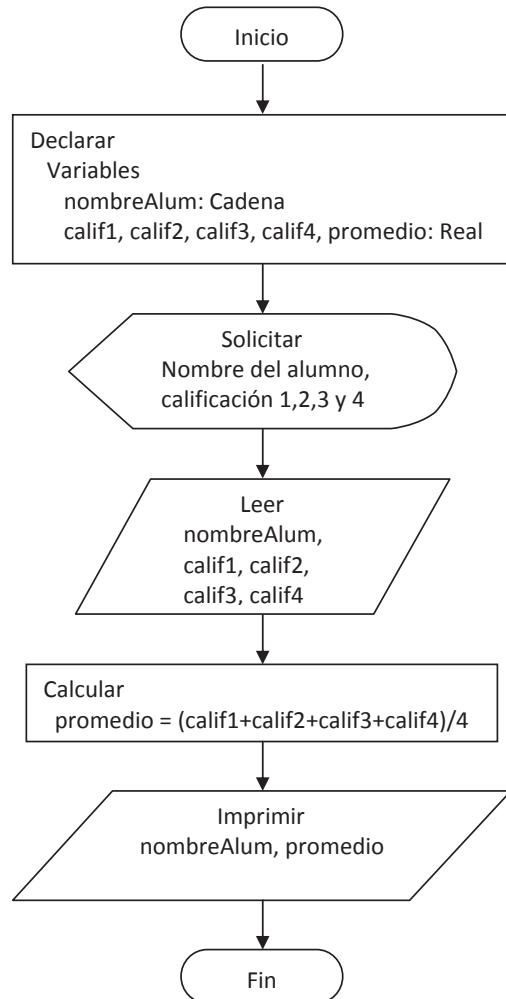
Ejercicio 3.5.1 (página 62).

### Algoritmo ÁREA TRIÁNGULO



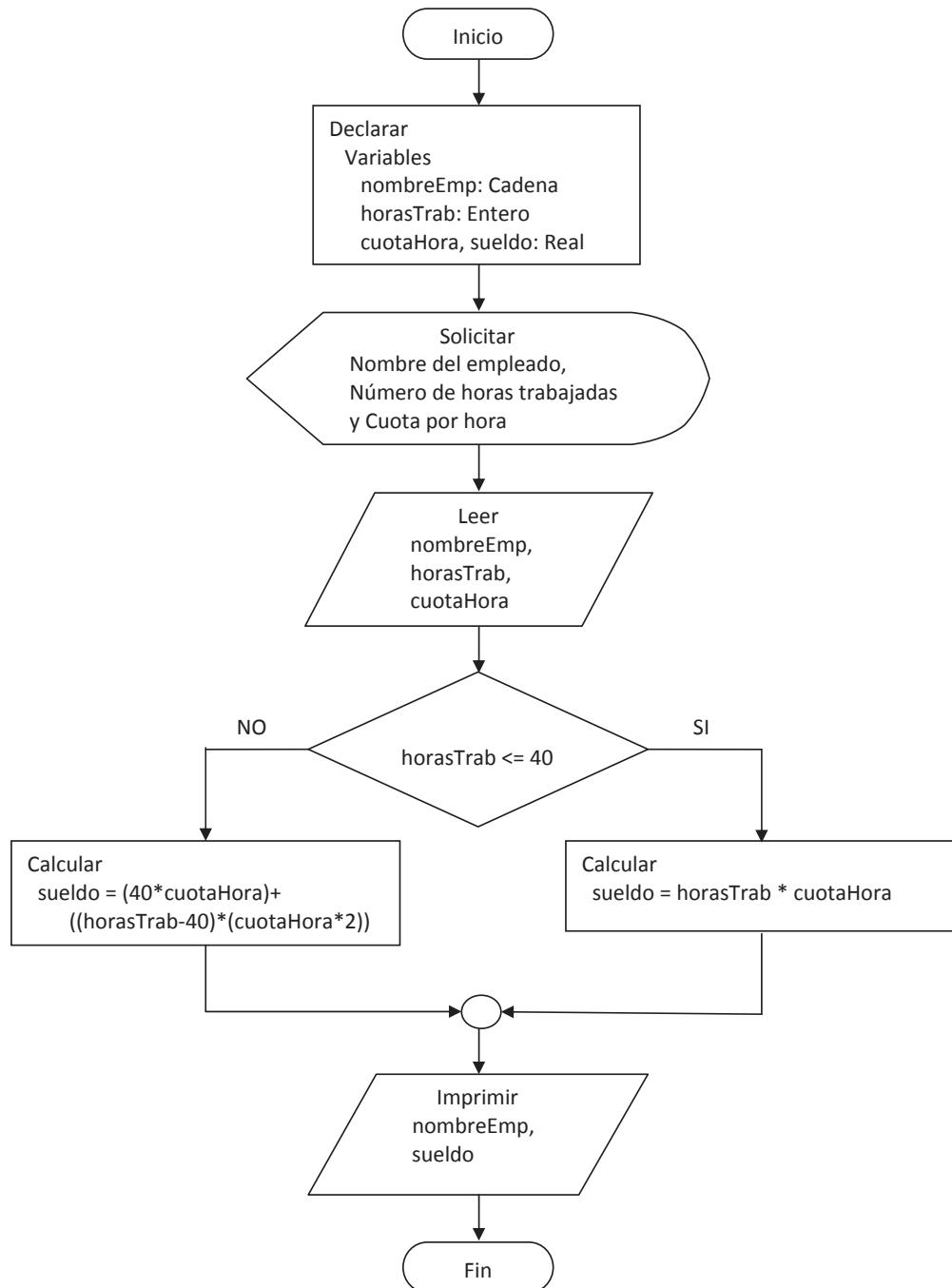
Ejercicio 3.5.2 (página 63).

### Algoritmo CALCULA PROMEDIO DE UN ALUMNO



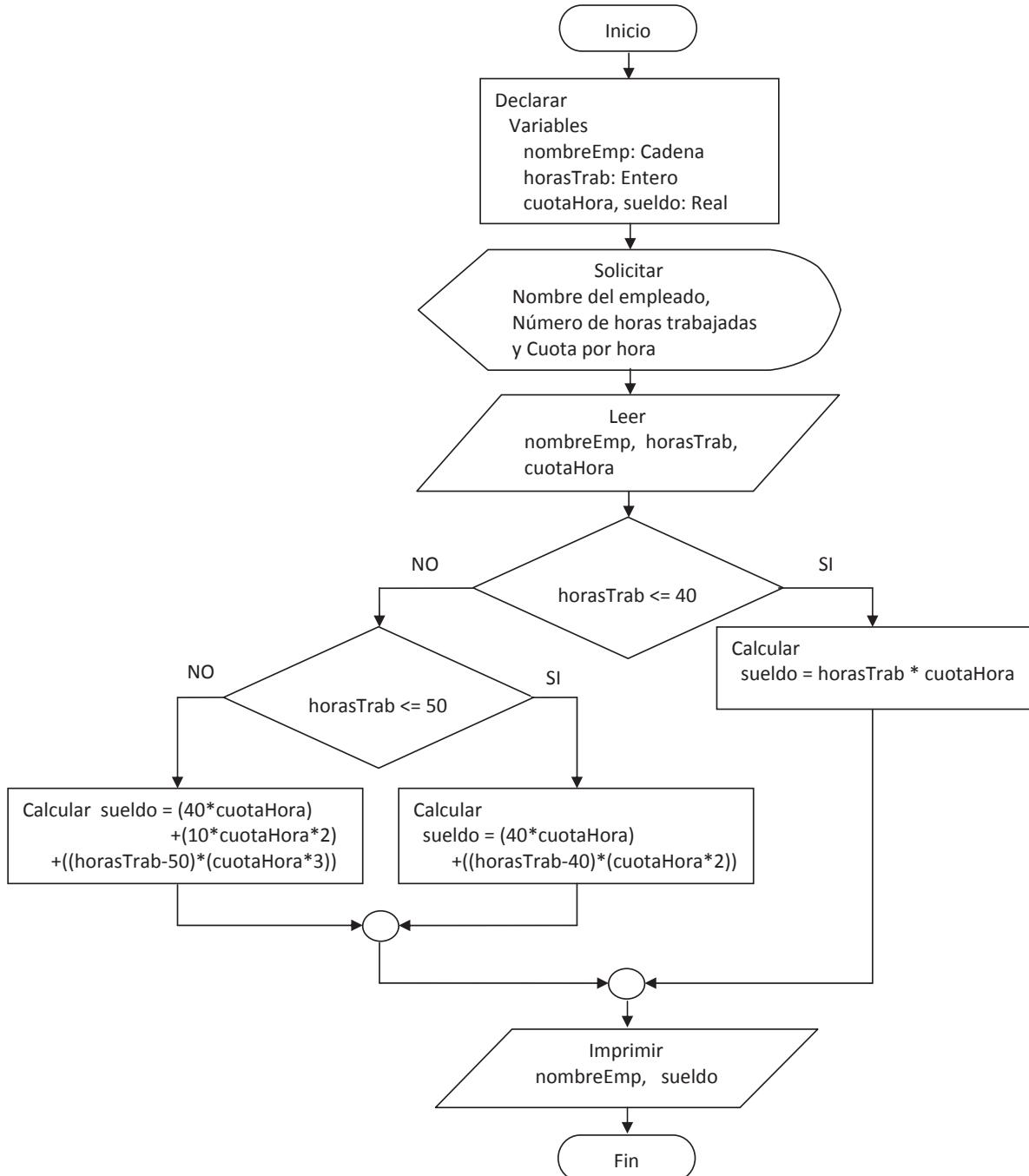
Ejercicio capítulo 4 (página 71).

### Algoritmo CÁLCULO SUELDO DOBLE



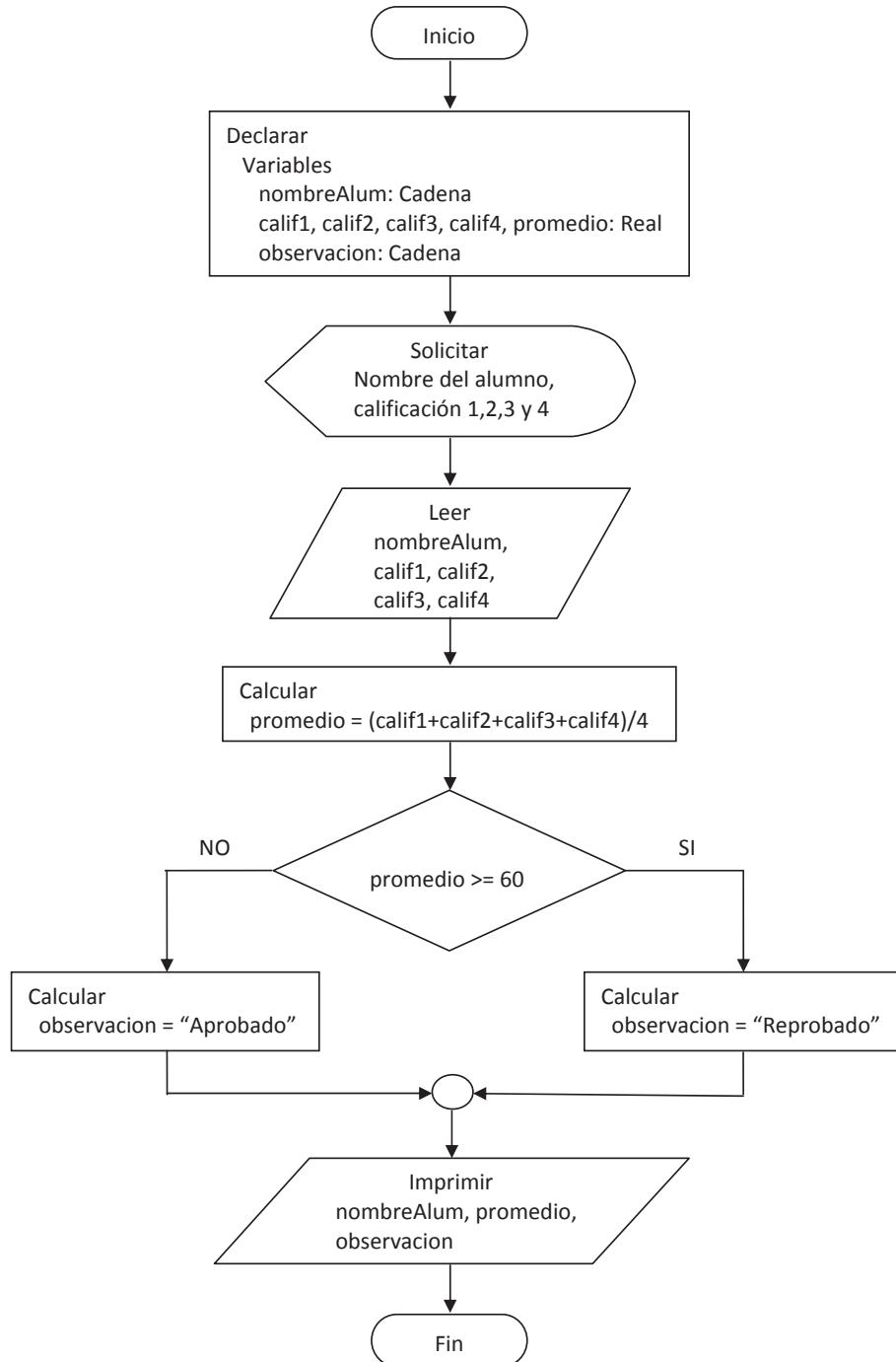
Ejercicio capítulo 4 (página 80).

### Algoritmo CÁLCULO SUELDO TRIPLE



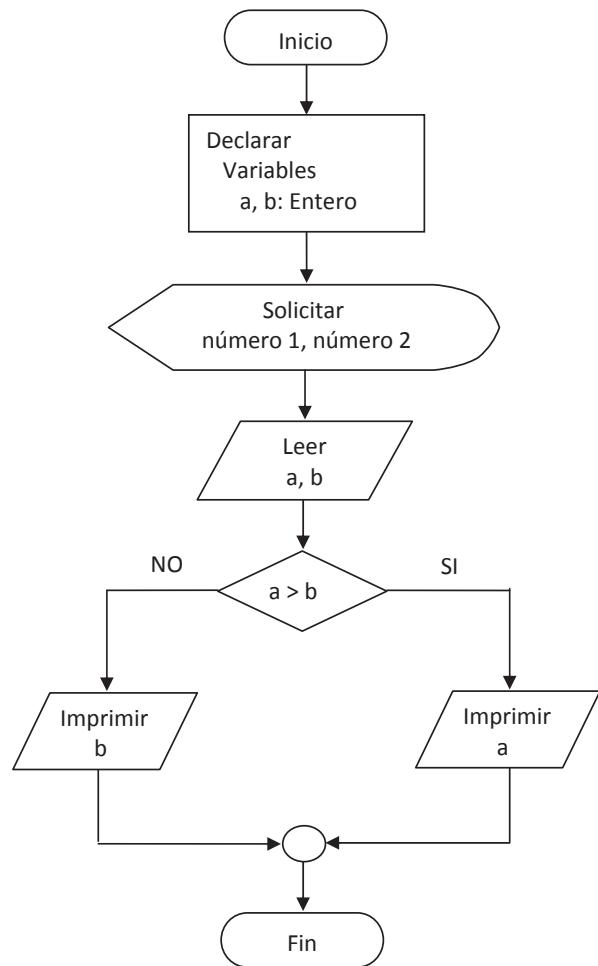
Ejercicio 4.1.4.1 (página 81).

### Algoritmo CALCULA PROMEDIO DE UN ALUMNO



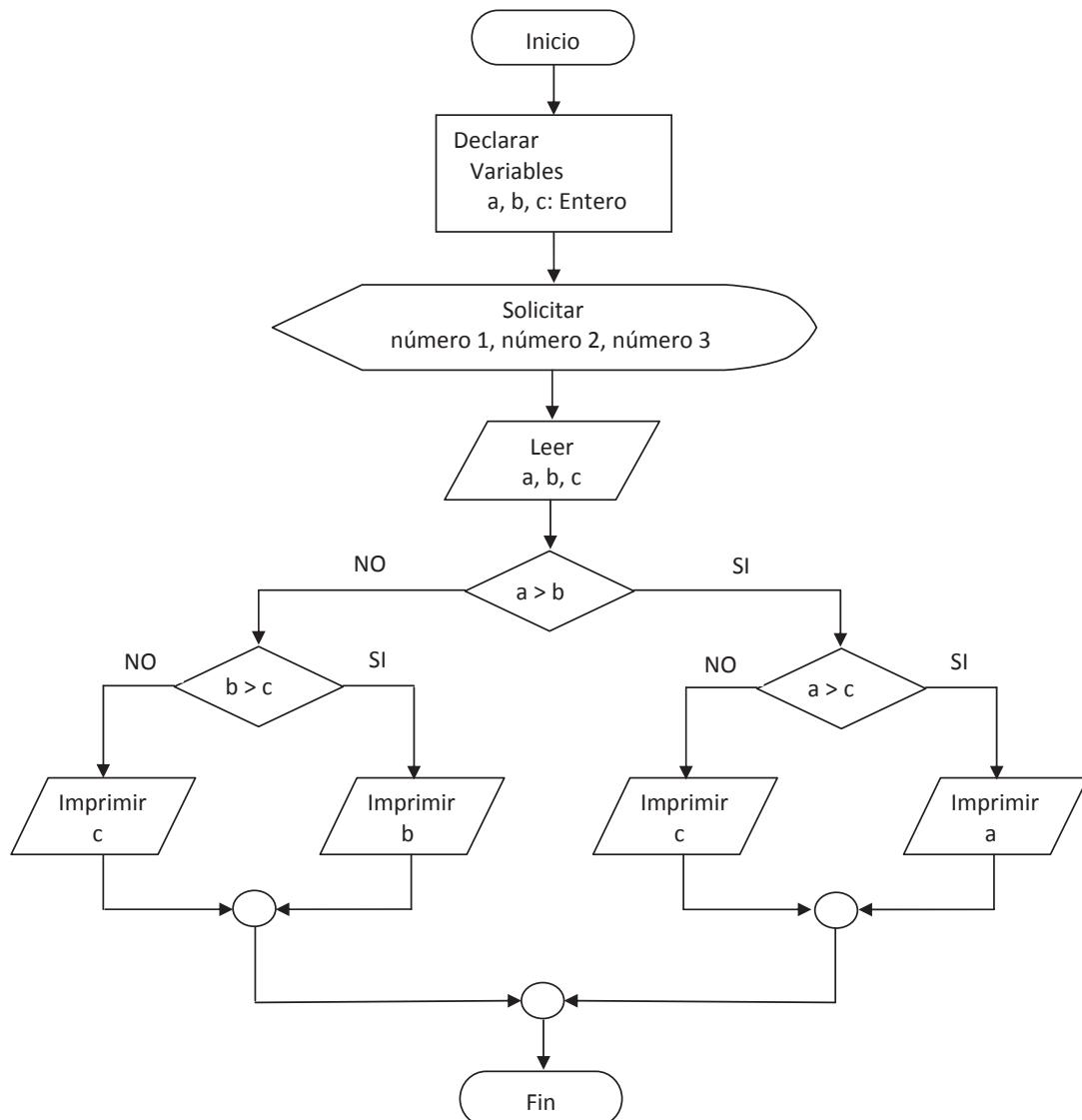
Ejercicio 4.1.4.2 (página 82).

### Algoritmo MAYOR 2 NÚMEROS



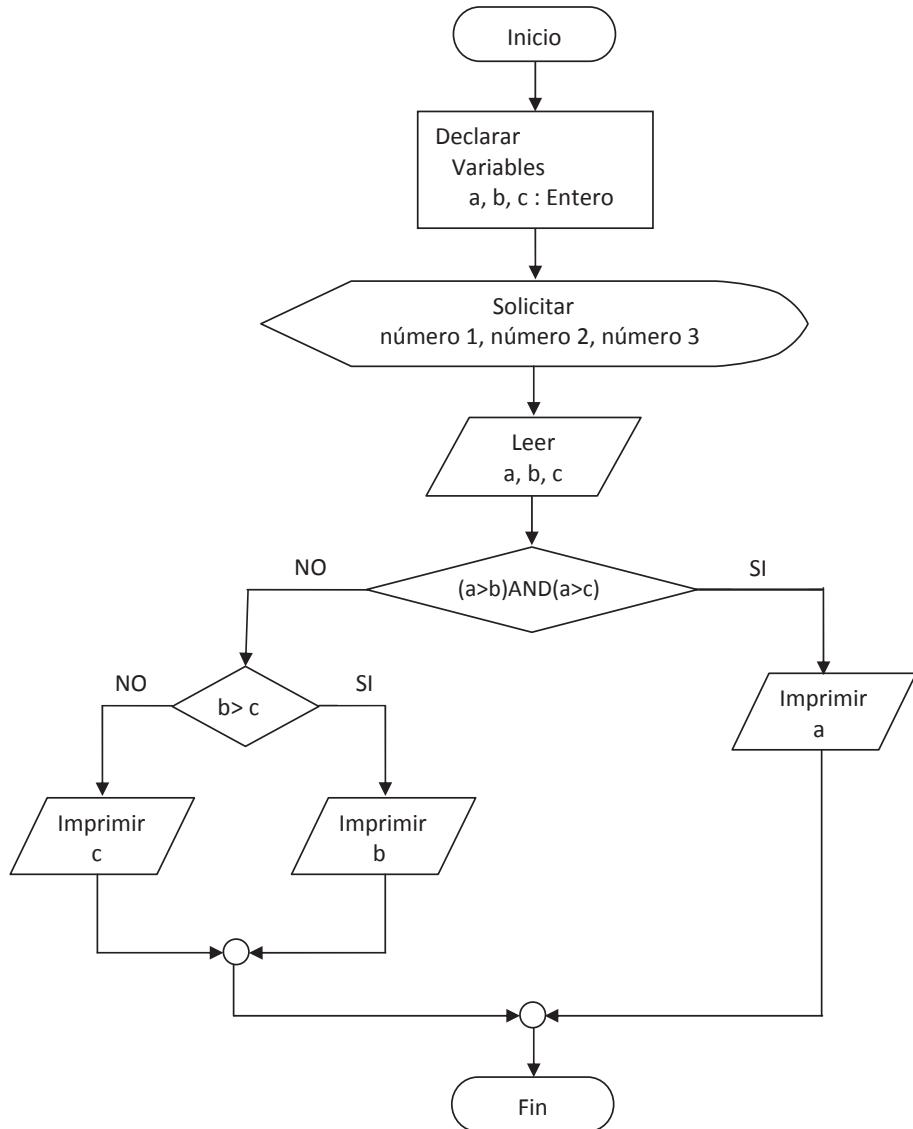
Ejercicio 4.1.4.3 (página 83). Primer método de solución.

### Algoritmo MAYOR 3 NÚMEROS



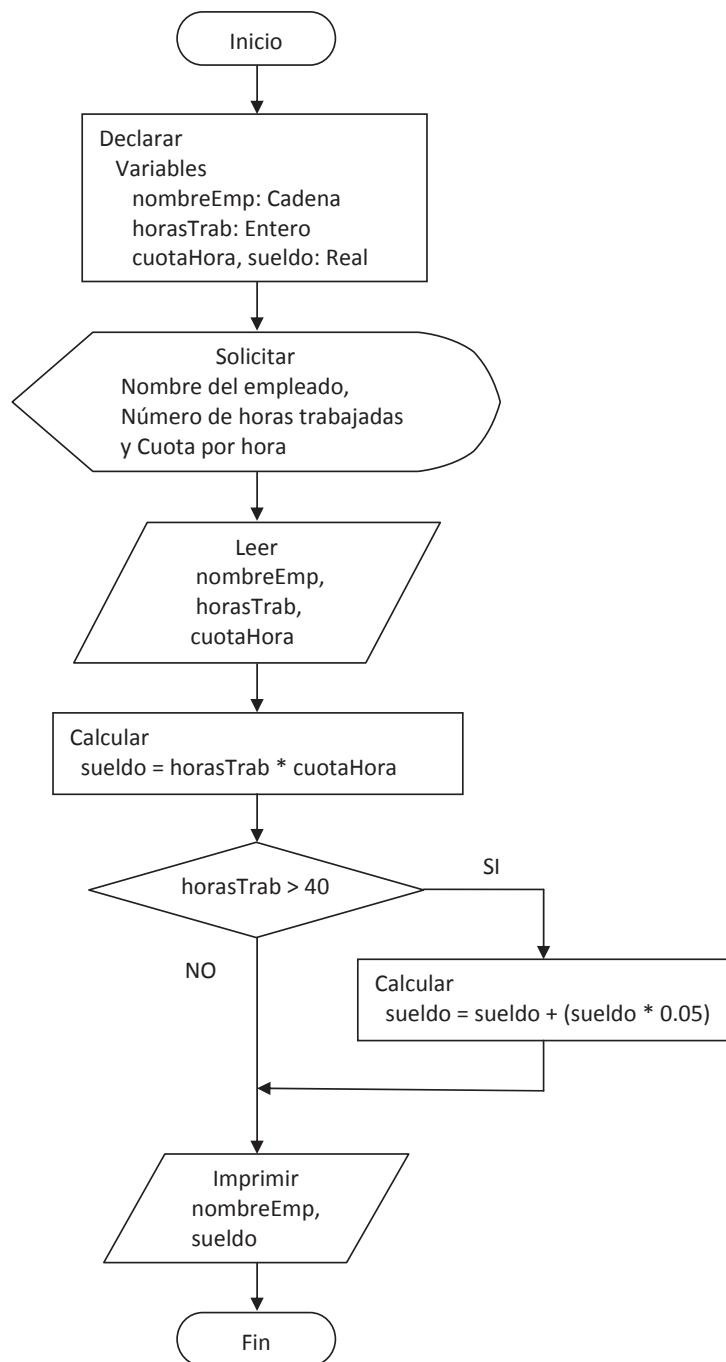
Ejercicio 4.1.4.5 (página 84). Segundo método de solución.

### Algoritmo MAYOR 3 NÚMEROS



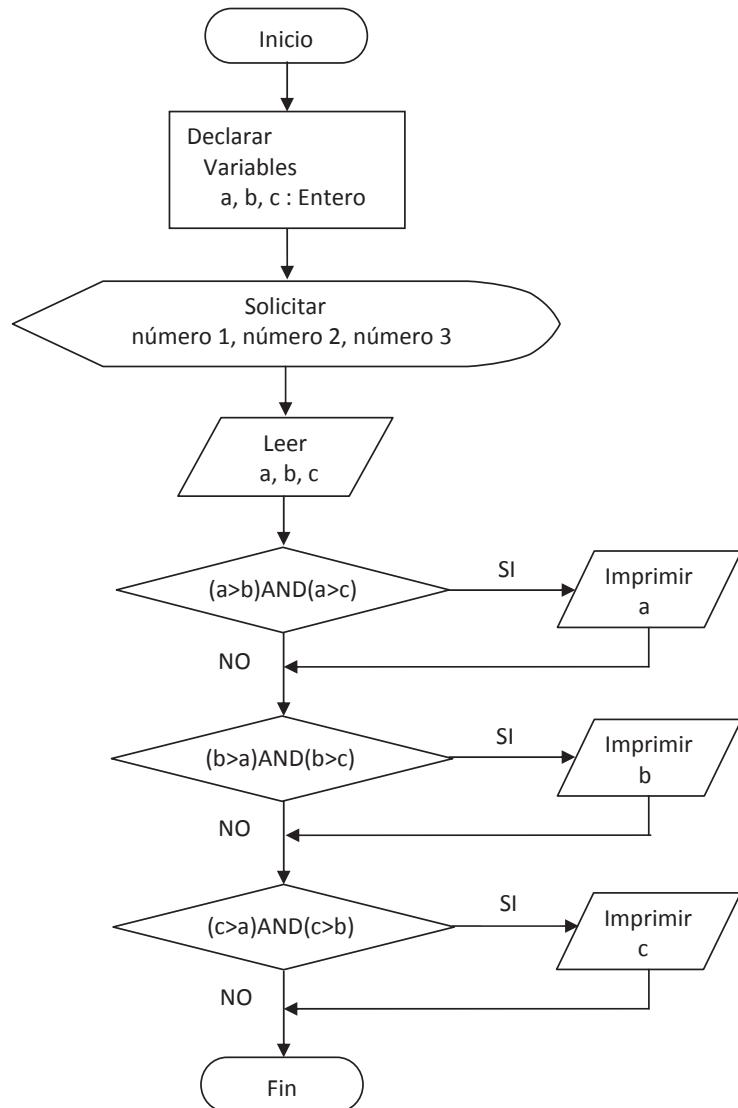
Ejercicio capítulo 4 (página 86).

### Algoritmo CÁLCULO SUELDO CON INCENTIVO



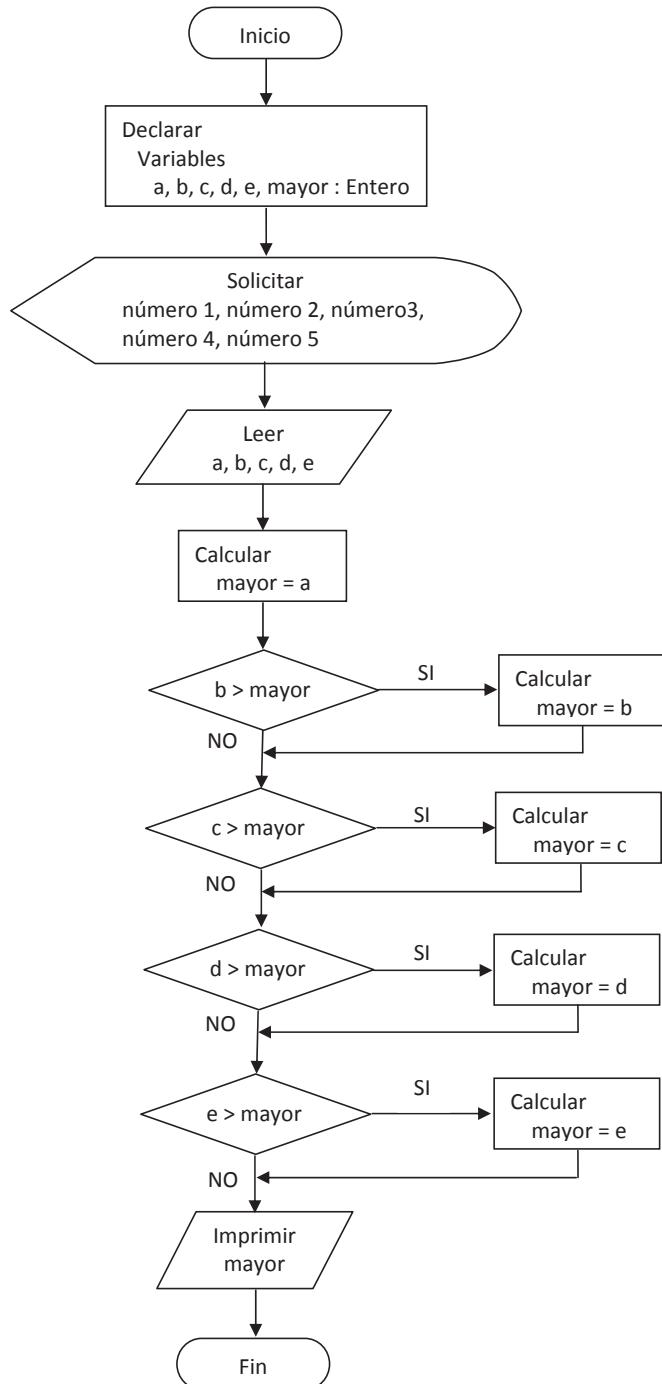
Ejercicio 4.2.1.1 (página 87). Tercer método de solución.

### Algoritmo MAYOR 3 NÚMEROS



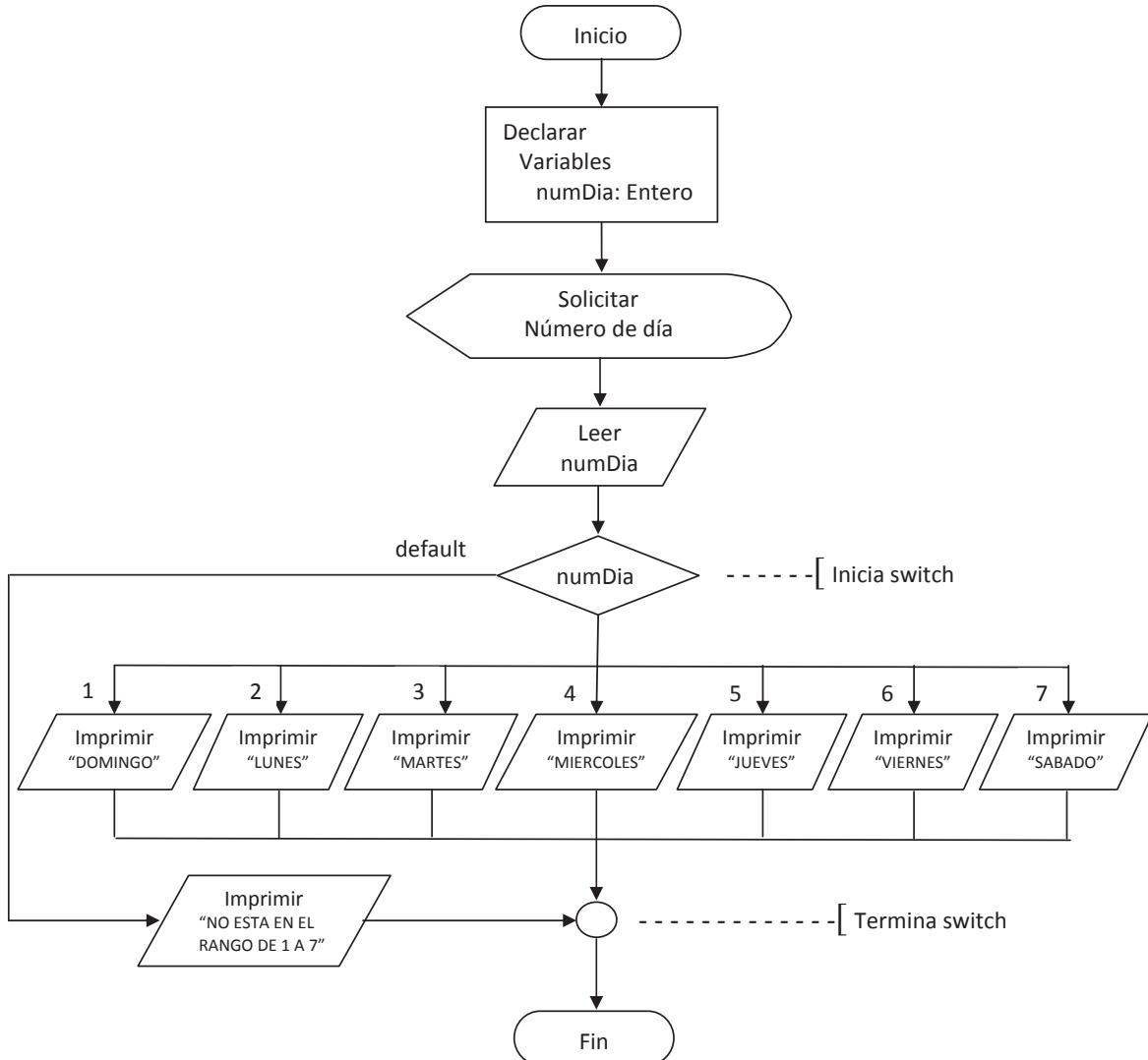
Ejercicio 4.2.1.2 (página 88).

### Algoritmo MAYOR 5 NÚMEROS



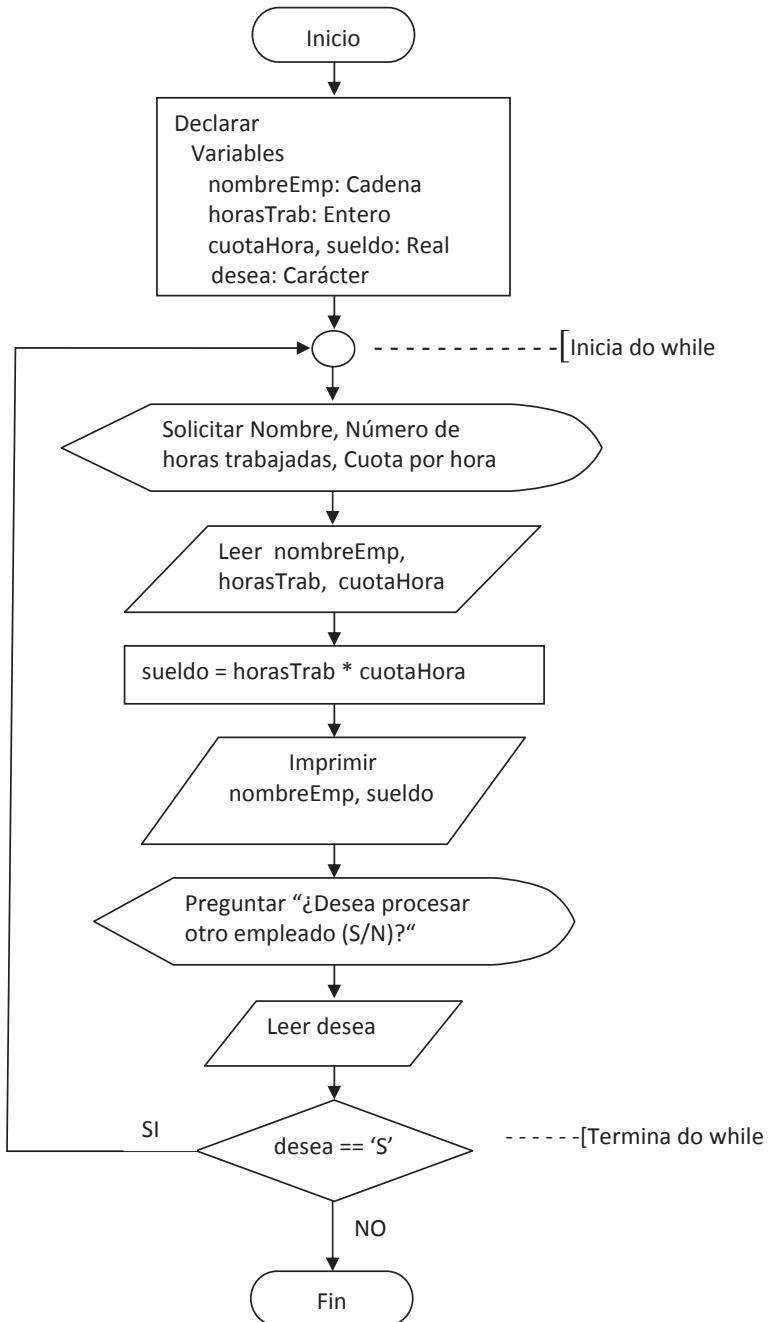
Ejercicio capítulo 4 (página 91).

### Algoritmo DICE DÍA



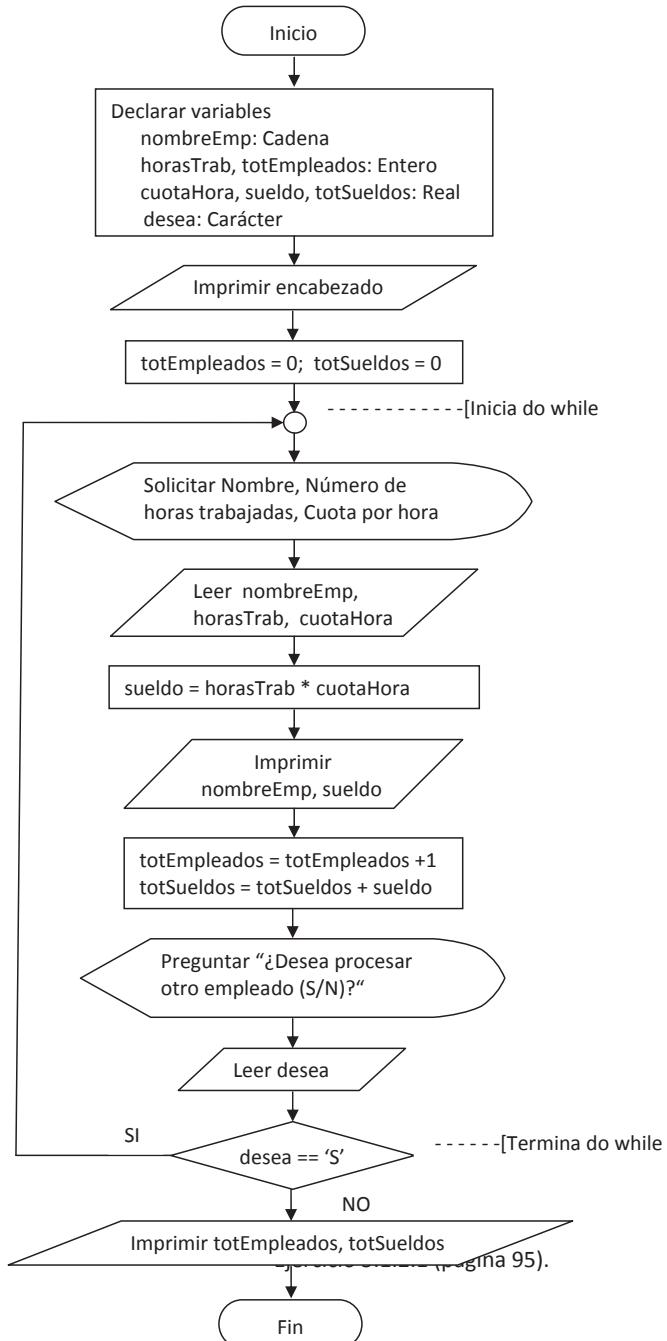
Primer ejemplo capítulo 5 (página 104).

### Algoritmo CALCULA SUELDOS DE EMPLEADOS



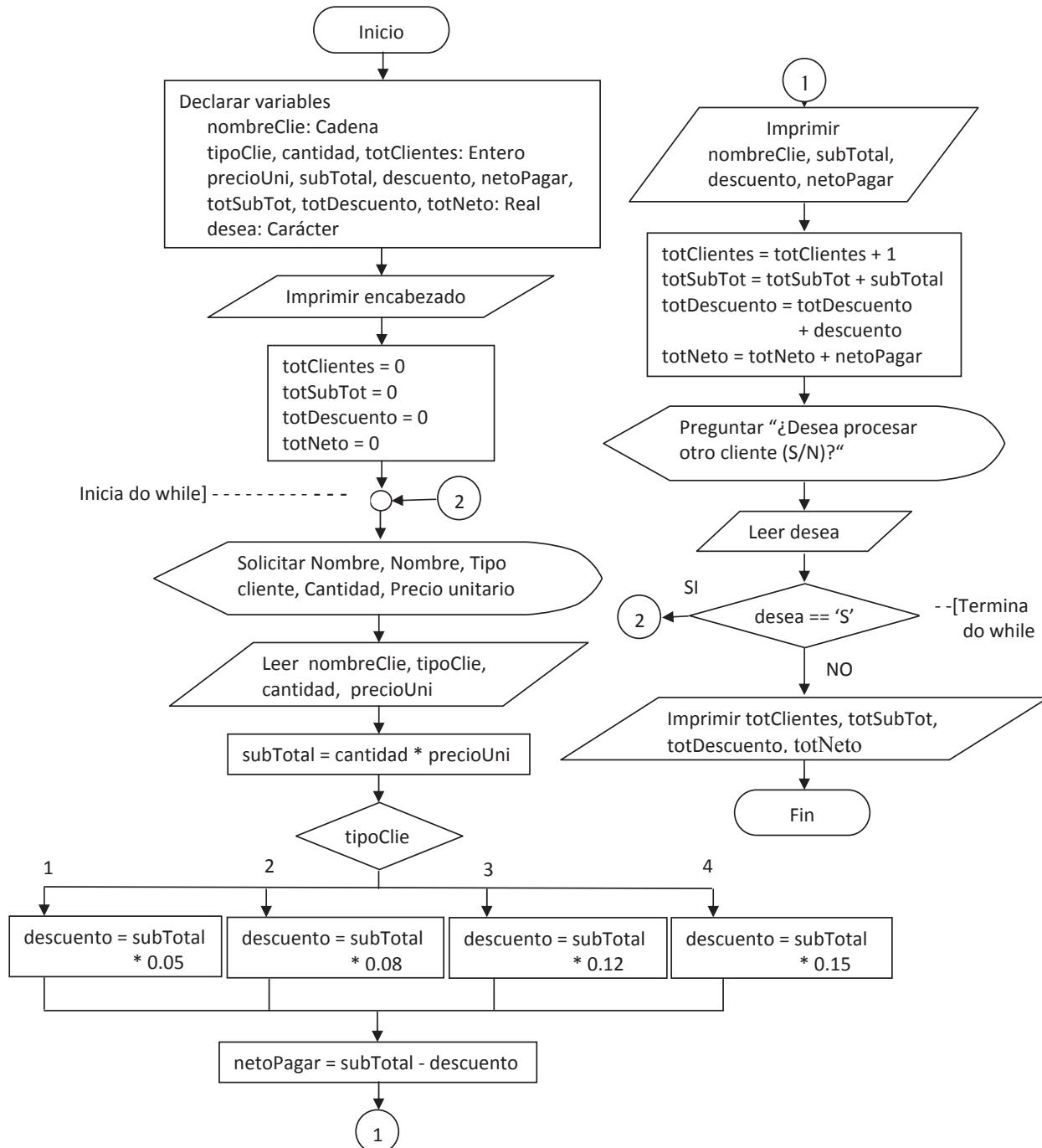
Segundo ejemplo capítulo 5 (página 106).

### Algoritmo CALCULA SUELDOS DE EMPLEADOS



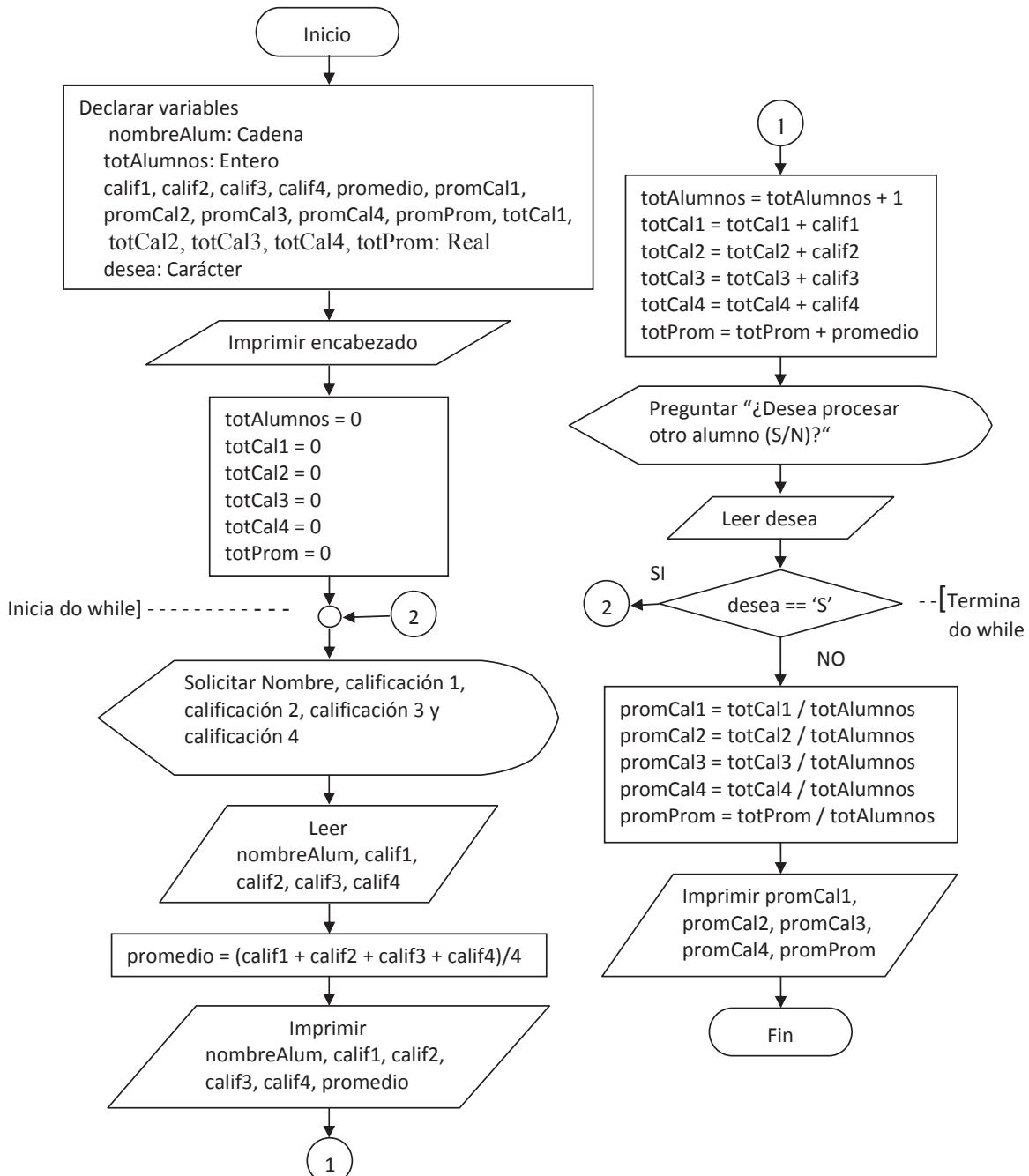
Ejercicio 5.1.2.1 (página 109).

### Algoritmo CLIENTES HOJAS HIELO SECO



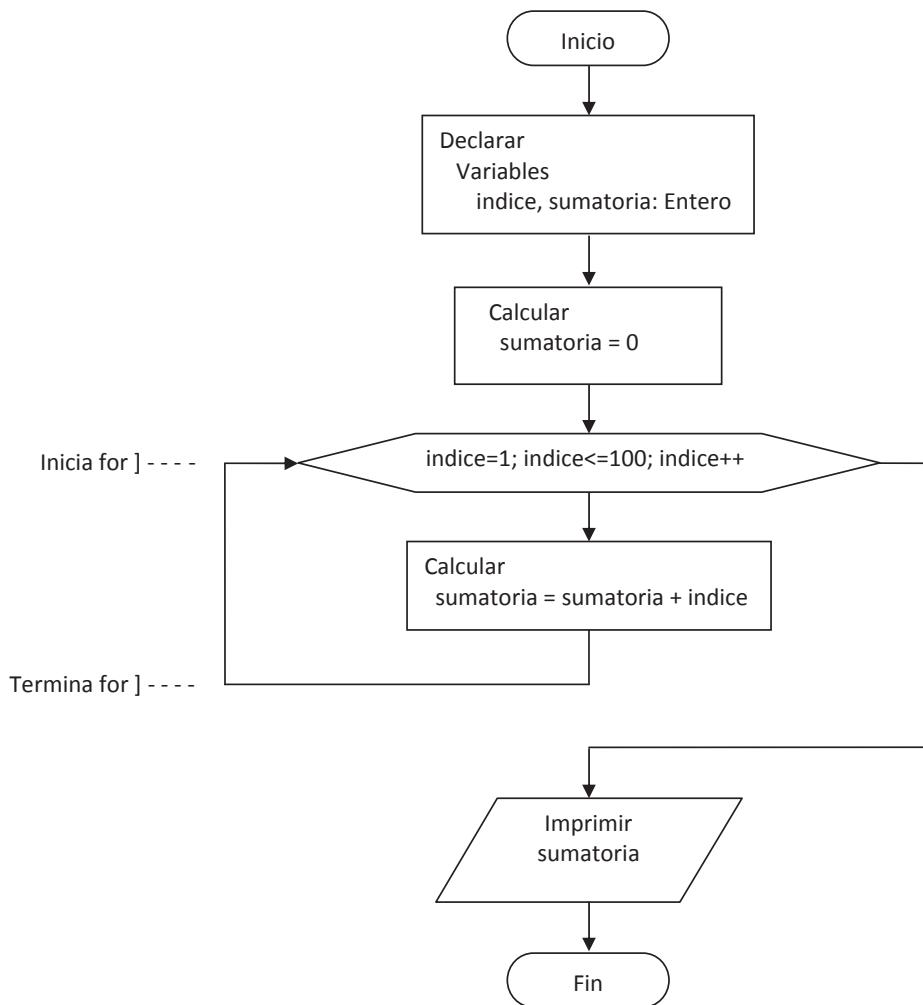
Ejercicio 5.1.2.2 (página 111).

### Algoritmo CALIFICACIONES DE ALUMNOS



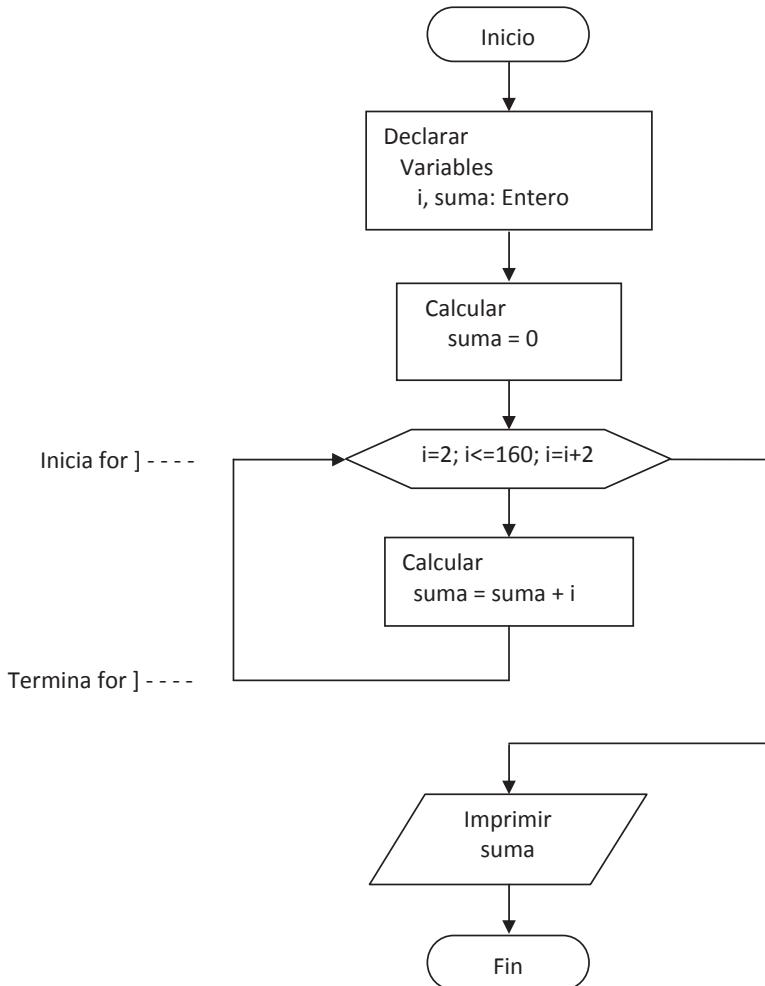
Ejercicio capítulo 5 (página 126).

### Algoritmo SUMA NÚMEROS 1-100



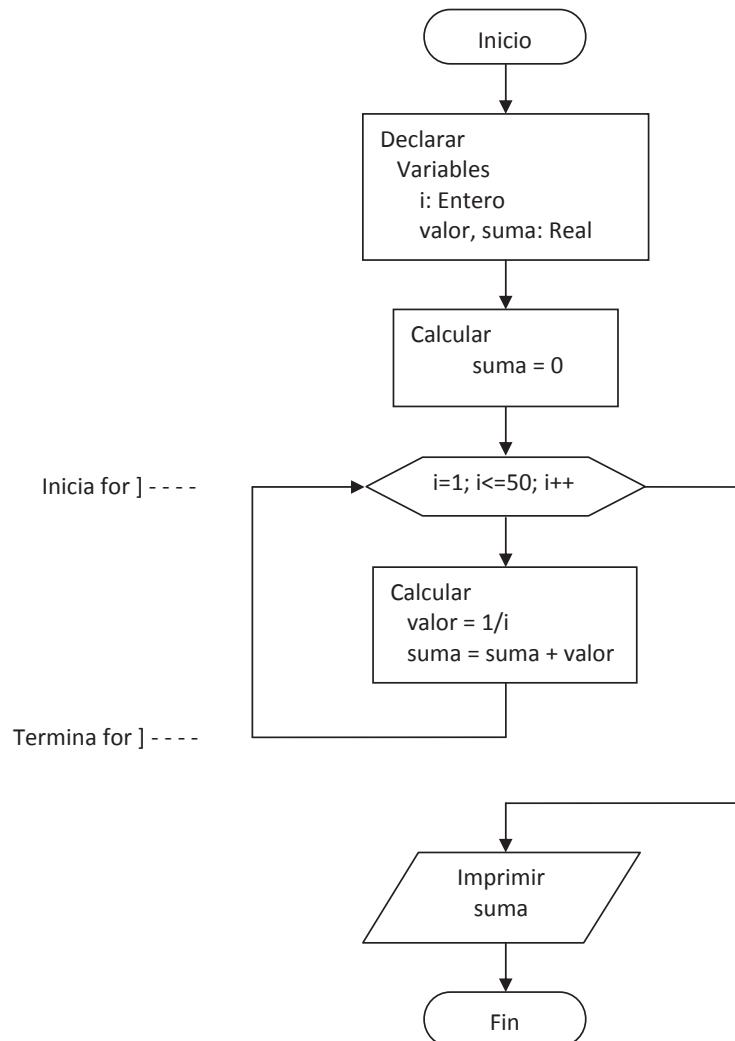
Ejercicio 5.3.2.1 (página 127).

### Algoritmo SUMA NÚMEROS 2-160



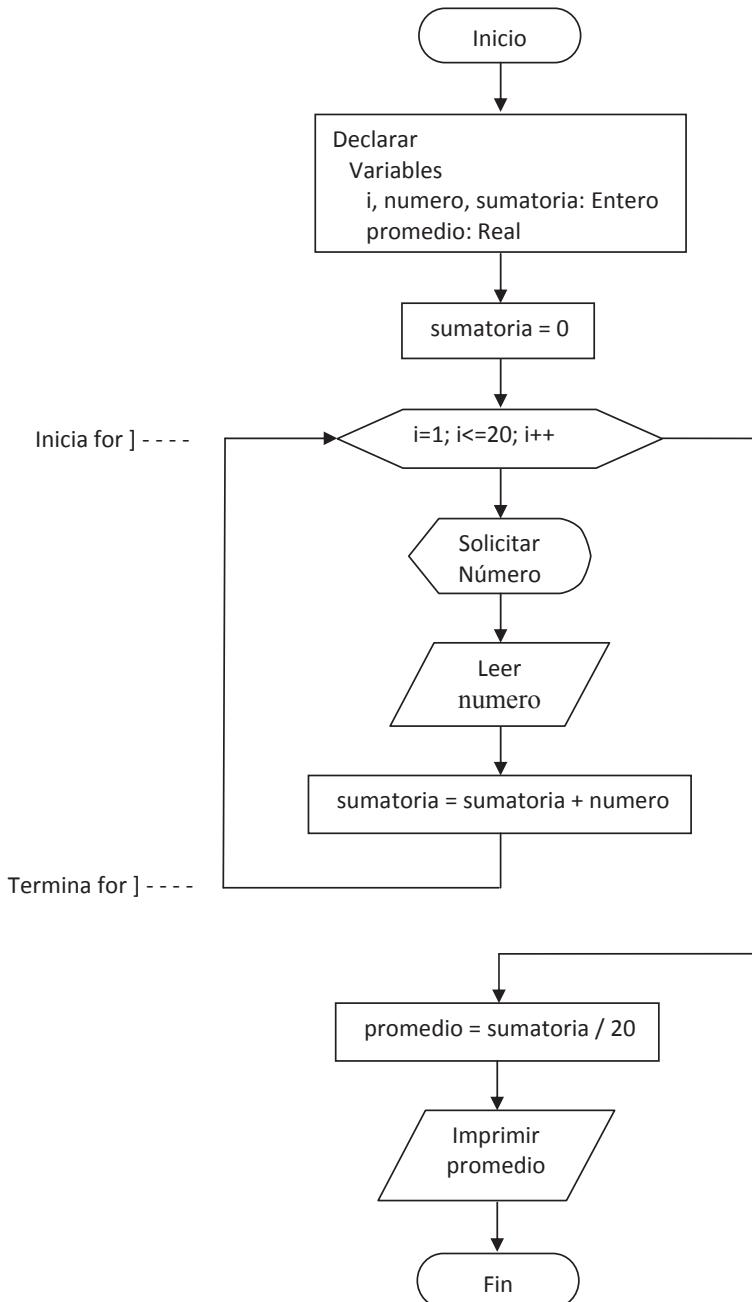
Ejercicio 5.3.2.2 (página 128).

### Algoritmo SUMATORIA



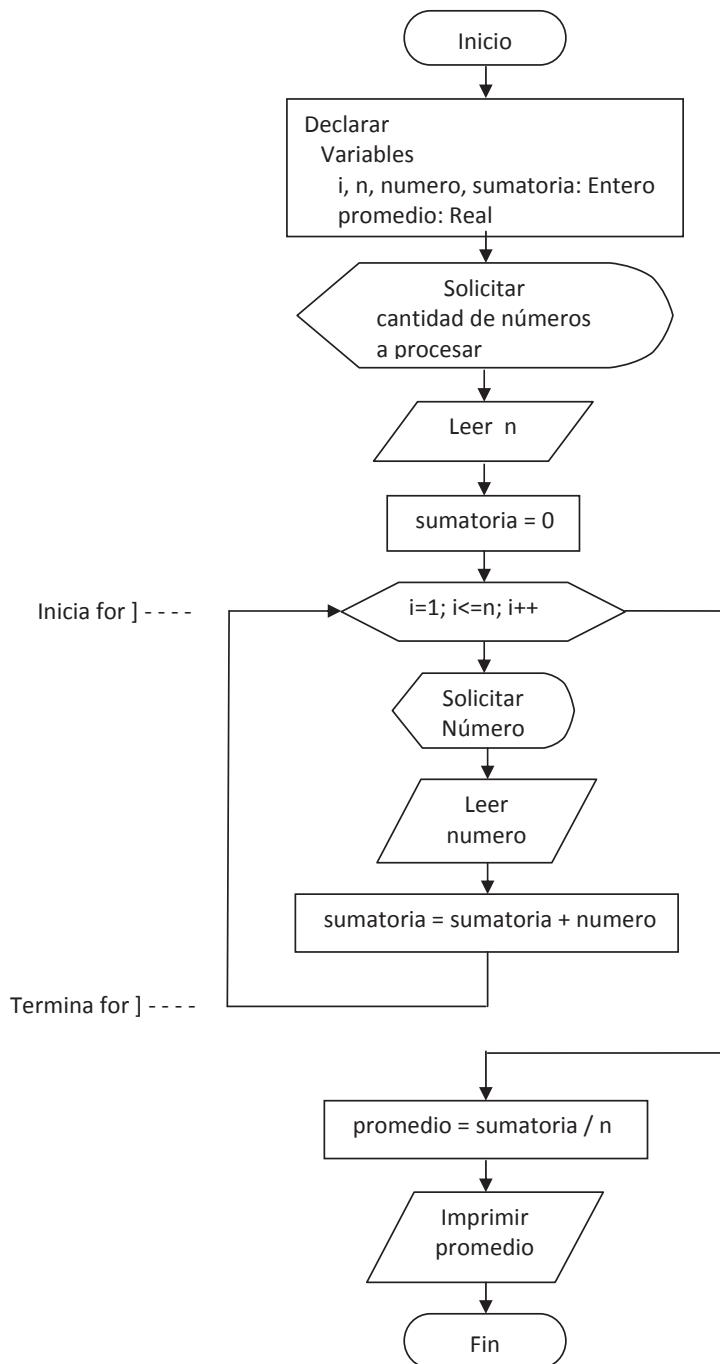
Ejercicio 5.3.2.3 (página 129).

### Algoritmo PROMEDIO DE 20 NÚMEROS



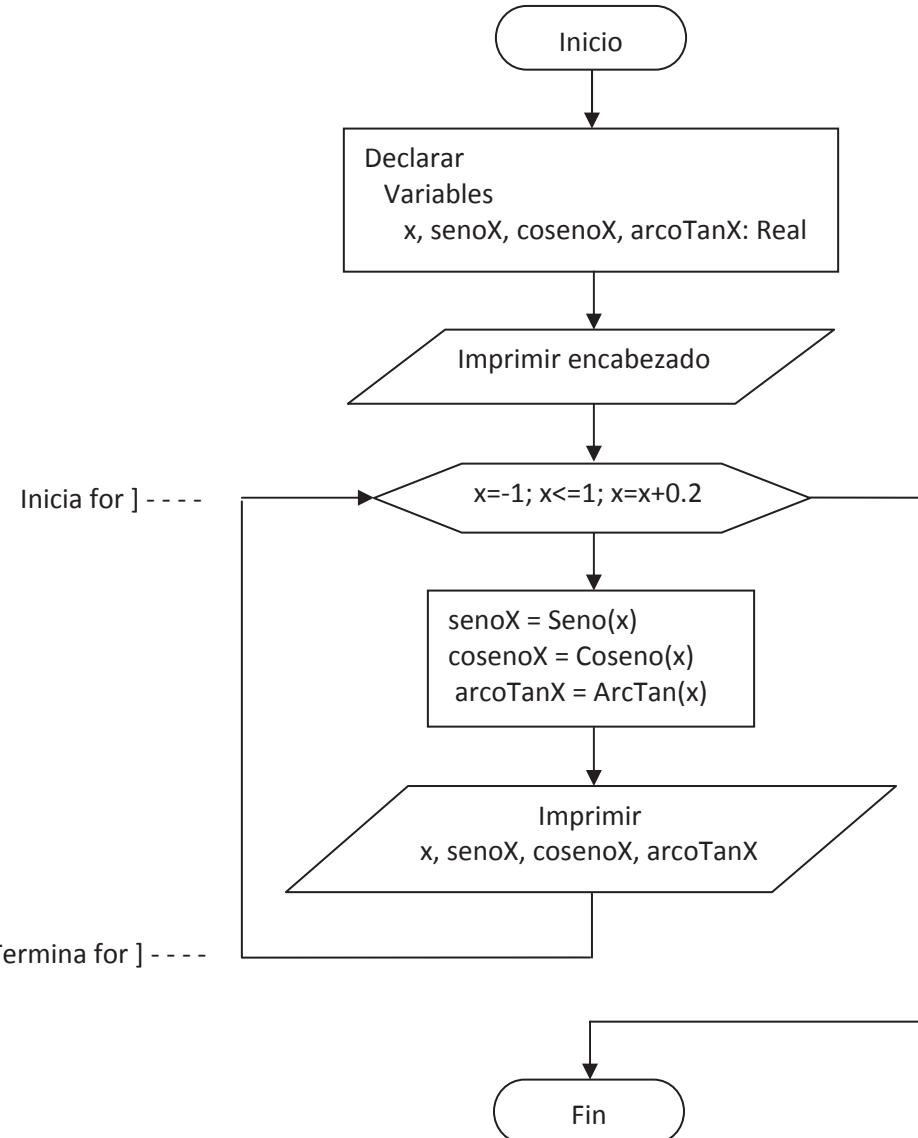
Ejercicio 5.3.2.4 (página 130).

### Algoritmo PROMEDIO DE N NÚMEROS



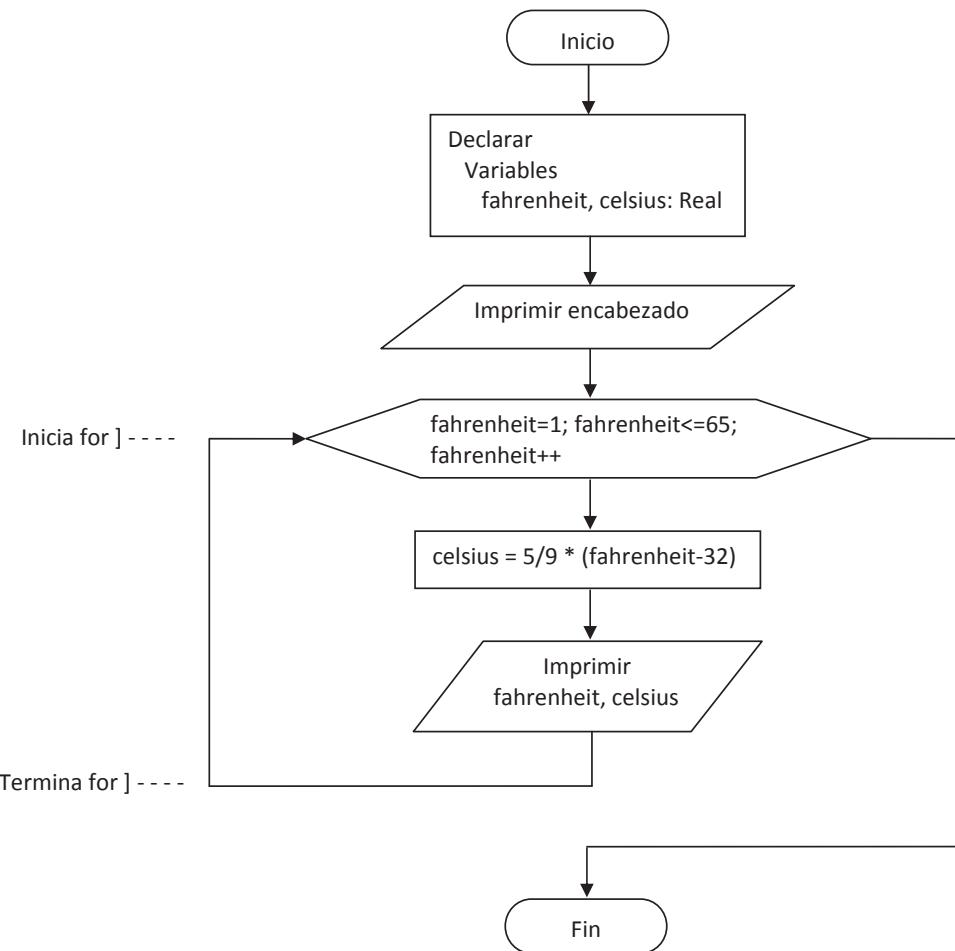
Ejercicio 5.3.2.5 (página 131).

### Algoritmo SENO COSENO ARCO TANGENTE DE -1 HASTA 1



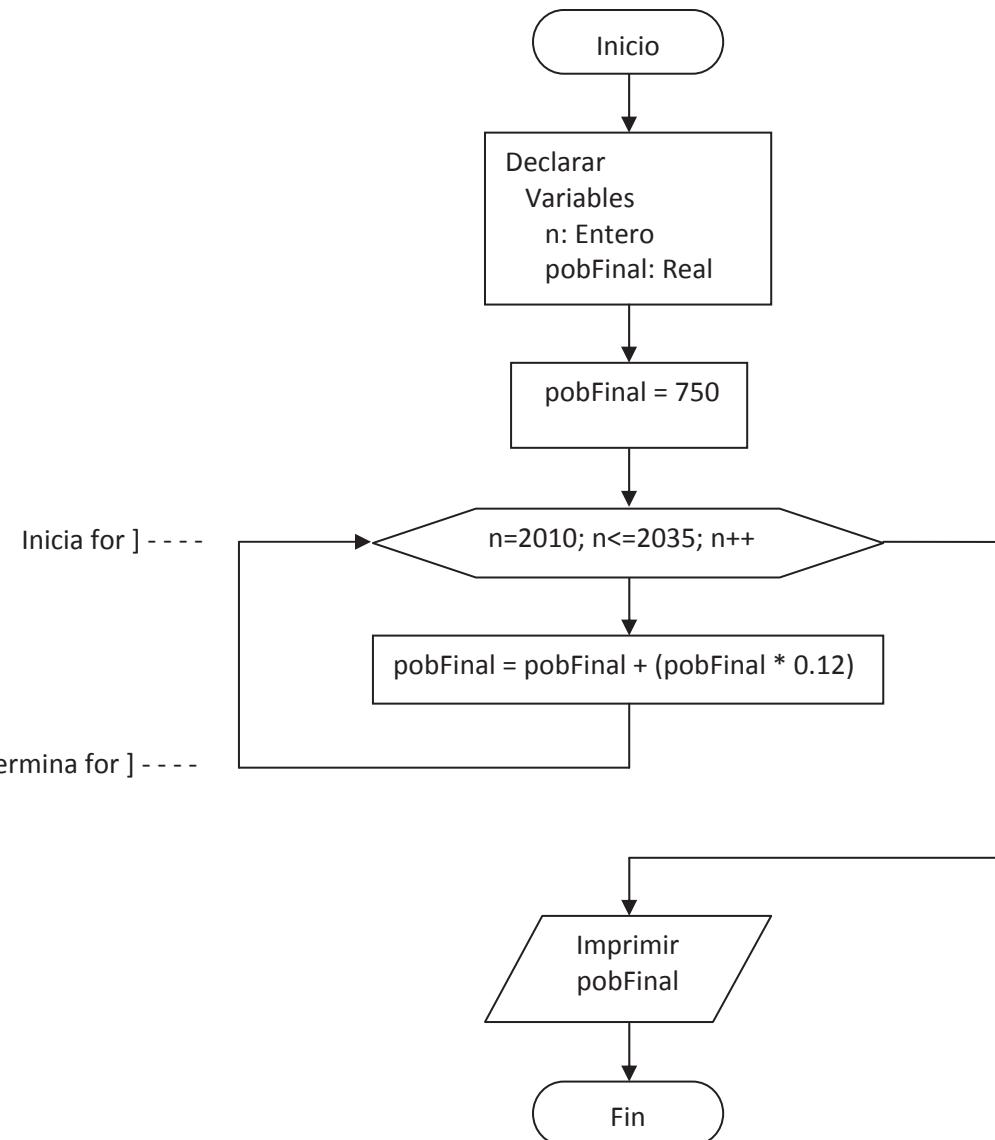
Ejercicio 5.3.2.6 (página 131).

### Algoritmo EQUIVALENCIAS FAHRENHEIT CELSIUS



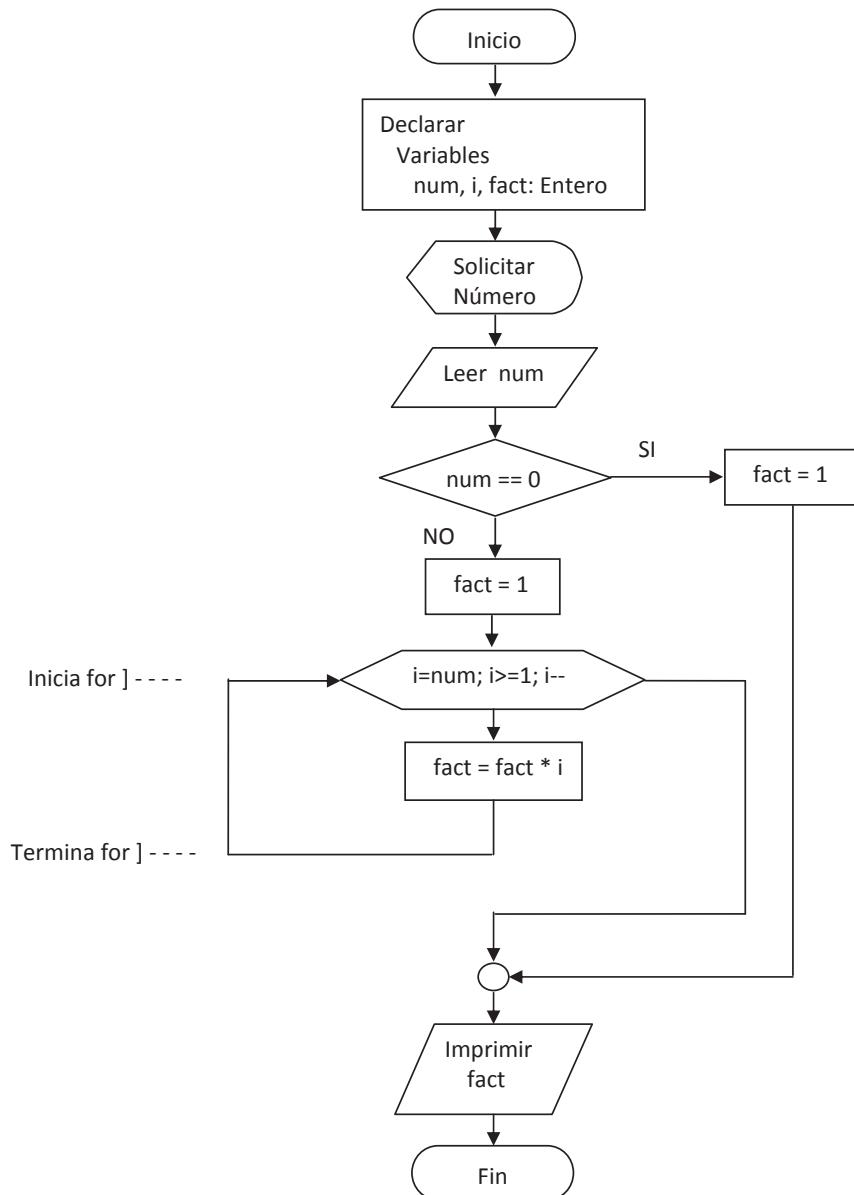
Ejercicio 5.3.2.7 (página 132).

### Algoritmo ESTIMAR POBLACIÓN



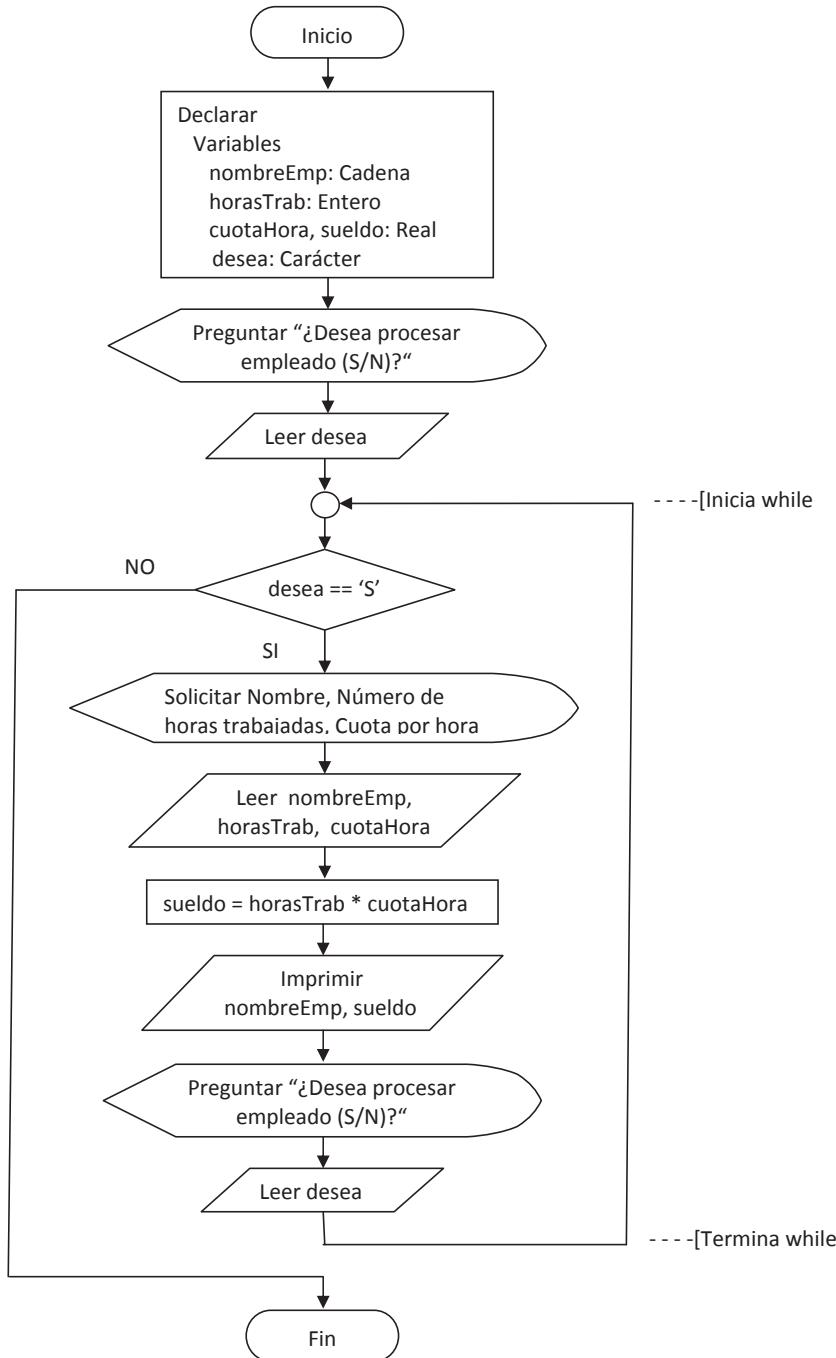
Ejercicio 5.3.2.8 (página 133).

### Algoritmo FACTORIAL



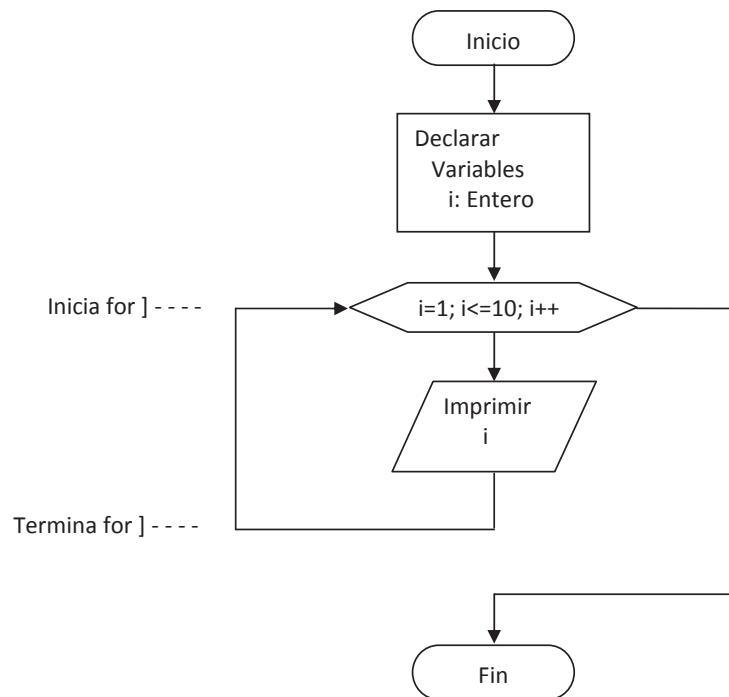
Ejemplo capítulo 5 (página 146).

### Algoritmo CALCULA SUELDOS DE EMPLEADOS



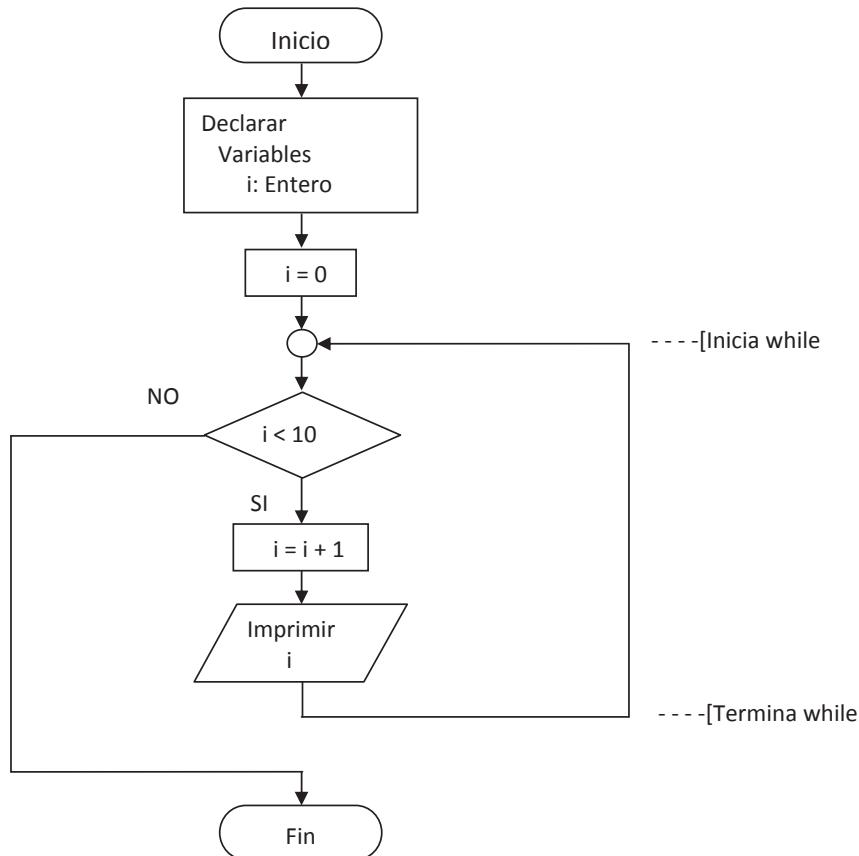
Ejemplo del apartado 5.5.2 Simulación del for con while (página 147).

### Algoritmo IMPRIME 1-10



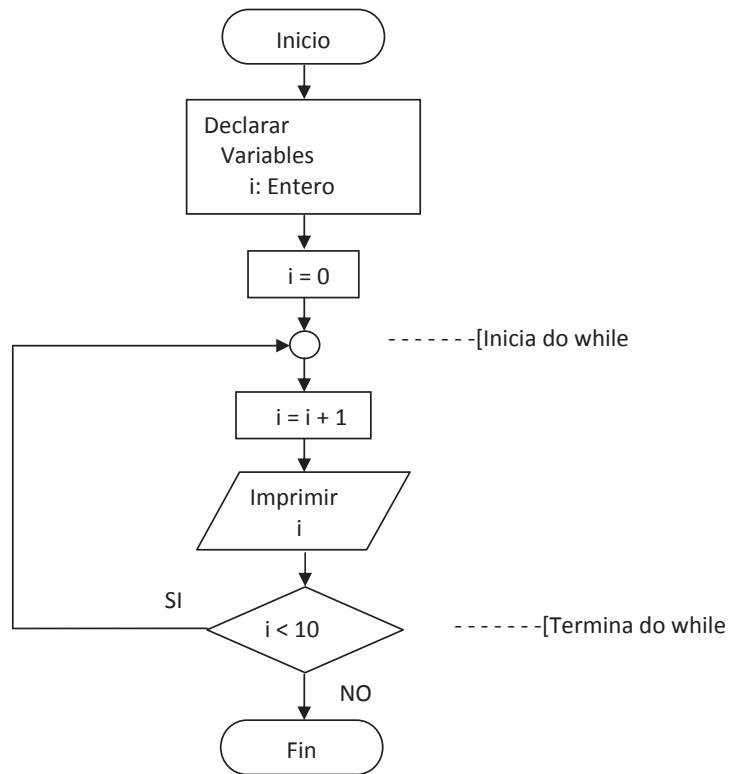
Ejemplo del apartado 5.5.2 Simulación del for con while (página 148).

### Algoritmo IMPRIME 1-10



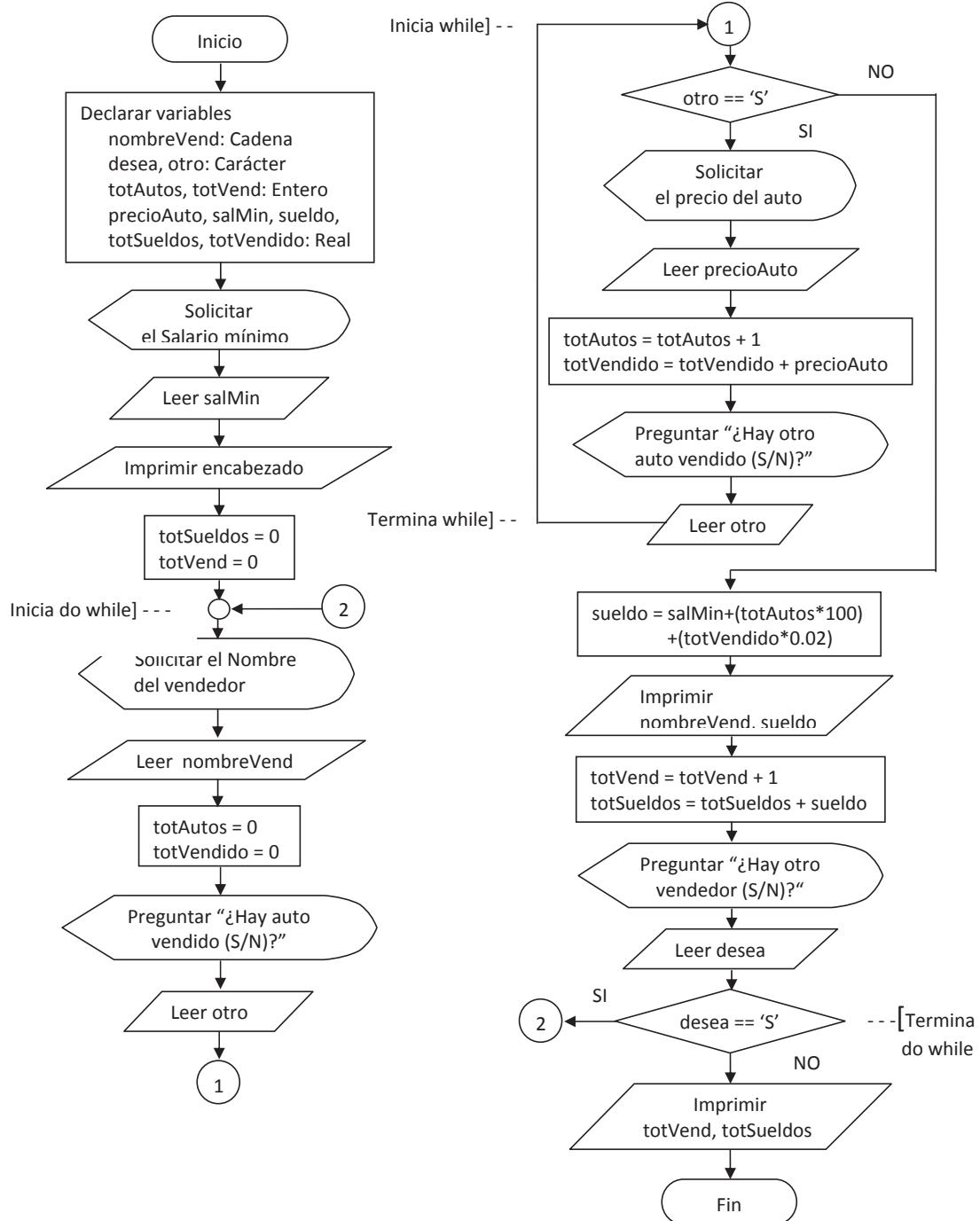
Ejercicio 5.5.2 Simulación del for con do while (página 148).

### Algoritmo IMPRIME 1-10



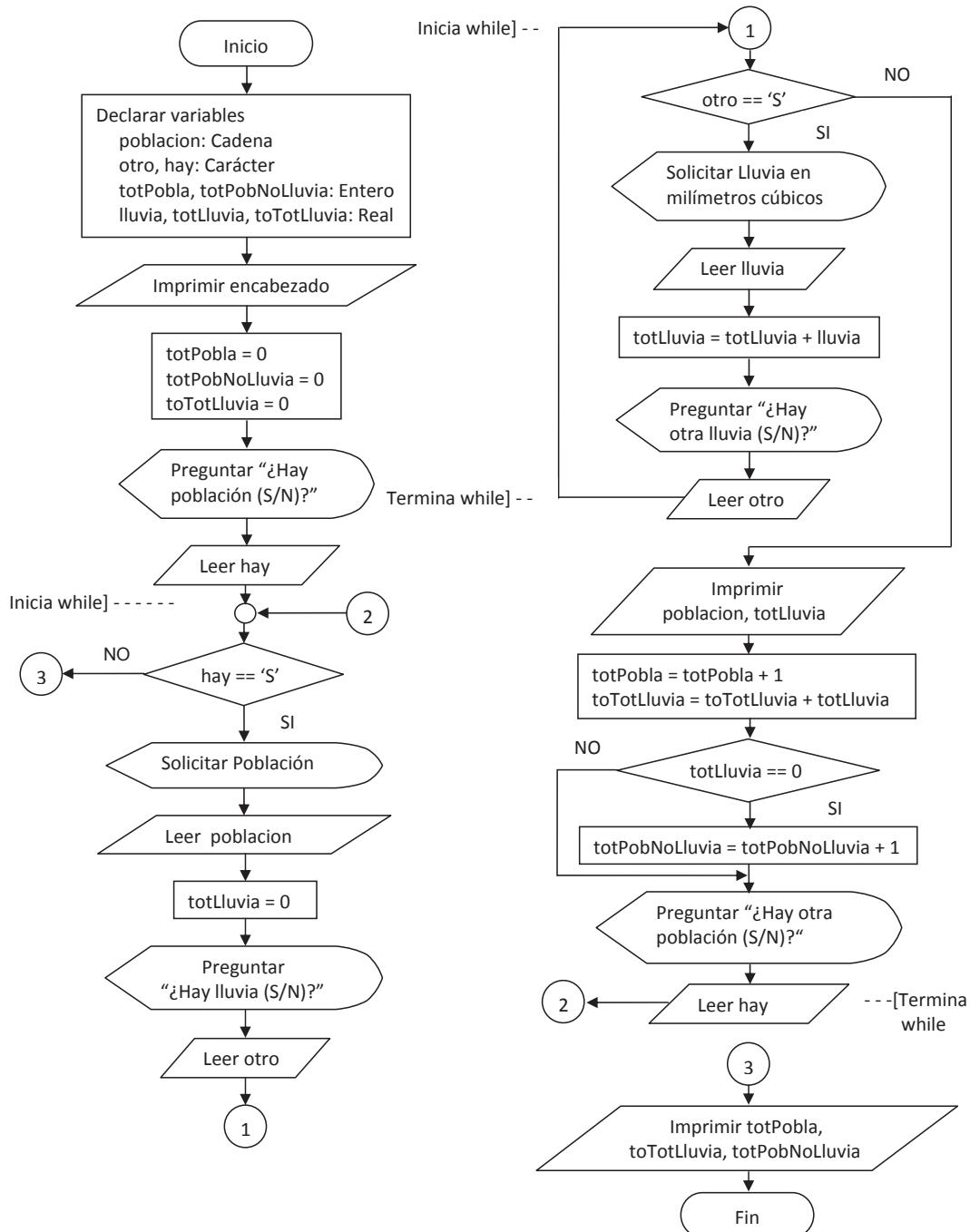
Ejercicio 5.5.3.1 (página 150).

### Algoritmo VENDEDORES DE AUTOMÓVILES



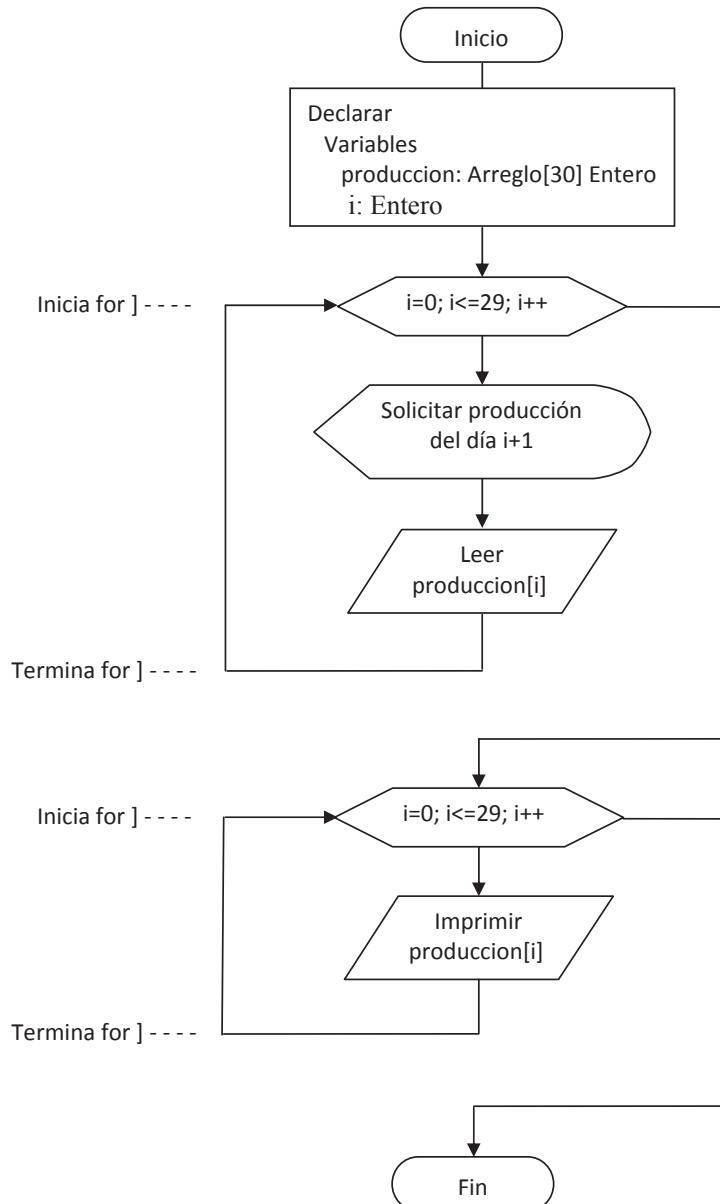
Ejercicio 5.5.3.2 (página 152).

### Algoritmo LLUVIAS



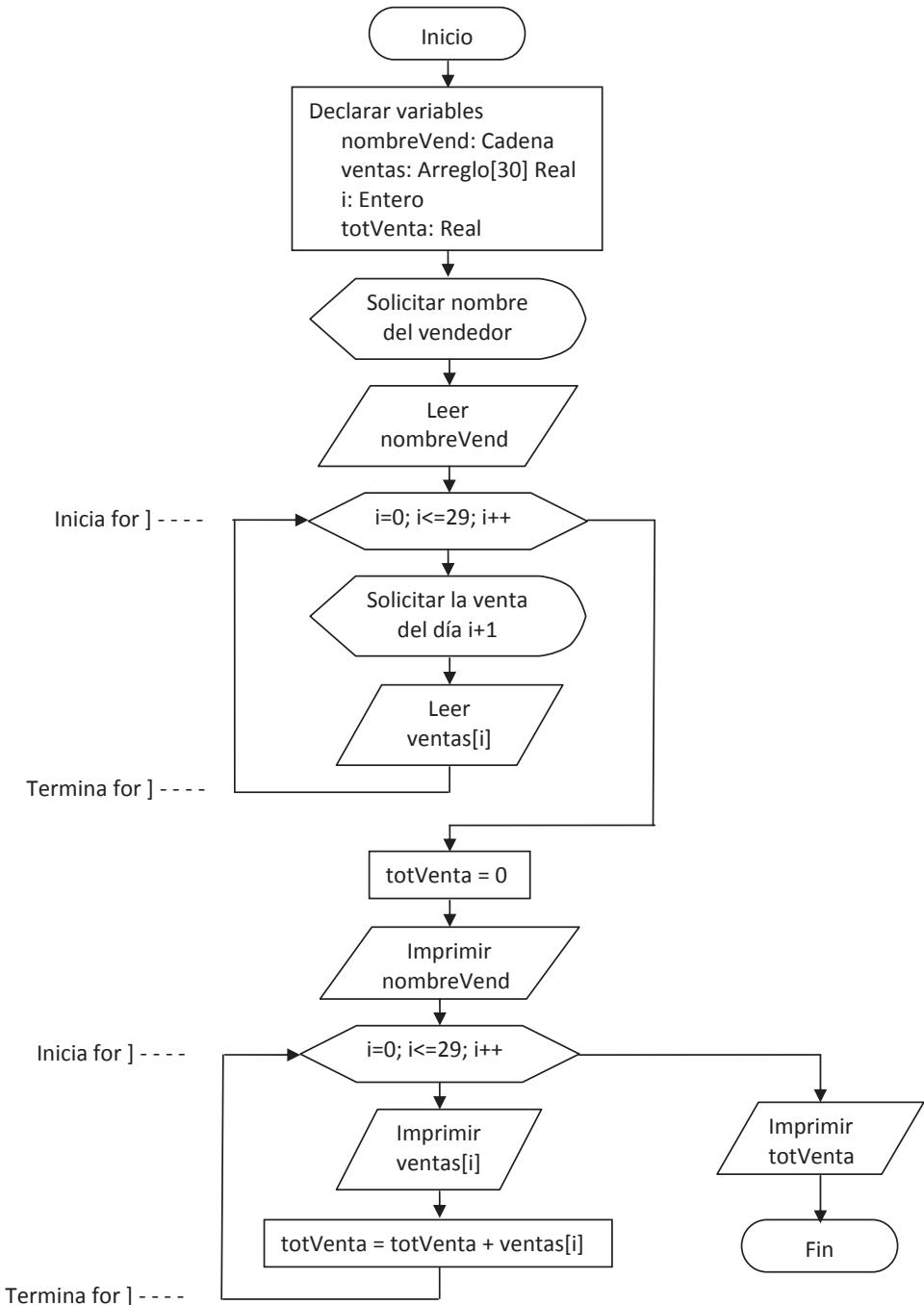
Primer ejemplo capítulo 6 (página 164).

### Algoritmo PRODUCCIÓN 30 DÍAS



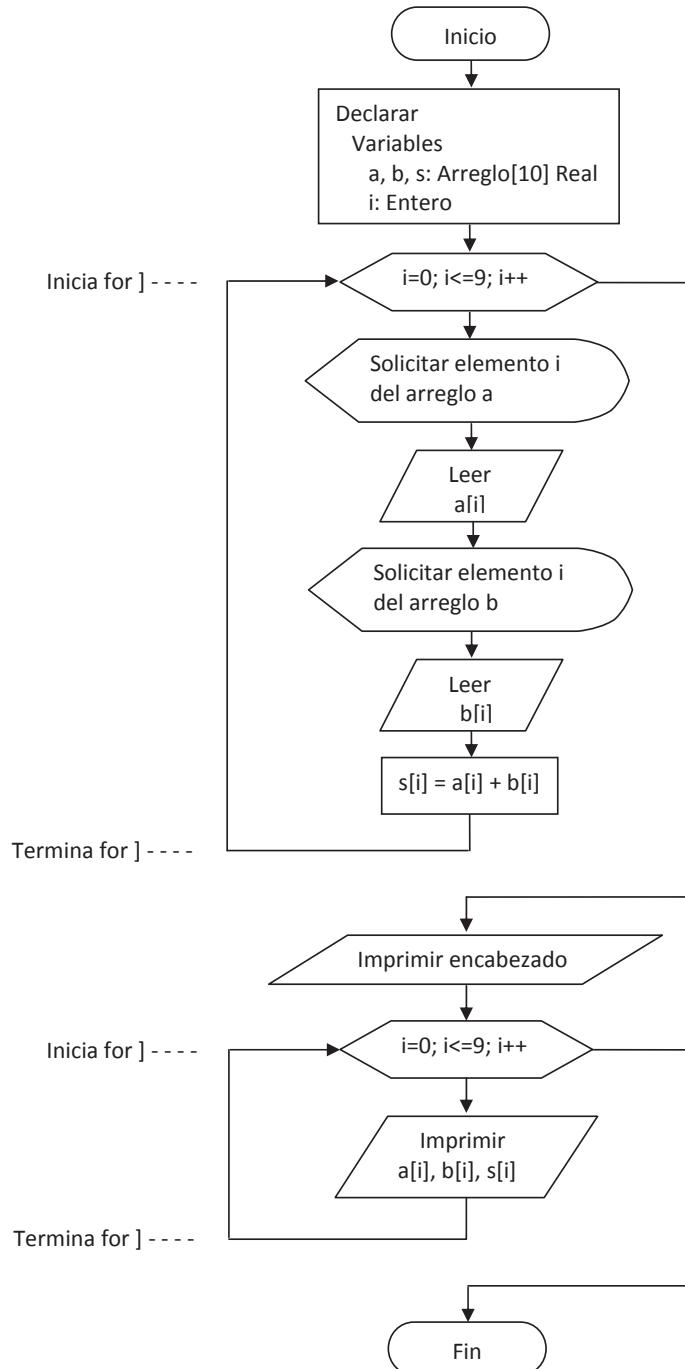
Ejercicio 6.1.1.1 (página 165).

### Algoritmo VENTAS MES



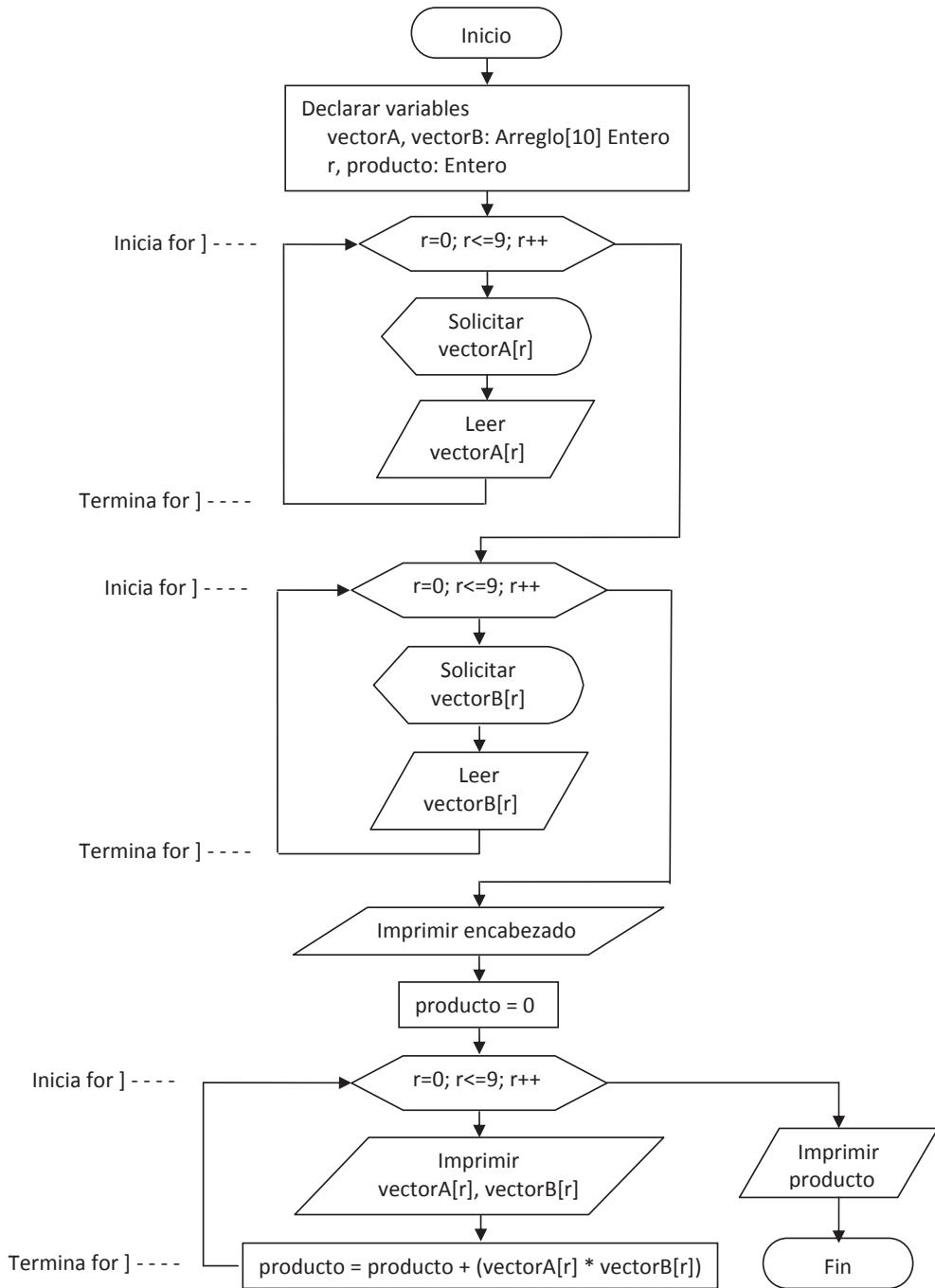
Ejercicio 6.1.1.2 (página 166).

### Algoritmo SUMA ARREGLOS



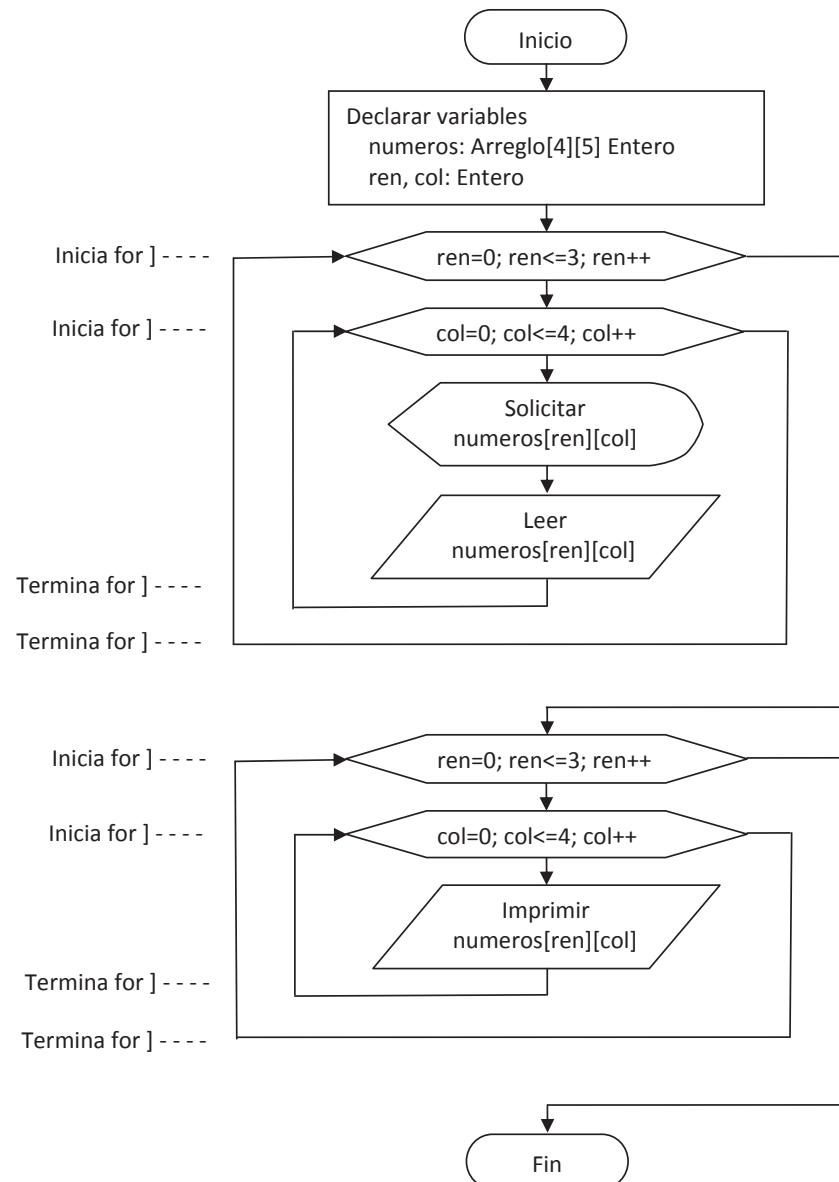
Ejercicio 6.1.1.3 (página 168).

### Algoritmo PRODUCTO DE VECTORES



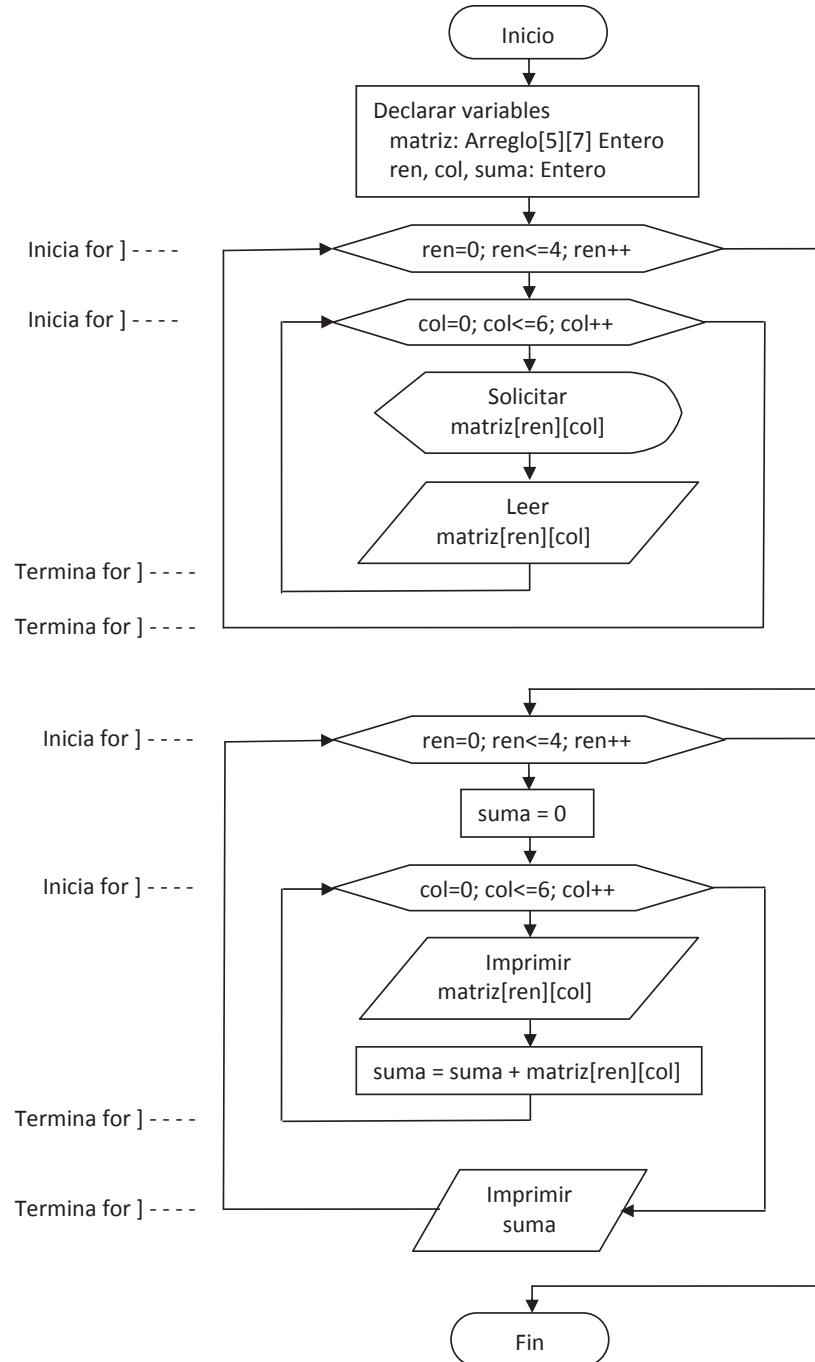
Primer ejemplo punto 6.2 (página 171).

### Algoritmo MATRIZ NÚMEROS



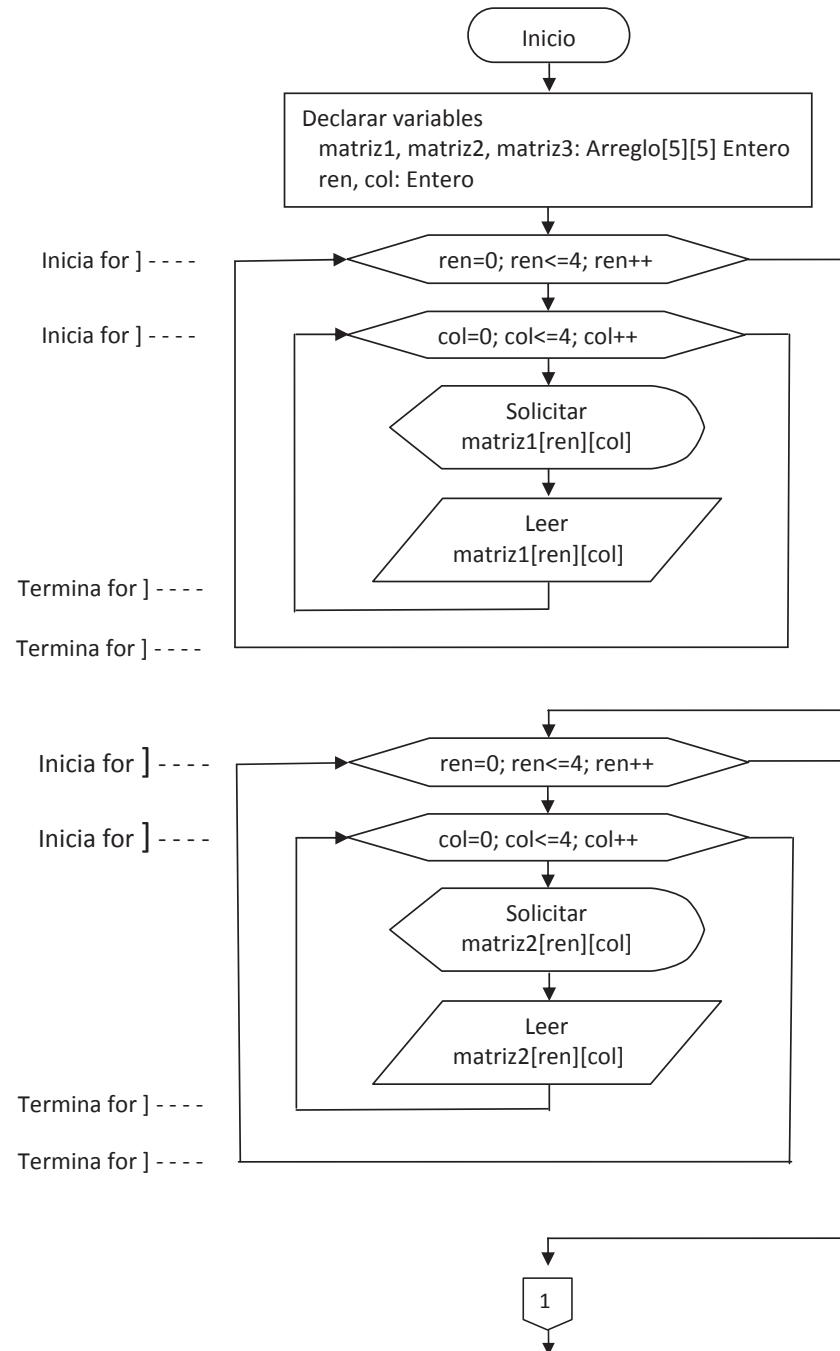
Ejercicio 6.2.1.1 (página 173).

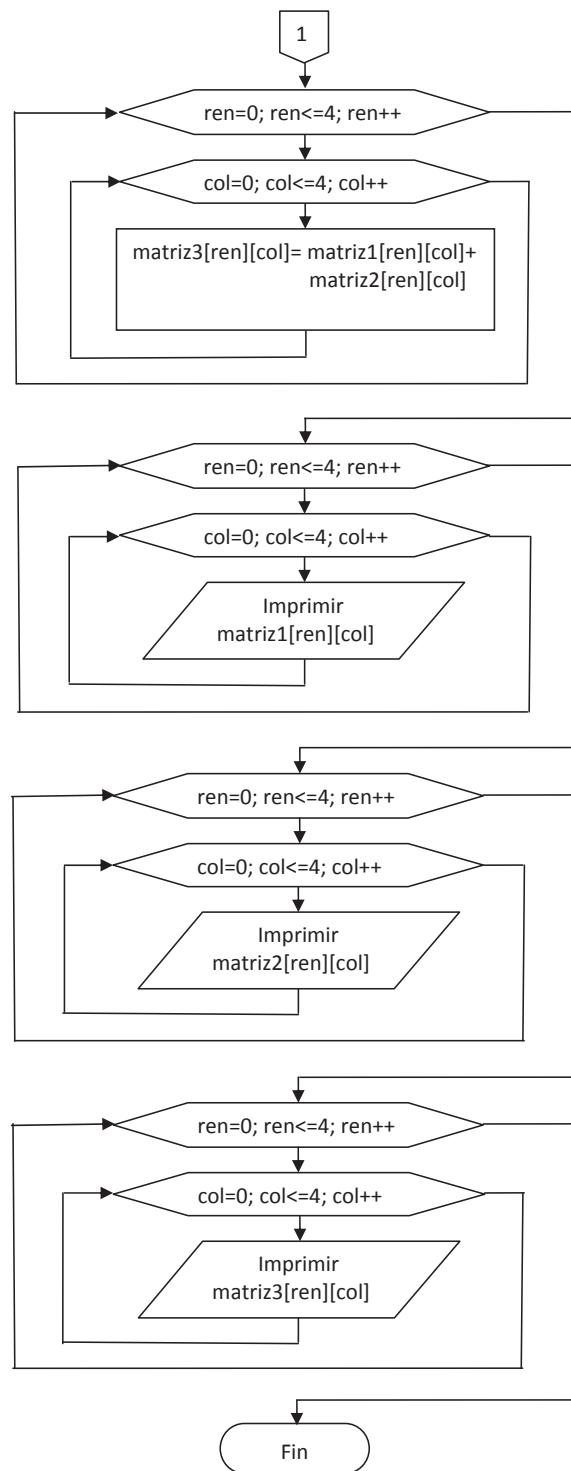
### Algoritmo SUMA POR RENGLONES



Ejercicio 6.2.1.2 (página 174).

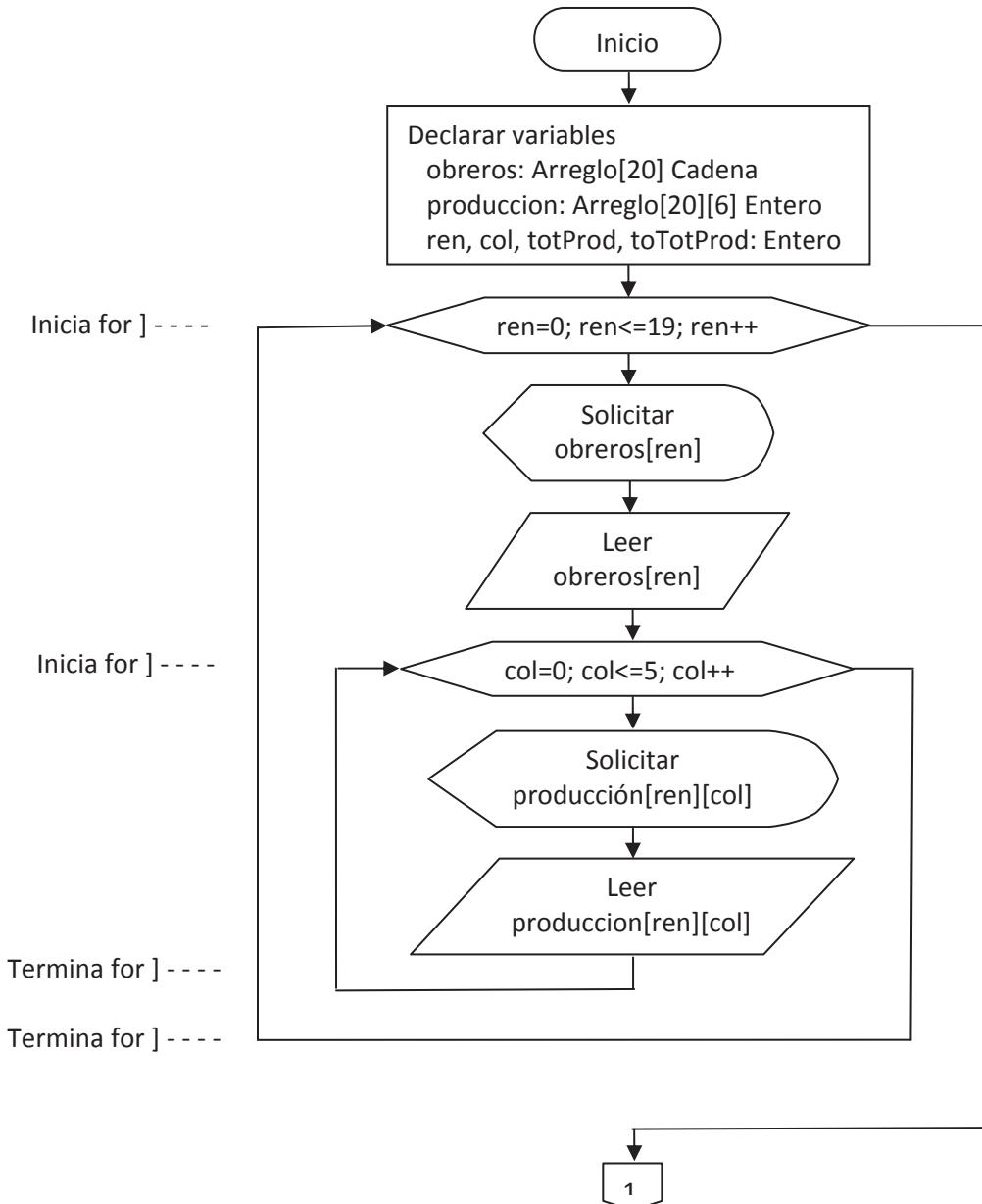
### Algoritmo SUMA MATRICES

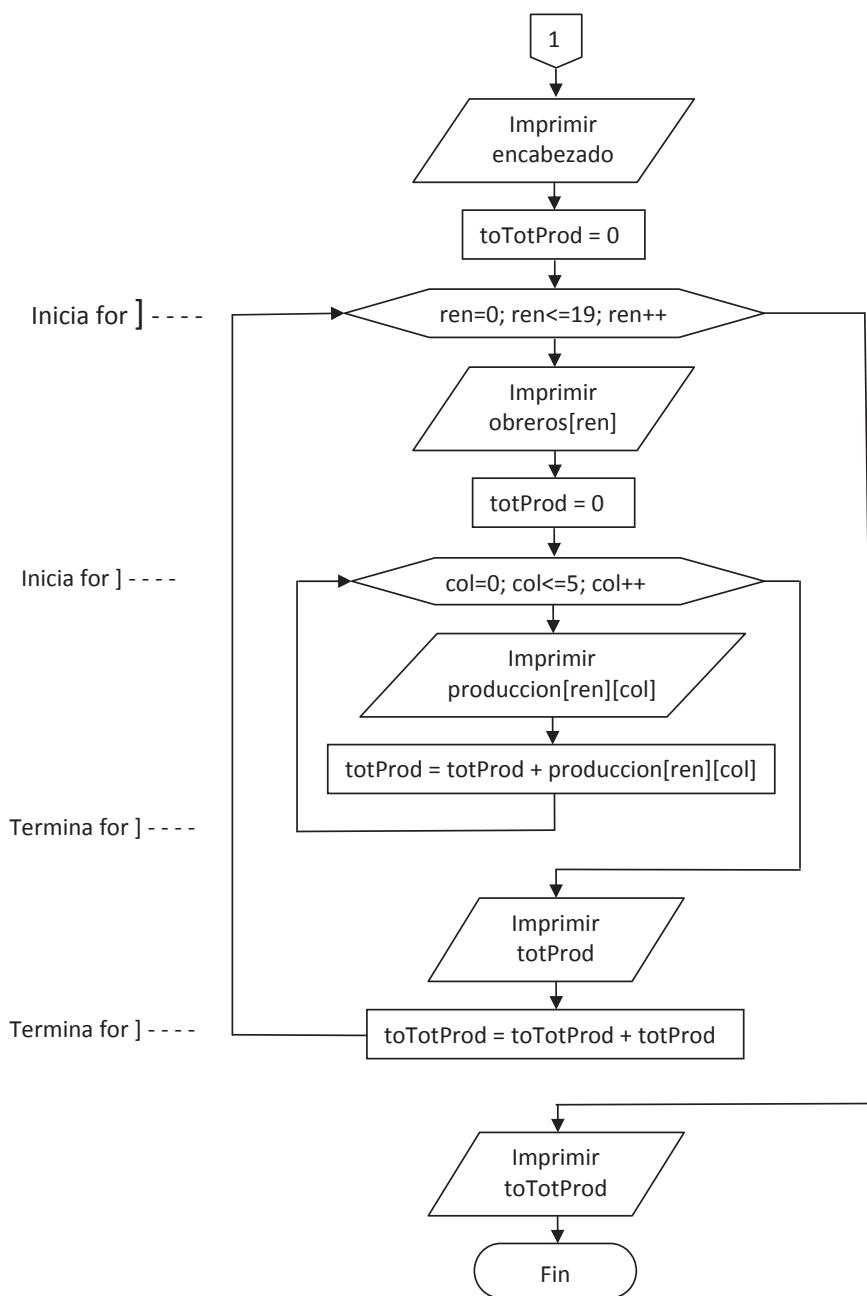




Ejercicio 6.2.1.3 (página 176).

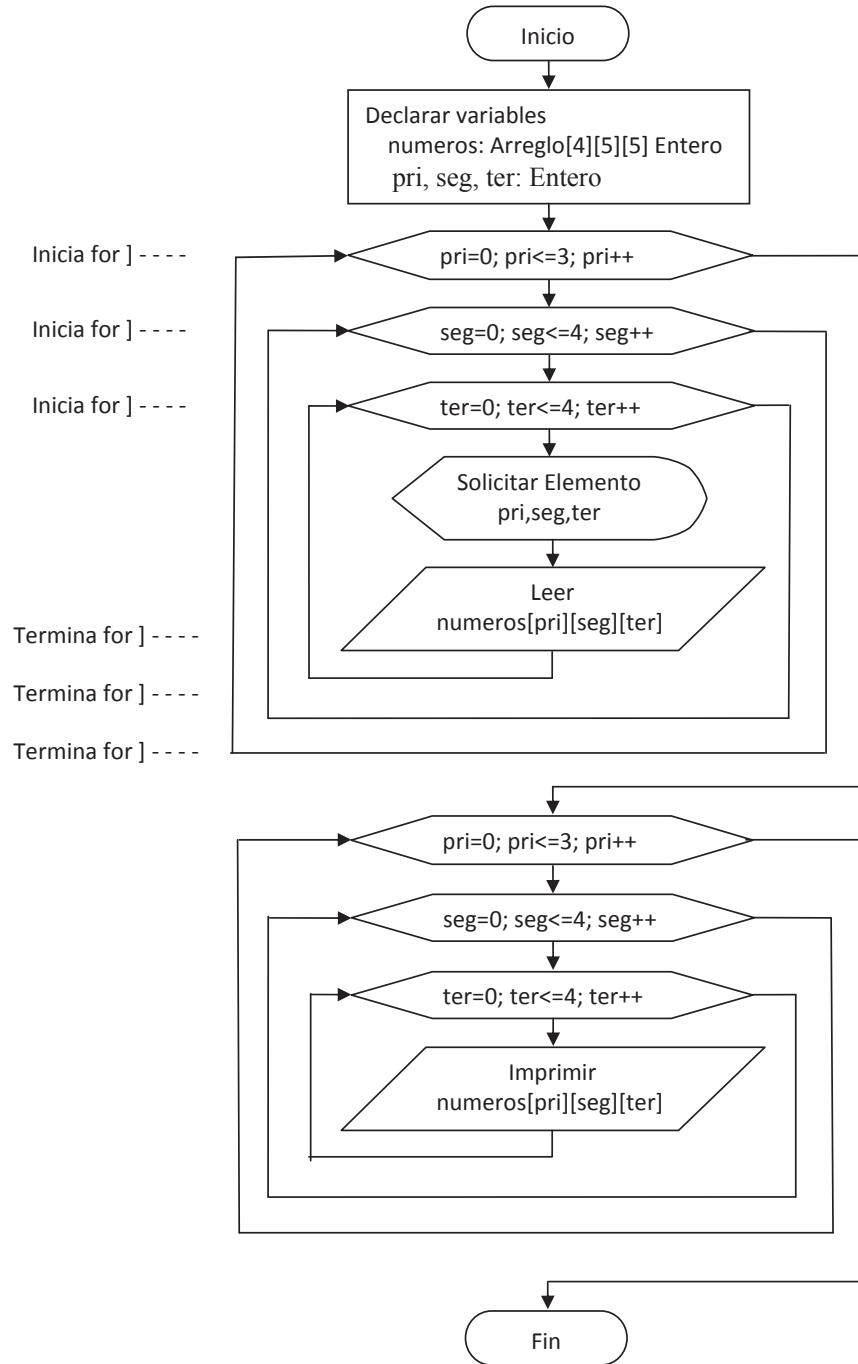
### Algoritmo PRODUCCIÓN 20 OBREROS





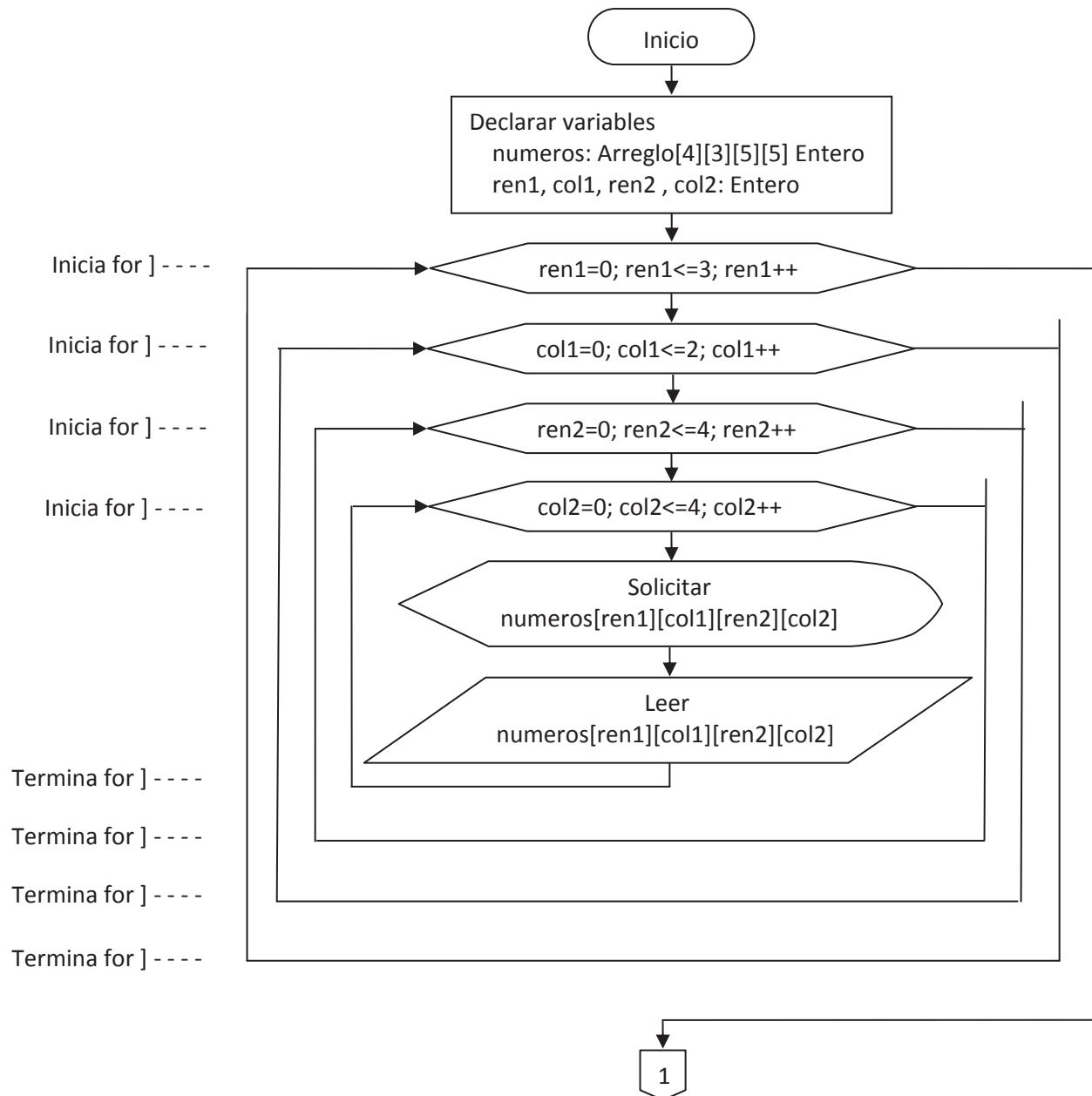
Primer ejemplo punto 6.3 (página 182).

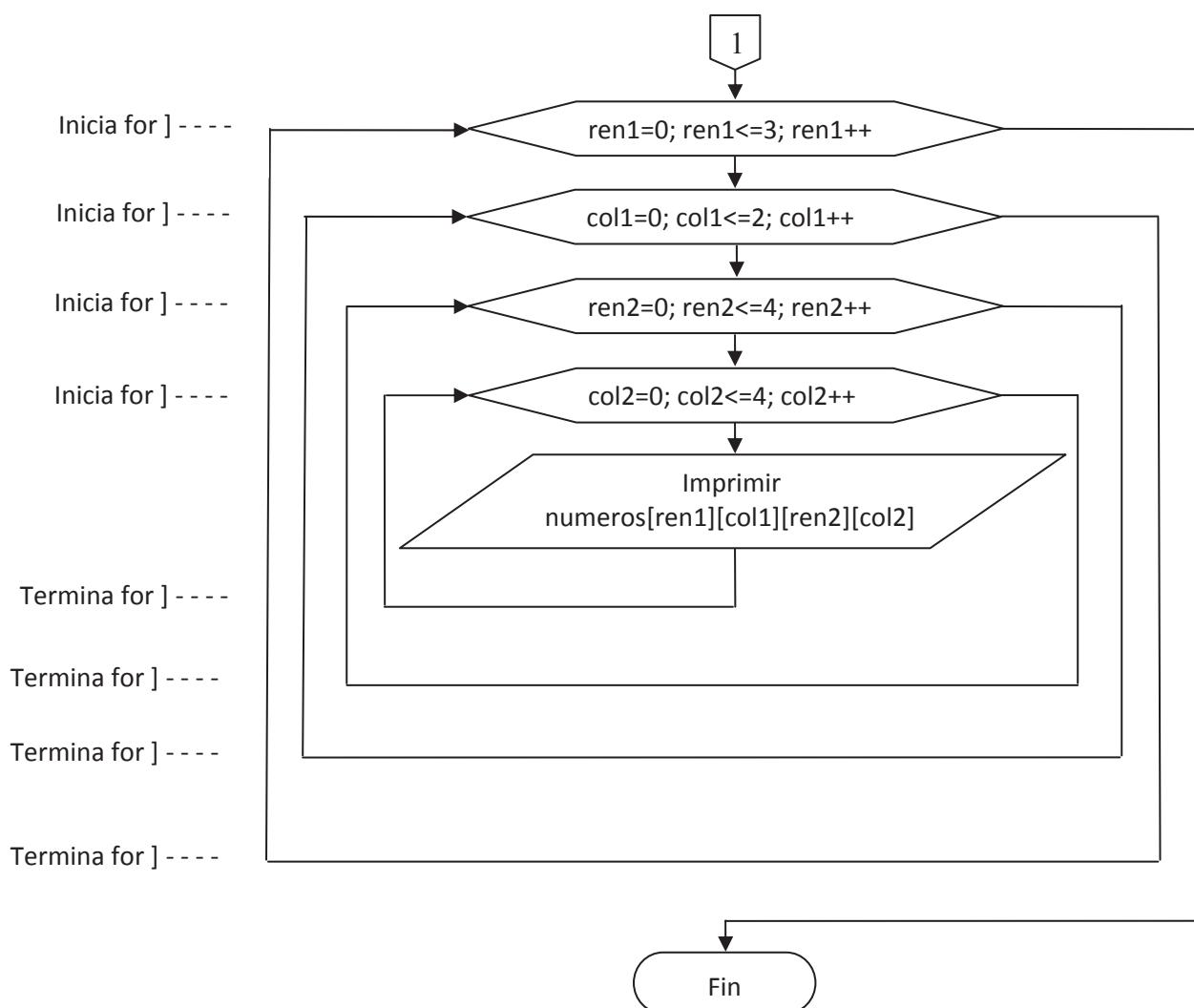
### Algoritmo ARREGLO TRIDIMENSIONAL



Primer ejemplo punto 6.4 (página 187).

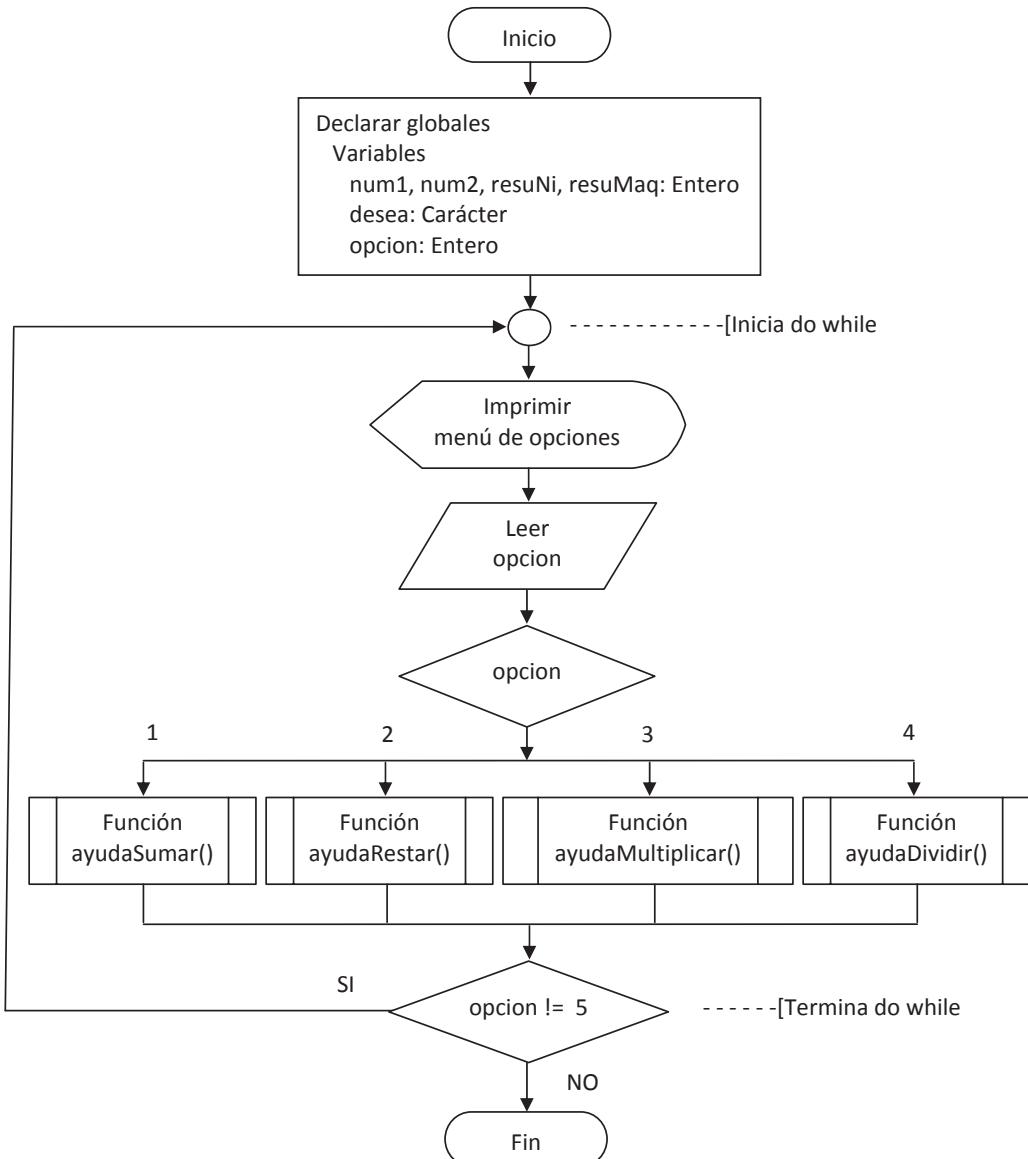
### Algoritmo ARREGLO TETRADIMENSIONAL



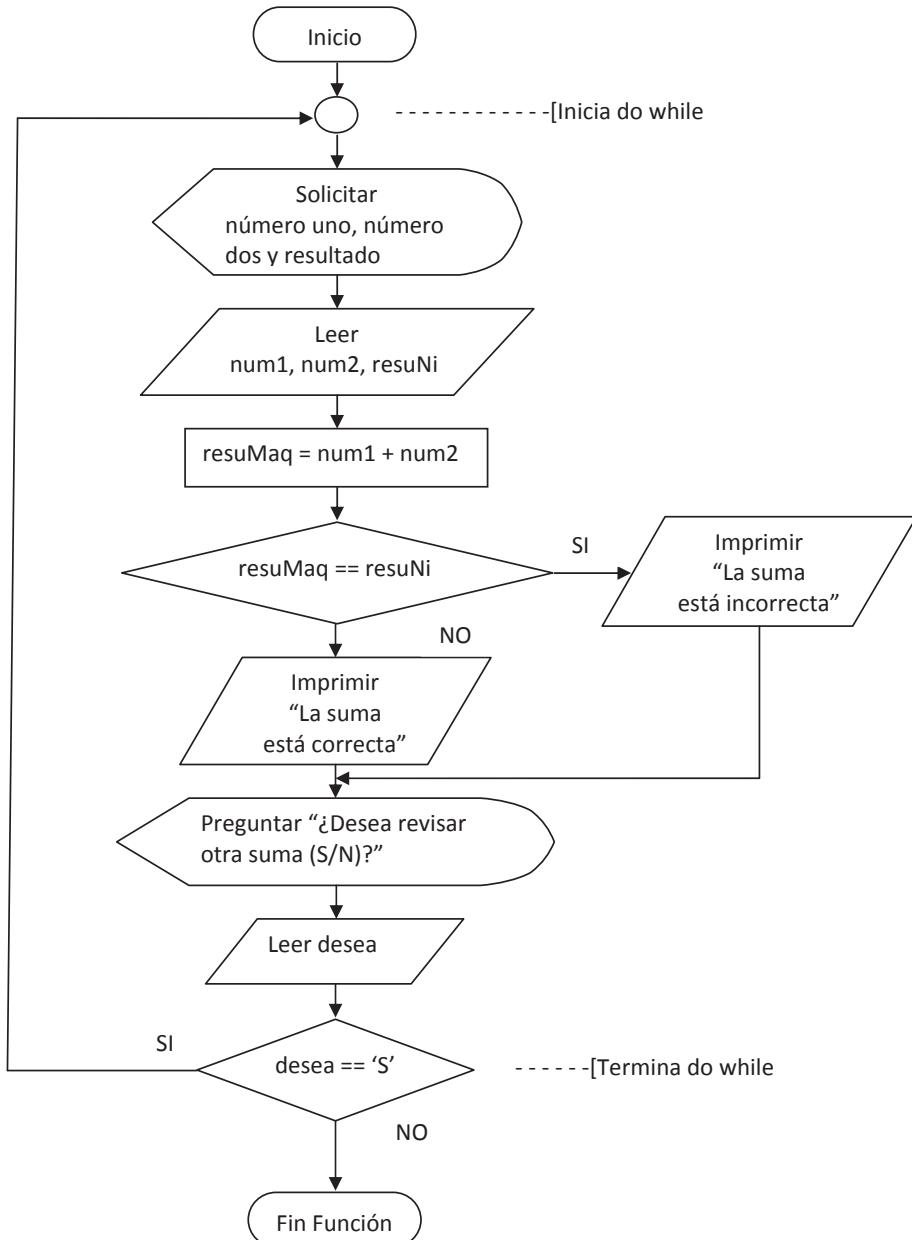


Primer ejemplo capítulo 7 (página 206).

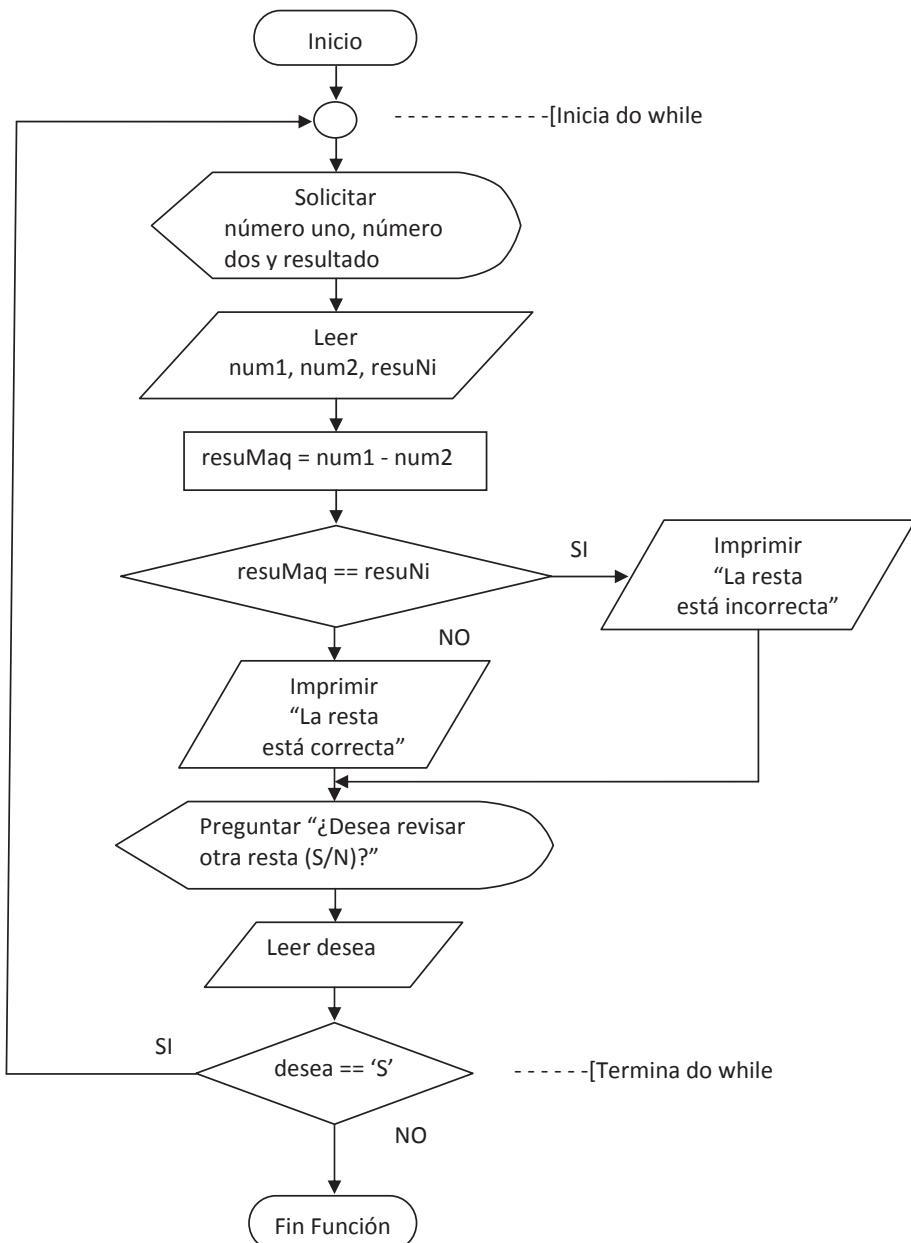
### Algoritmo AYUDA Función principal()

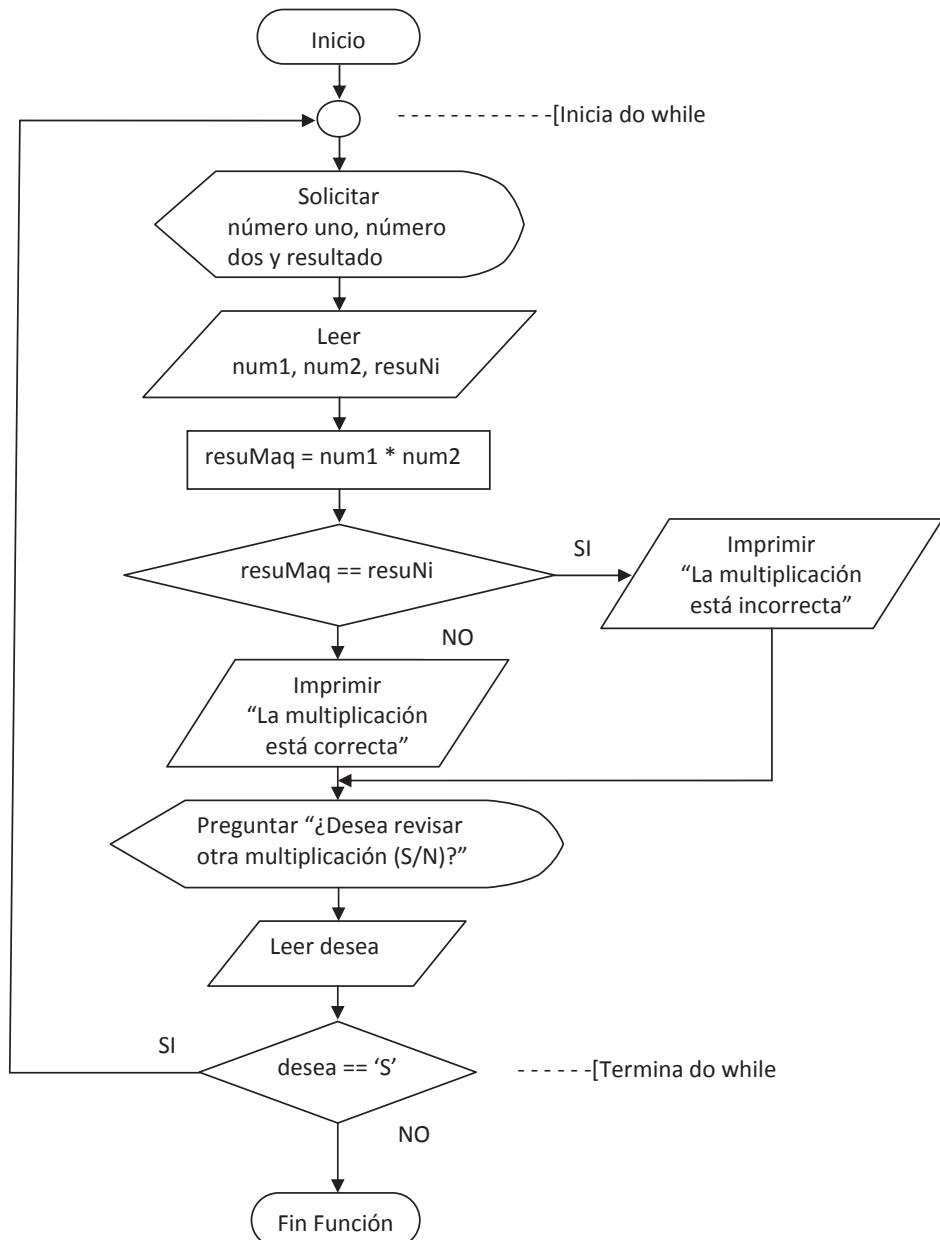


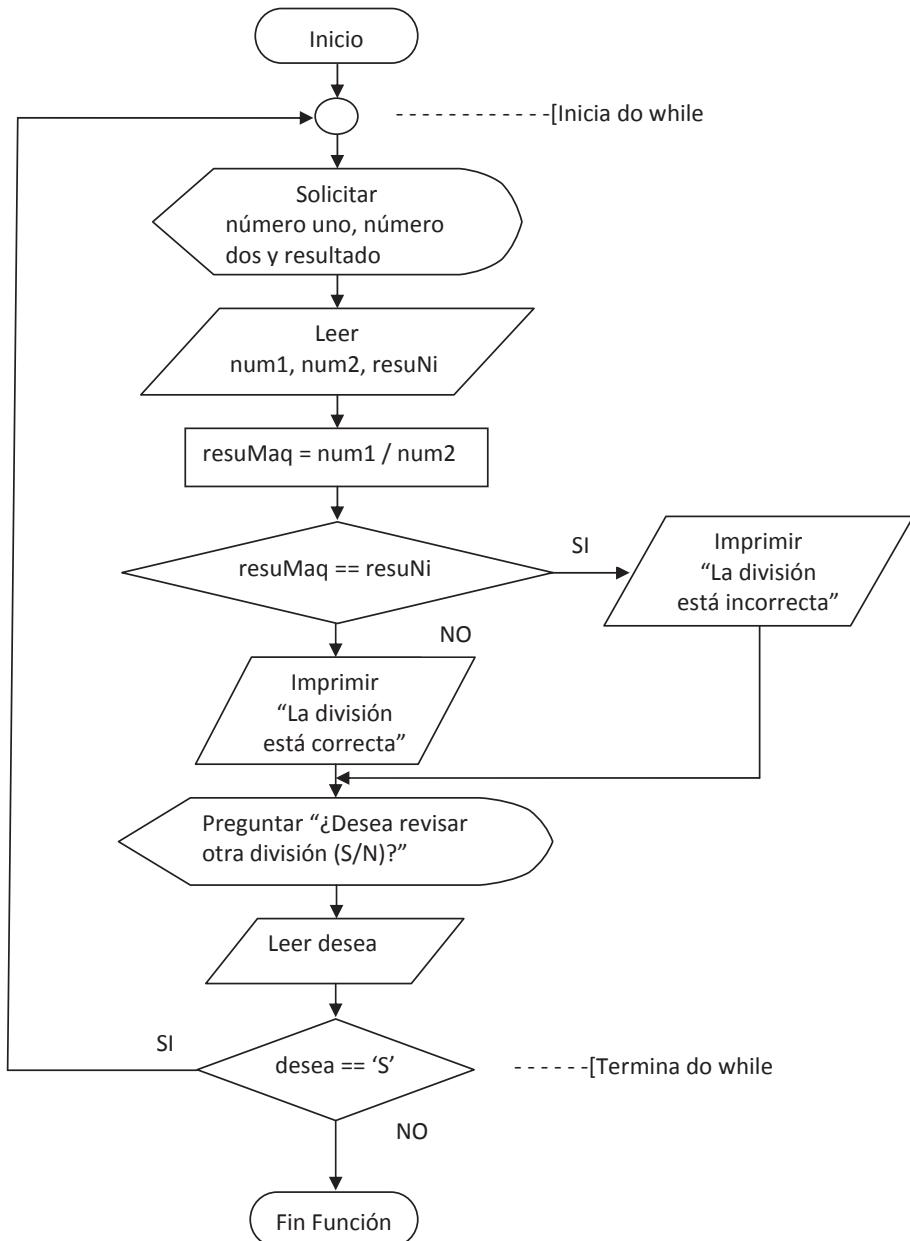
## Función ayudaSumar()



### Función ayudaRestar()



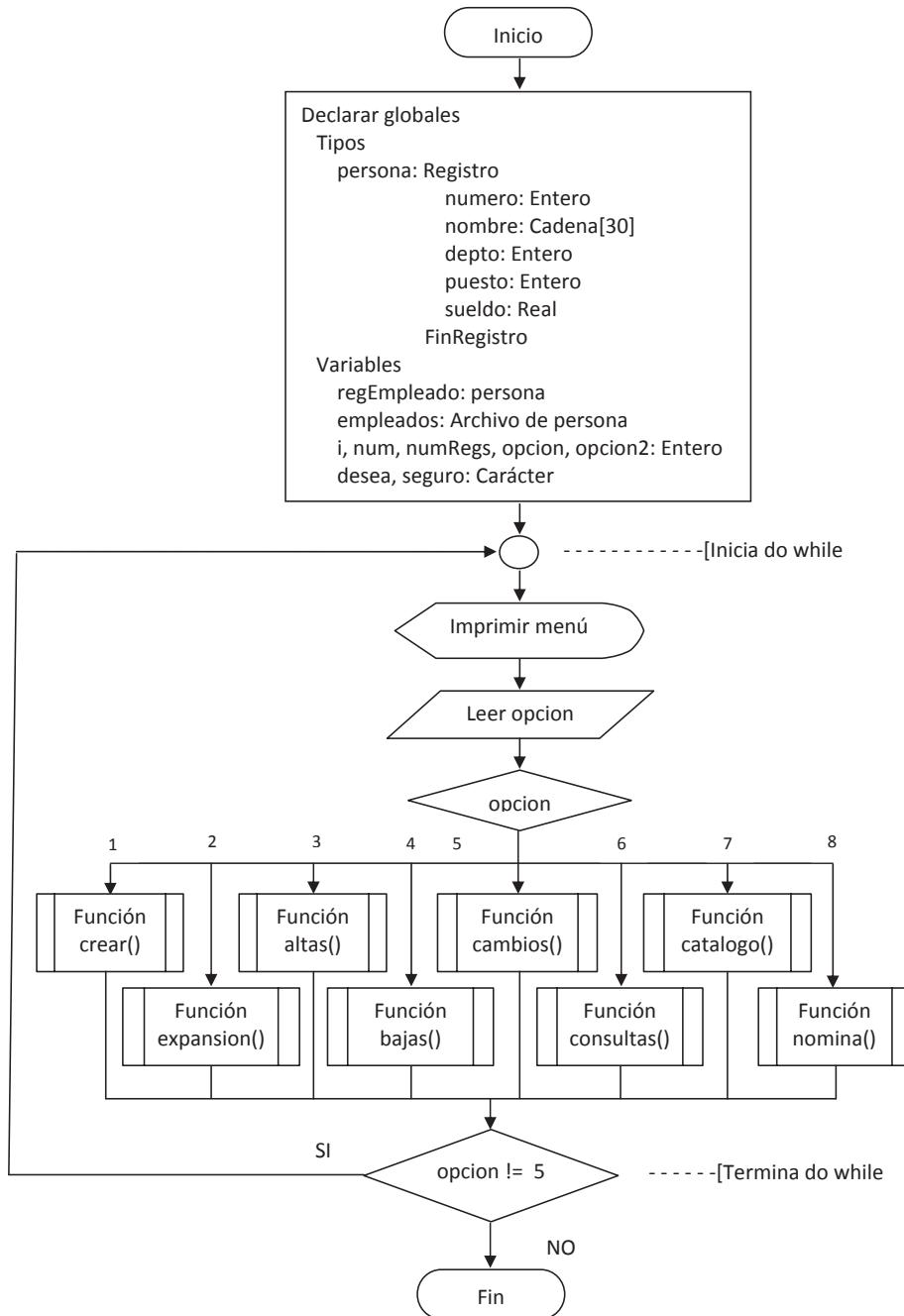
**Función ayudaMultiplicar()**

**Función ayudaDividir()**

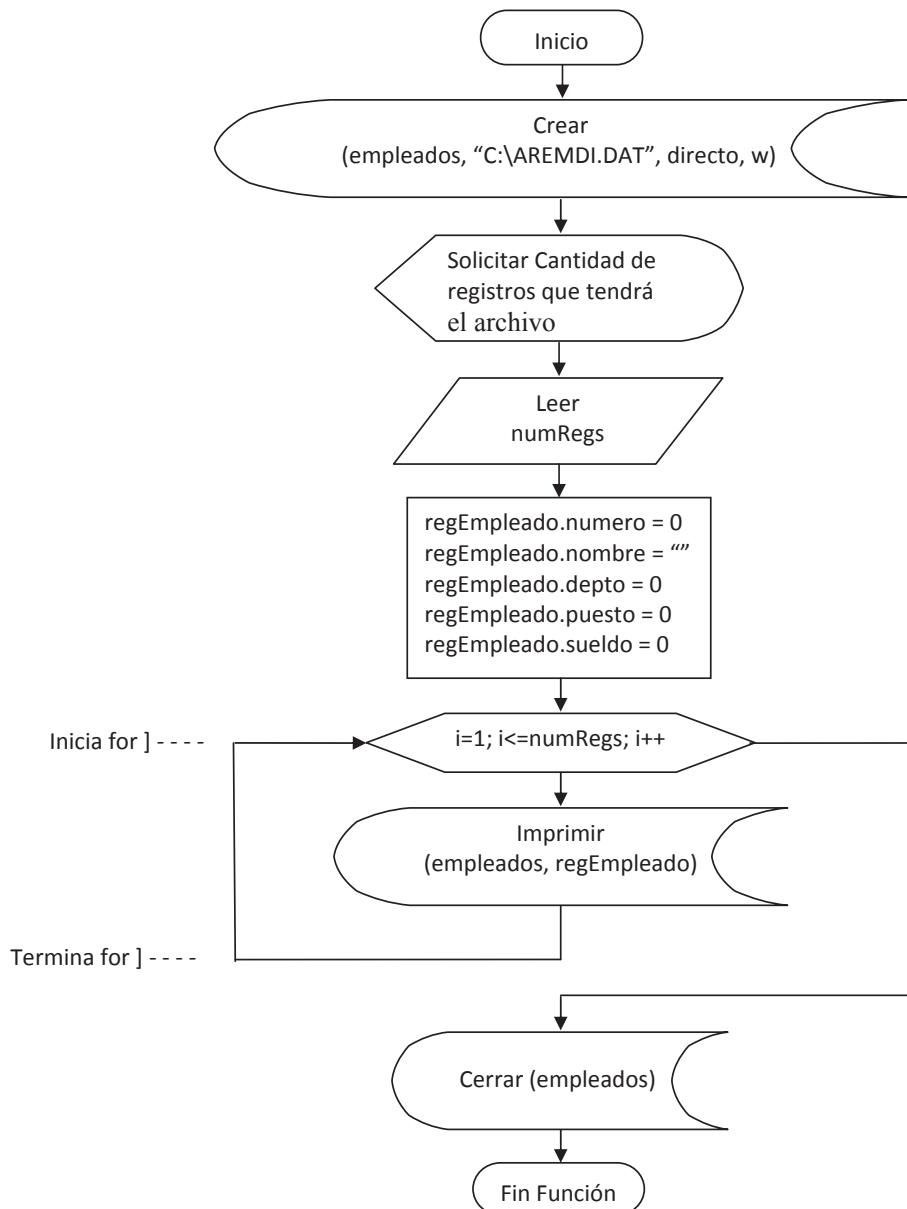
Ejemplo capítulo 8 (página 286).

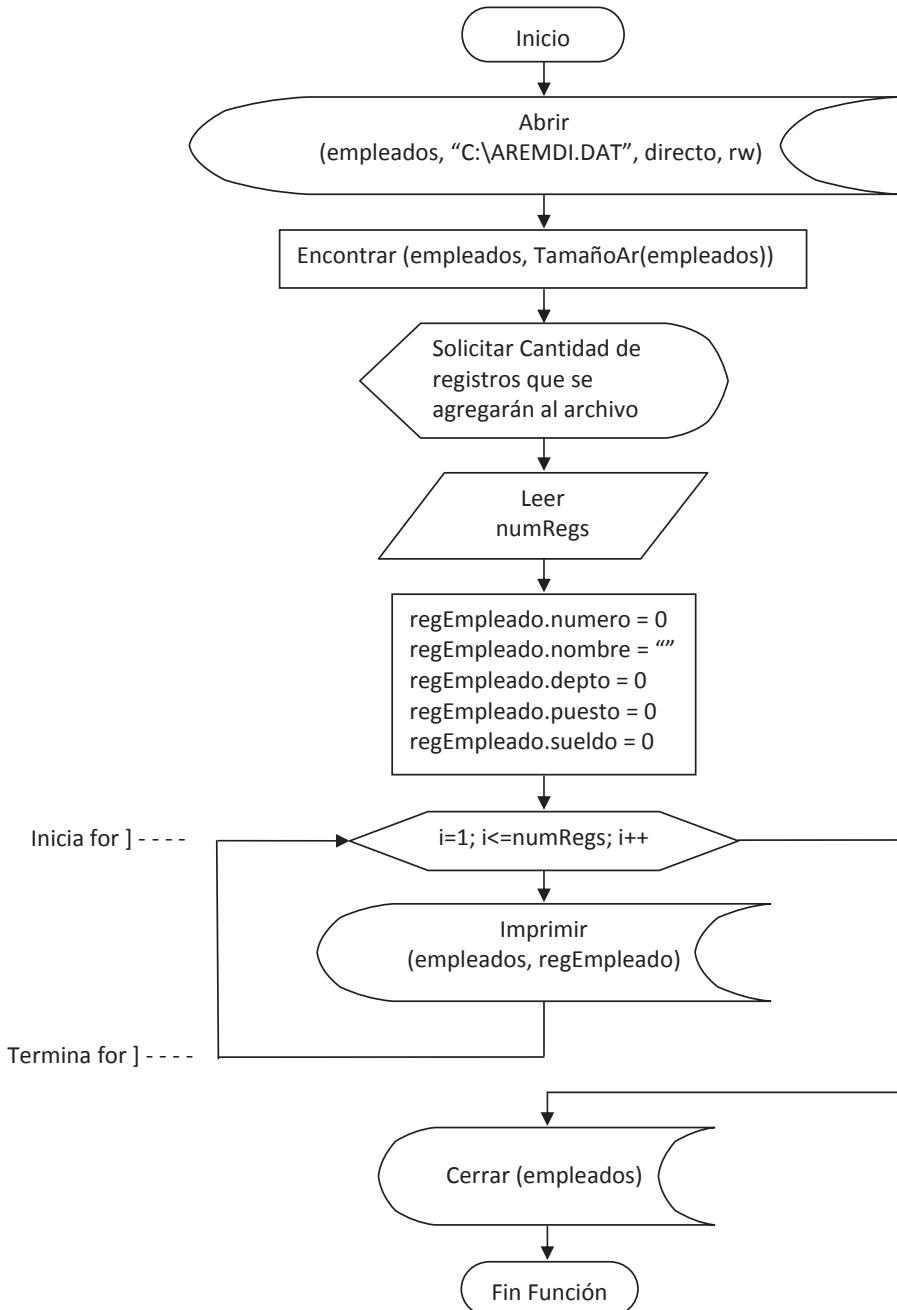
### Algoritmo SISTEMA DE NÓMINA DIRECTO

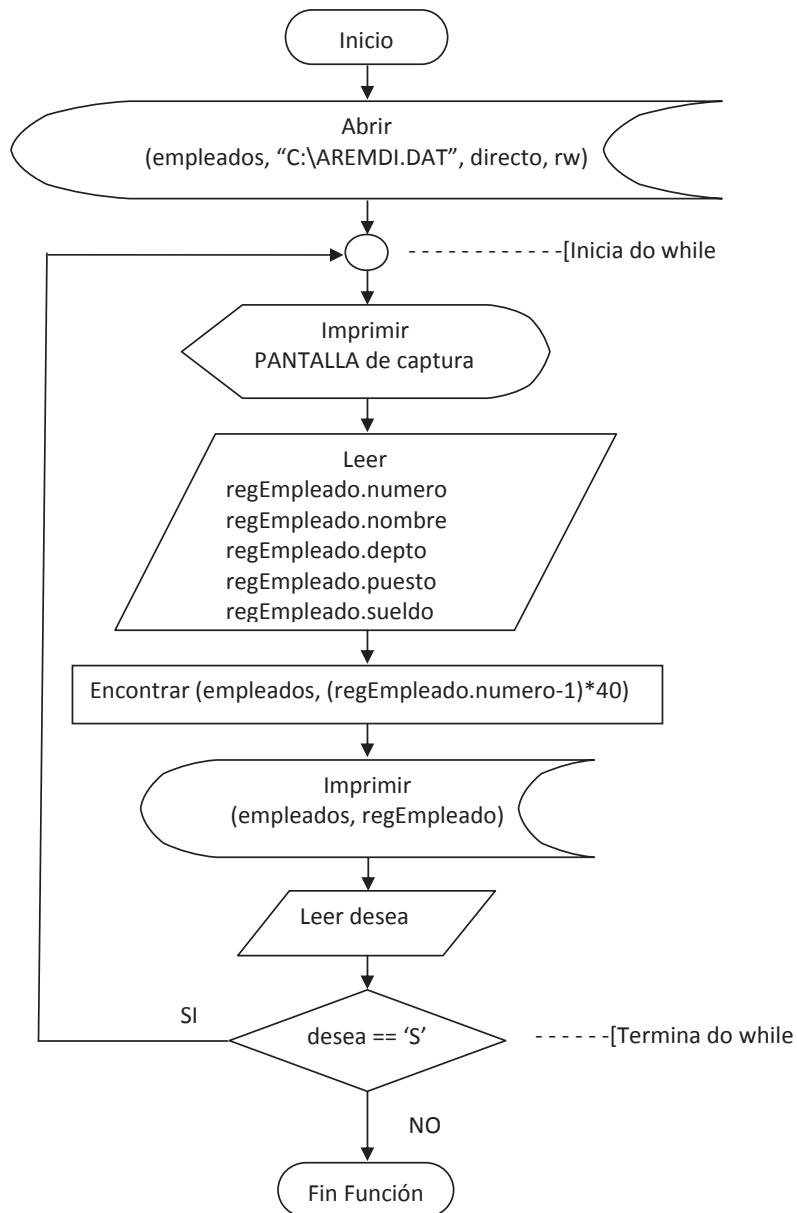
#### Función principal()



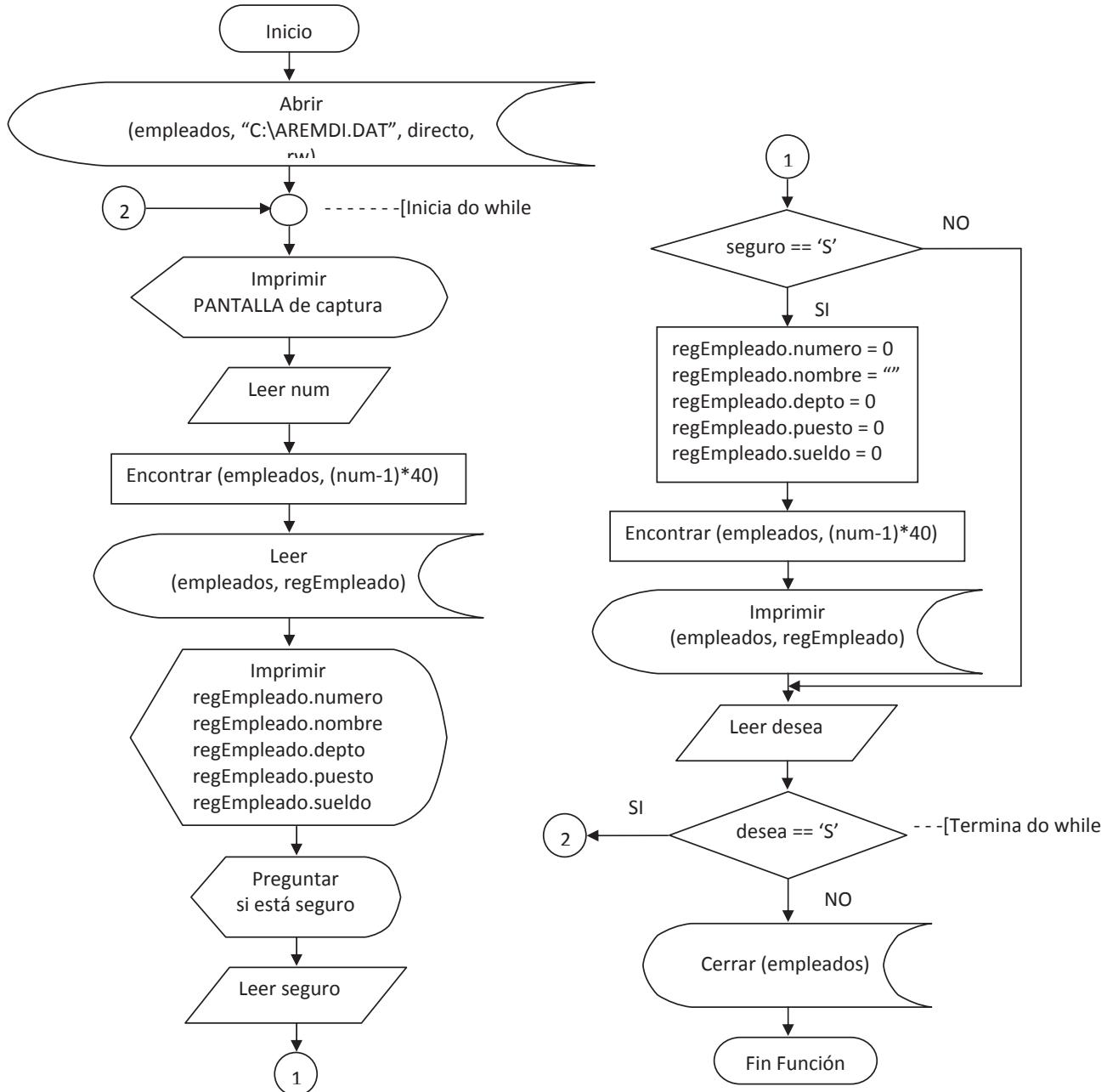
### Función crear()



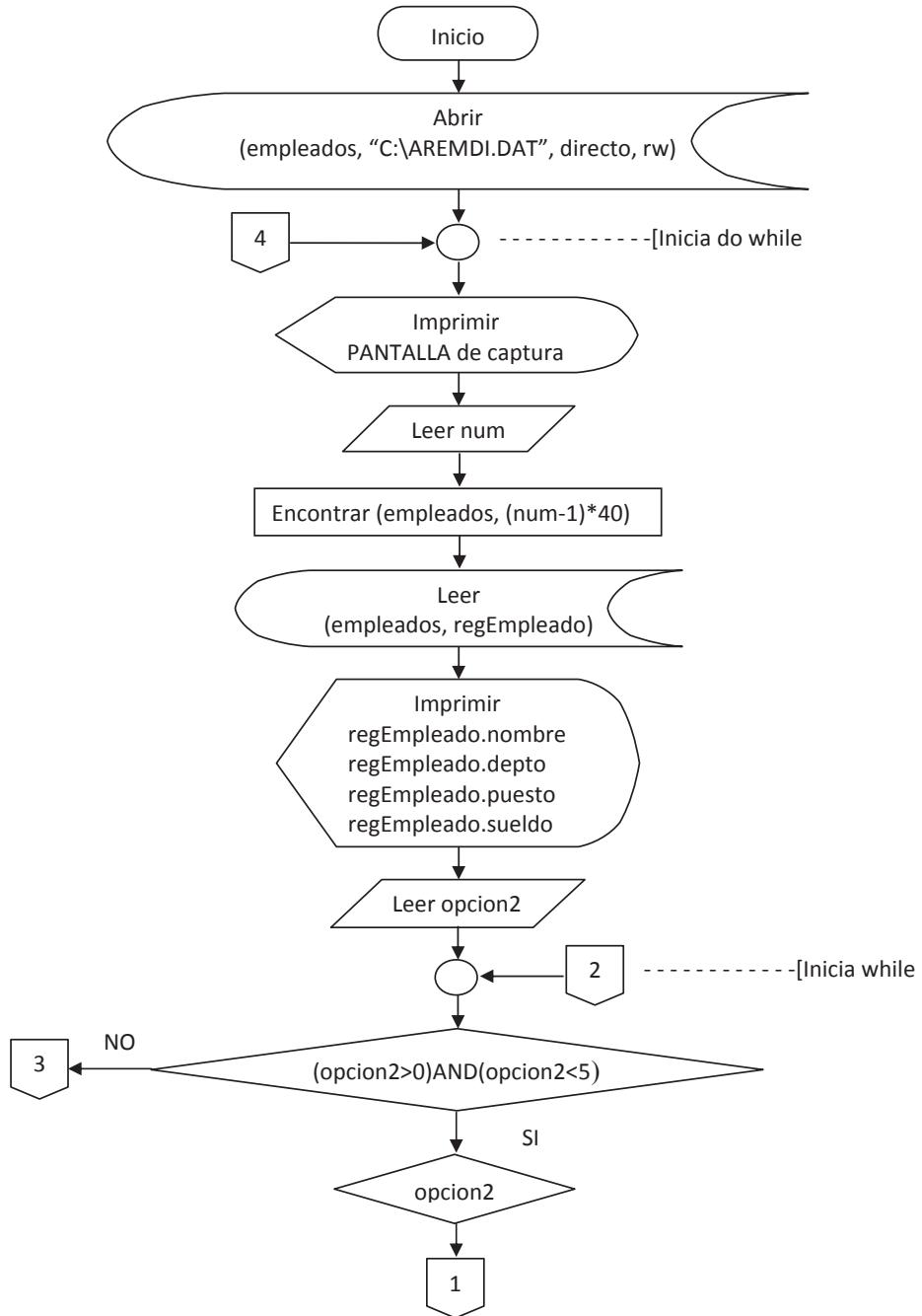
**Función expansion()**

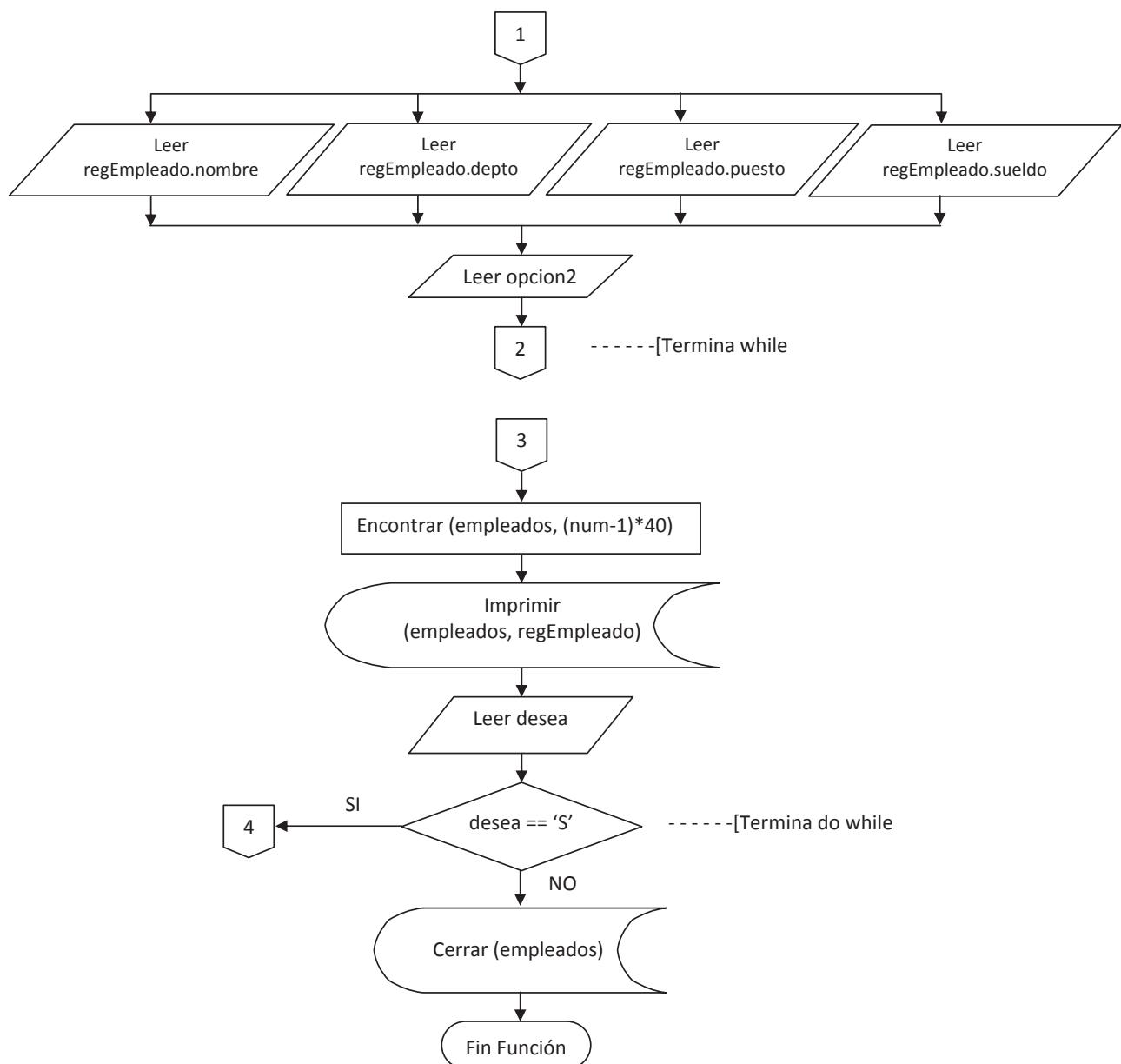
**Función altas()**

## Función bajas()

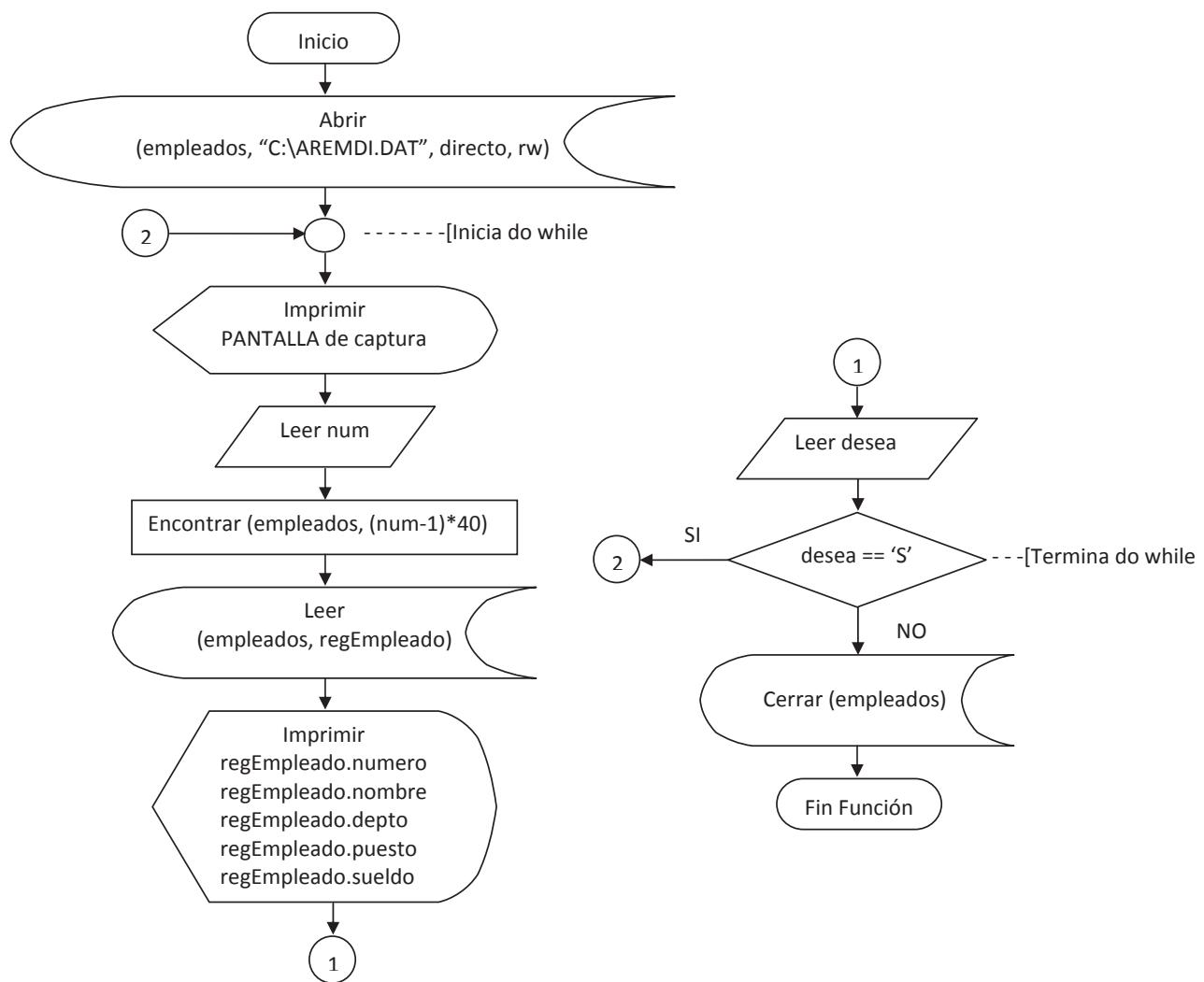


### Función cambios()

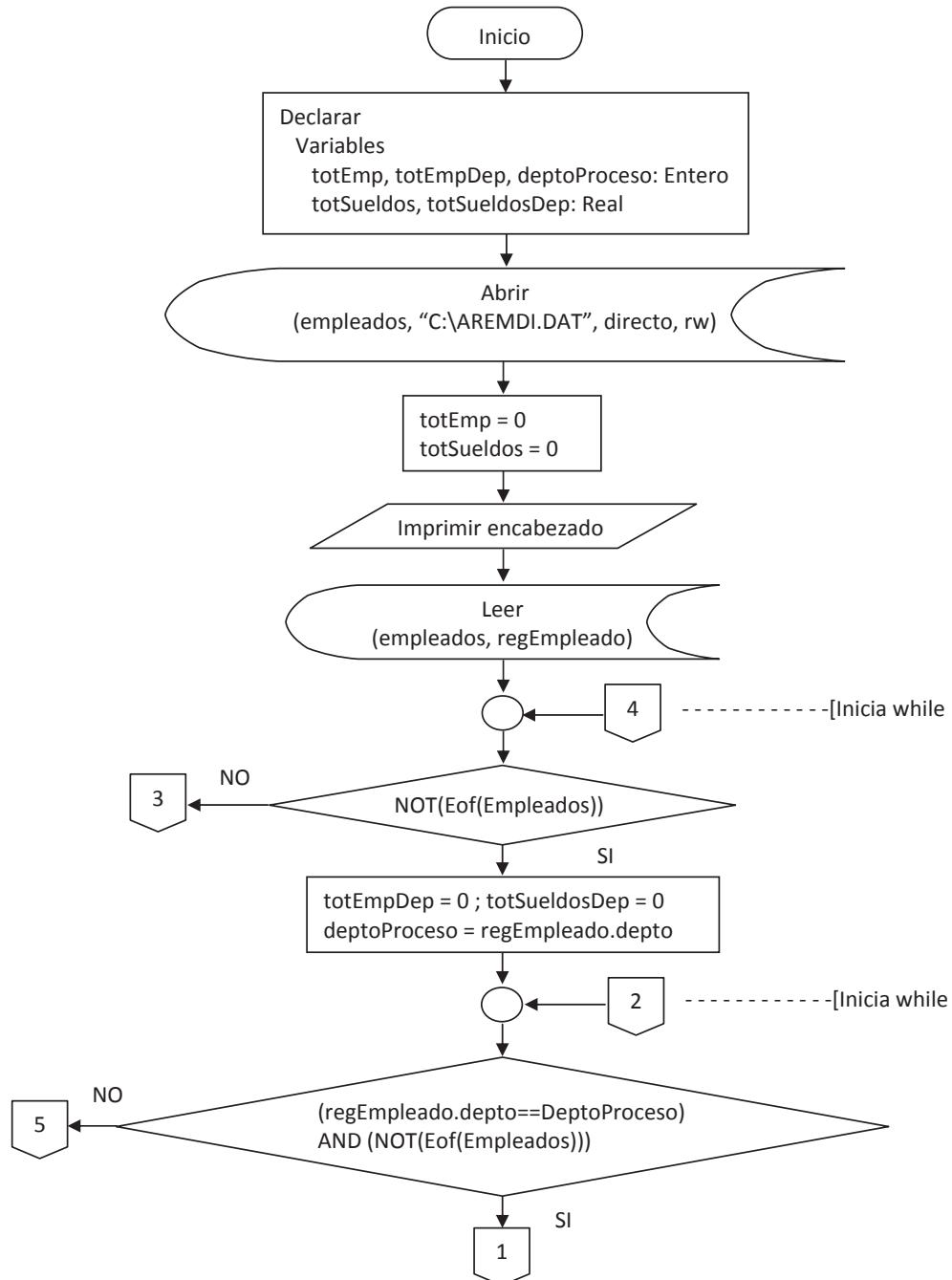


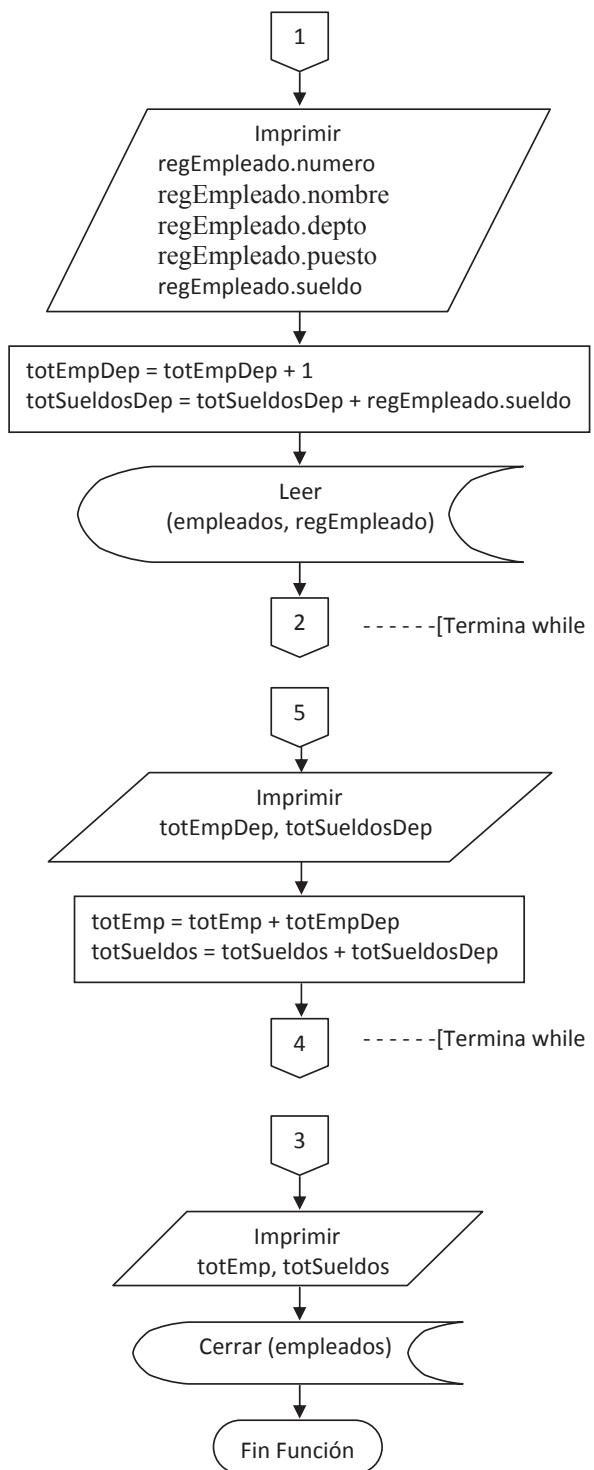


### Función consultas()

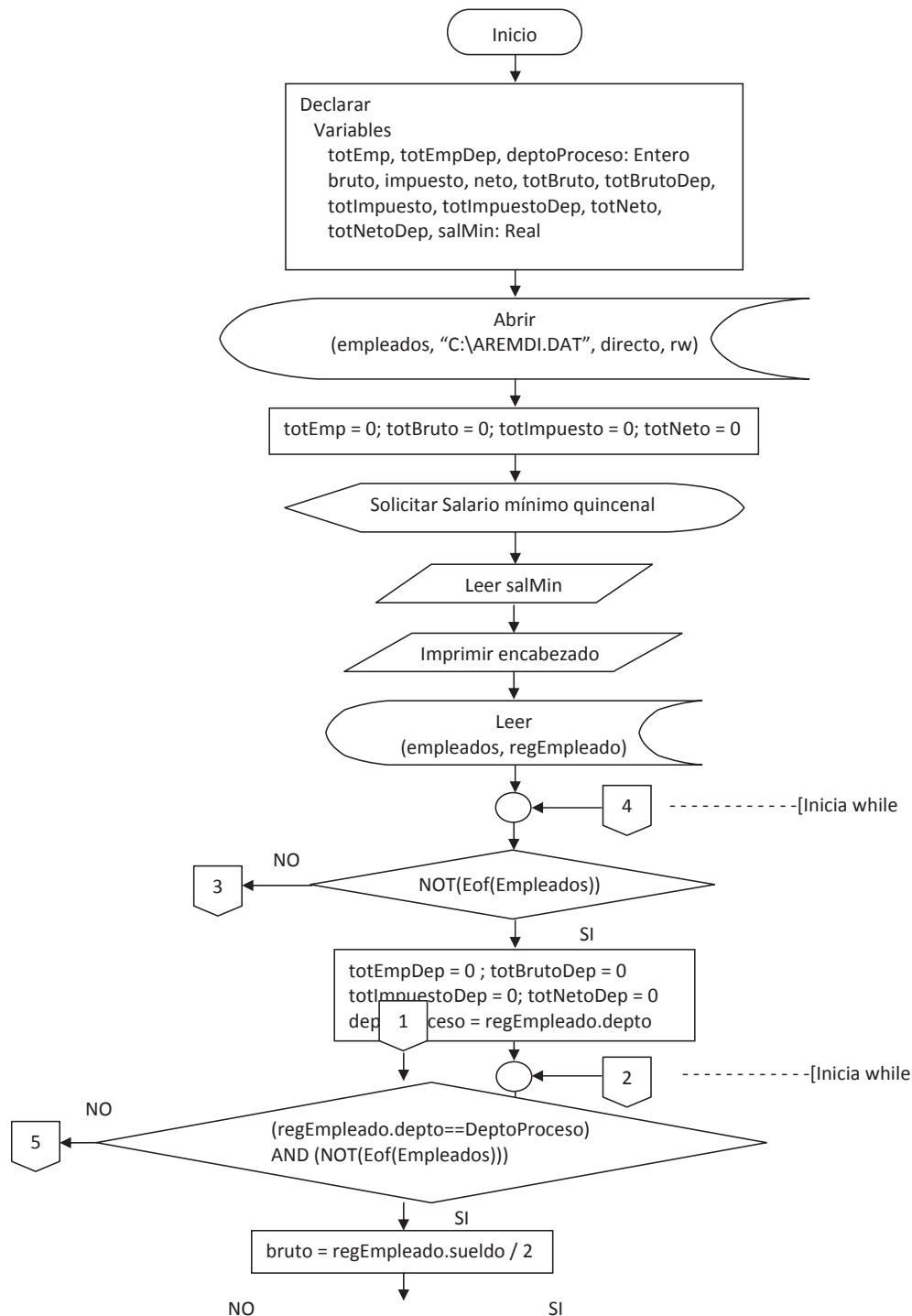


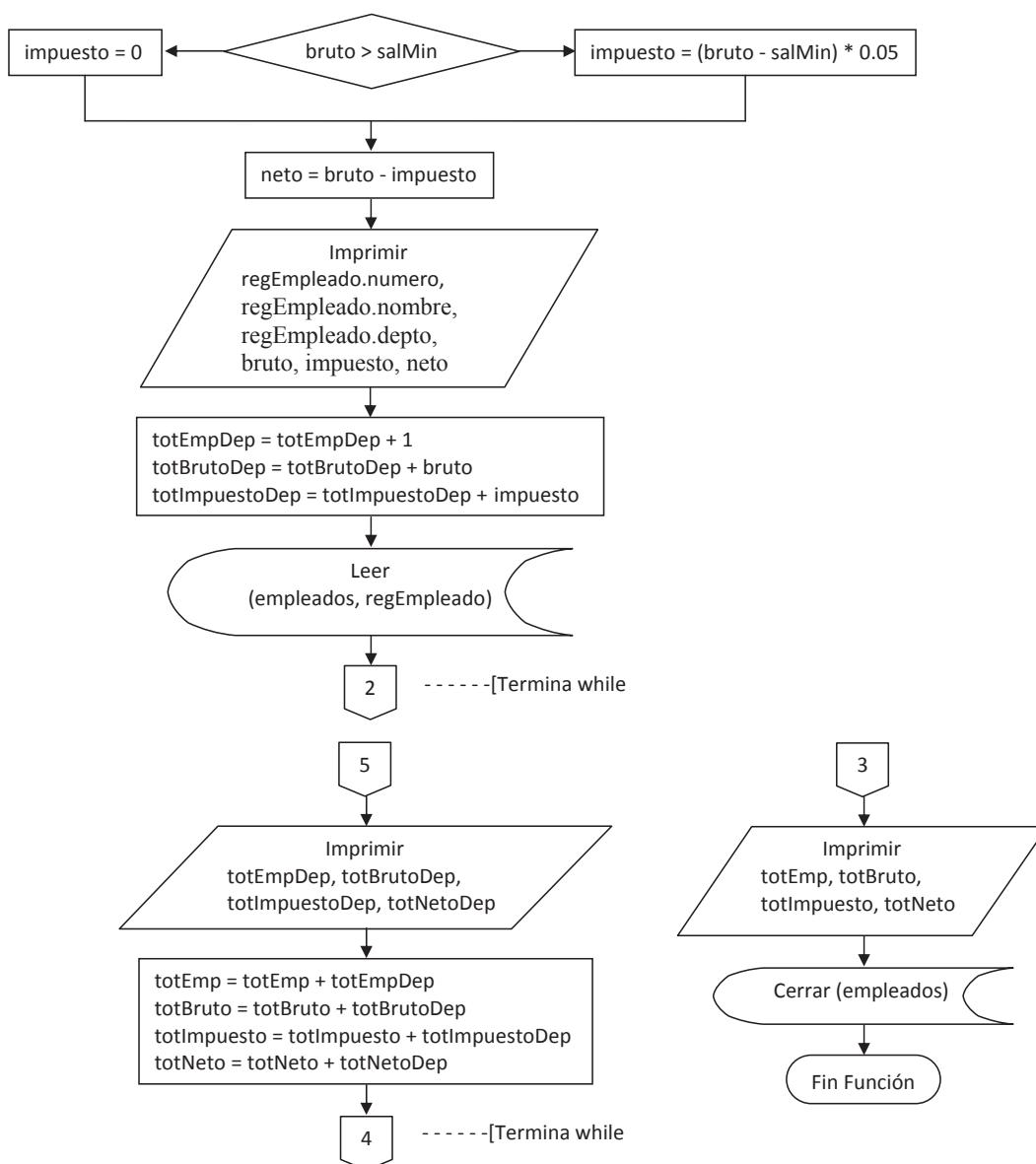
## Función catalogo()





## Función nomina()





# C

## Apéndice C: Diagramas Warnier

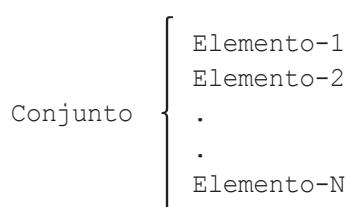
En este apéndice se presenta una técnica adicional para el diseño de programas, es decir, para elaborar algoritmos, se trata de los diagramas Warnier. Dicha técnica tiene el nombre de su autor (el francés Jean Dominique Warnier), y es él quien tiene todo el mérito de este invento. En éste libro, se han tomado esas ideas como base conjuntamente con los conceptos de nuestra metodología poniendo a su disposición otro método adicional de diseño de programas (algoritmos). En éste trabajo se hacen algunas adaptaciones con el propósito de aprovechar los conceptos de la metodología original de éste autor conjuntamente con los conceptos de Warnier, mostrando una técnica que permitirá al estudiante tener otra alternativa para el diseño de programas (algoritmos). El tratamiento que se le ha dado a ésta técnica, presupone que el lector ya ha estudiado la metodología original del libro, es decir, en seudocódigo.

## 1. El diagrama Warnier

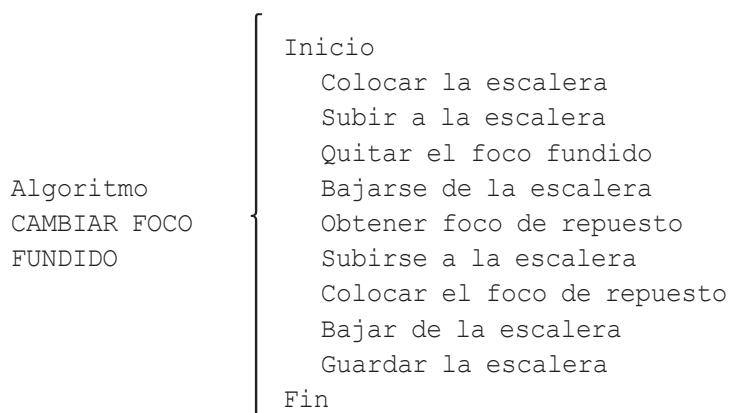
El diagrama Warnier utiliza el símbolo de la “llave”:



mediante la que se describen conjuntos; del lado izquierdo se coloca el nombre del conjunto y del lado derecho los elementos que lo componen, como se muestra a continuación:



Jean Dominique Warnier aplicó éstos conceptos a las secuencias lógicas de la programación, formando una técnica de diseño de programas (algoritmos). El algoritmo se considera como un conjunto, y, sus elementos, son las acciones del mismo. Un algoritmo como el de cambiar un foco fundido (que hicimos en el capítulo 1), para el robot, queda de la siguiente forma:



## 2. Elementos para solucionar problemas en diagramas Warnier

Los tipos de datos, la forma de dar nombres de identificadores y las operaciones primitivas elementales que utilizaremos aquí, para explicar el uso de los diagramas Warnier, serán los mismos que explicamos en el capítulo 2 de nuestra metodología original:

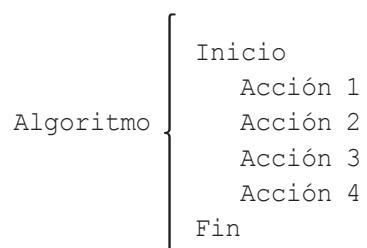
```

Declarar
    Constantes
        ---
        ---
    Tipos
        ---
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
    Solicitar    Nombre del empleado, Número de horas
                trabajadas y Cuota por hora
    Leer nombreEmp, horasTrab, cuotaHora
    Calcular sueldo = horasTrab * cuotaHora
    Imprimir nombreEmp, sueldo

```

## 3. La secuenciación

La secuenciación se representa de arriba hacia abajo y de izquierda a derecha con el formato:



Ejemplo:

Nuestro primer algoritmo del capítulo 3 (página 43).

```

Algoritmo      Inicio
CALCULA        Declarar
SUELDO         Variables
DE UN          nombreEmp: Cadena
EMPLEADO       horasTrab: Entero
                cuotaHora, sueldo: Real
                Solicitar Nombre del empleado,
                Número de horas trabajadas
                y Cuota por hora
                Leer nombreEmp, horasTrab, cuotaHora
                Calcular sueldo = horasTrab * cuotaHora
                Imprimir nombreEmp, sueldo
Fin
  
```

## 4. La selección

La selección tiene tres formas: la doble (if-then-else), la simple (if-then) y la múltiple (switch).

### 4.1 La selección doble (if-then-else)

La selección doble se representa mediante el siguiente formato:

```

Condición      SI   { Acción(es)
                  ⊕
                  NO  { Acción(es)
  
```

Donde:

Condición Es una expresión lógica mediante la que plantea la condición de ejecución selectiva. Se utilizan los mismos conceptos explicados en el capítulo 4.

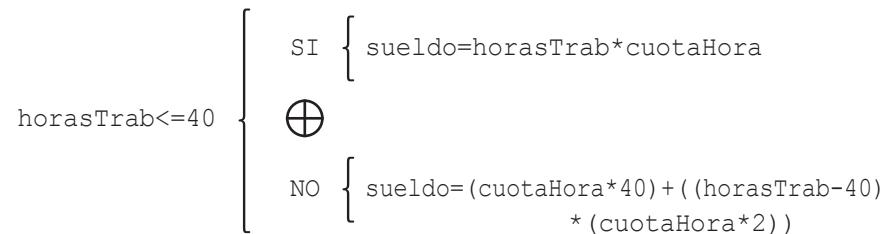
SI Es el camino de acción para cuando se cumple la condición.

NO Es el camino de acción para cuando no se cumple la condición.

⊕ Es el símbolo que representa la exclusión, es decir, uno ú otro.

Acción(es) Son las acciones que se ejecutarán.

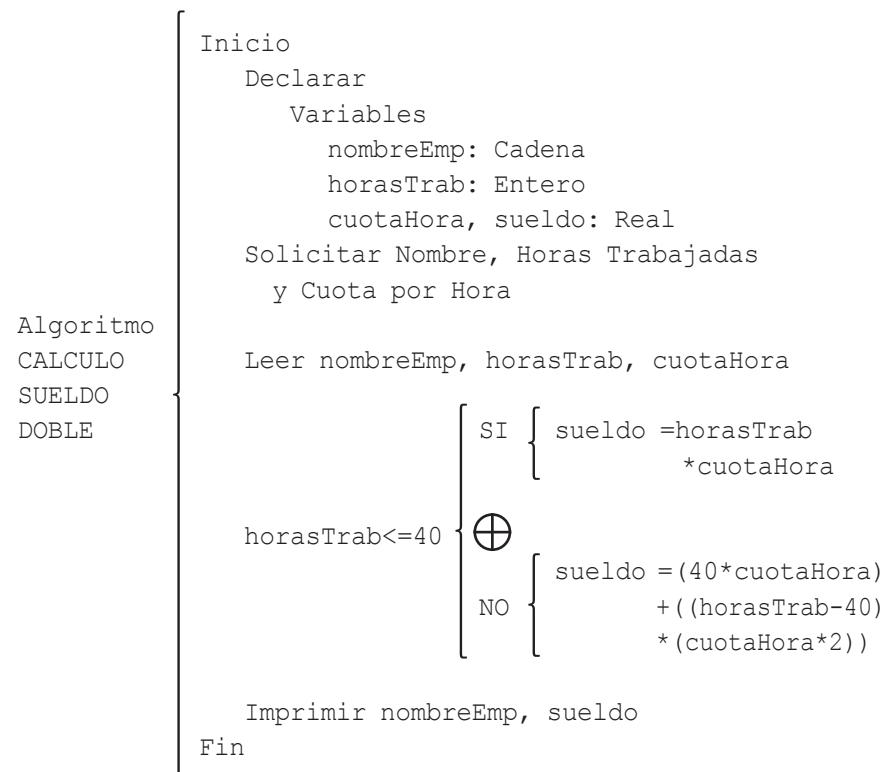
Ejemplo: El ejemplo del capítulo 4 (página 60).



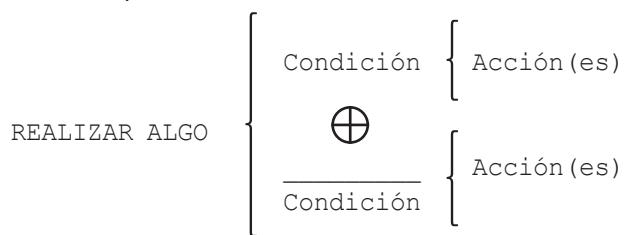
*Explicación:*

Se evalúa la expresión lógica (horasTrab<=40); si se cumple se va por el camino SI, en caso de no cumplirse se va por el camino NO, es decir, calcula el sueldo de una o de la otra forma.

A continuación se presenta el algoritmo completo:



Otra forma de plantear la selección es:



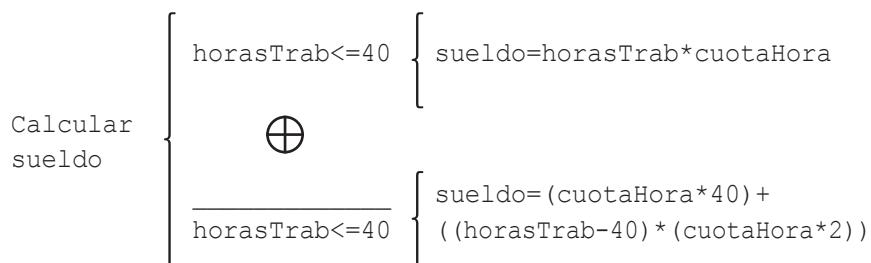
Donde:

REALIZAR ALGO Es algún proceso a ejecutar, por ejemplo hacer un cálculo.

Condición Es la condición.

Condición Es la negación de la condición.

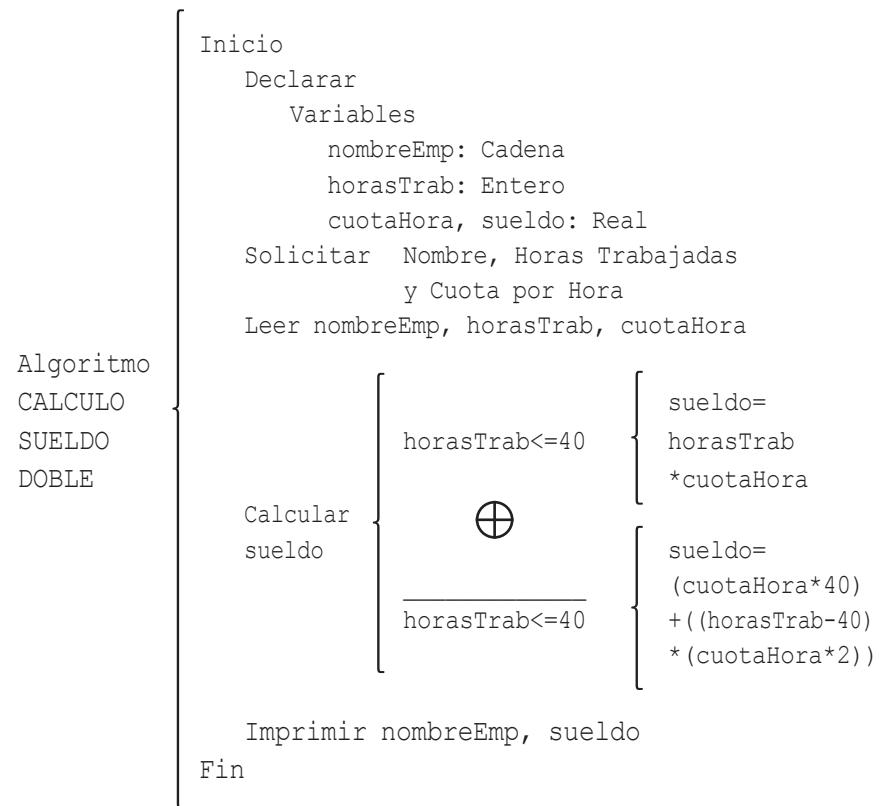
Ejemplo:



*Explicación:*

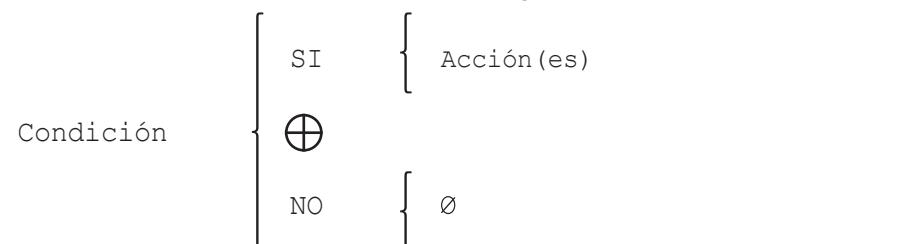
Se tiene la acción Calcular sueldo, la cual tiene dos alternativas de acción; se plantea la condición para una alternativa ( $\text{horasTrab}<=40$ ) y lo que se hace en tal caso. Mientras que por otro lado se coloca la negación de la condición, es decir, el complemento, por ello se coloca la misma condición con la raya arriba (esto significa la negación de la condición), igualmente se coloca lo que debe hacer en éste caso.

A continuación se presenta el algoritmo completo, con esta nueva forma:



#### 4.2 La selección simple (if-then)

La selección simple se representa mediante el siguiente formato:



Donde:

- |           |                                                                                      |
|-----------|--------------------------------------------------------------------------------------|
| Condición | Es una expresión lógica mediante la que plantea la condición de ejecución selectiva. |
| SI        | Es el camino de acción para cuando se cumple la condición.                           |
| NO        | Es el camino de acción para cuando no se cumple la condición.                        |
|           | Es el símbolo que representa la exclusión, es decir, uno ú otro.                     |

- Acción(es) Son las acciones que se ejecutarán.  
 O Es el conjunto vacío, es decir, no hay acciones.

Otra forma como podría representarse la selección simple (if-then):

if  
 Condición    {  
     Acción(es)

Es decir, se plantea la condición, y sólo se coloca la opción SI; si se cumple, con las acciones a ejecutar en tal caso. Si se cumple la condición, entra a ejecutar las acciones; en caso de no cumplirse, no hace nada; y se va a la siguiente acción.

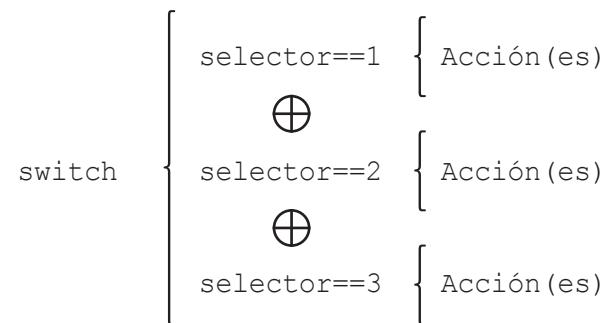
Ejemplo:

Algoritmo del capítulo 4 (página 75).

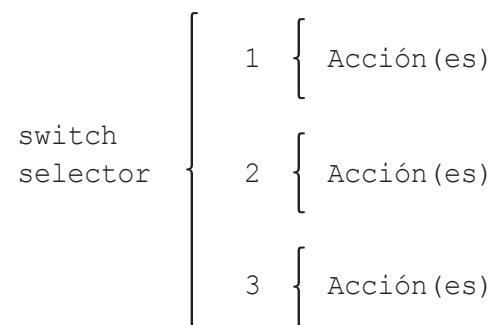
Algoritmo  
 MAYOR 5  
 NUMEROS    {  
 Inicio  
 Declarar  
 Variables  
 a, b, c, d, e, mayor: Entero  
 Solicitar número 1, número 2, número 3,  
 número 4, número 5  
 Leer a, b, c, d, e  
 mayor = a  
 if  
 b > mayor { mayor = b  
 if  
 c > mayor { mayor = c  
 if  
 d > mayor { mayor = d  
 if  
 e > mayor { mayor = e  
 Imprimir mayor  
 Fin

### 4.3 La selección múltiple (switch)

La selección múltiple se representa mediante el formato:



O bien, con el formato:



Donde:

selector

Es una variable de tipo entero que permitirá escoger la opción de acuerdo a su valor.

Ejemplo:

Algoritmo del capítulo 4 (página 94).

Algoritmo  
CLIENTE  
HOJAS  
HIELO  
SECO

```
Inicio
    Declarar
        Variables
            nombreClie: Cadena
            tipoClie, cantidad: Entero
            precioUni, subTotal, descuento,
            netoPagar: Real
        Solicitar Nombre, Tipo cliente,
                    Cantidad, Precio unitario
        Leer nombreClie, tipoClie,
                cantidad, precioUni
        subTotal = cantidad * precioUni

        switch tipoClie {
            1 { descuento=subTotal*0.05
            2 { descuento=subTotal*0.08
            3 { descuento=subTotal*0.12
            4 { descuento=subTotal*0.15

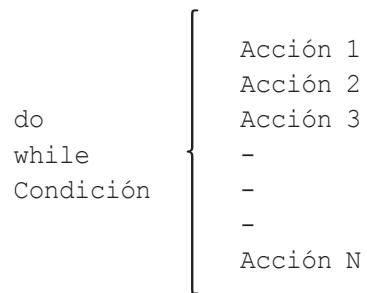
        netoPagar = subTotal - descuento
        Imprimir nombreClie, subTotal
                            descuento, netoPagar
    Fin
```

## 5. La repetición

La repetición tiene tres formas: do...while, for y while.

### 5.1 La repetición do...while.

La repetición do...while se representa mediante el siguiente formato:



Donde:

do ...while Condición.

Indica el conjunto, el cual es un ciclo repetitivo; las acciones que están a la derecha de la "llave", son las que se ejecutarán dentro del ciclo.

Ejercicio segundo del capítulo 5 (página 106).

```

    Inicio
    Declarar
        Variables
            nombreEmp: Cadena
            horasTrab, totEmpleados: Entero
            cuotaHora, sueldo, totSueldos: Real
            desea: Carácter
        Imprimir encabezado
        totEmpleados=0; totSueldos=0

Algoritmo
CALCULA
SUELDOS DE
cuotaHora
EMPLEADOS
{
    do
    while
    desea=='S'
    {
        Solicitar Nombre,
                    Horas trabajadas
                    y Cuota por Hora
        Leer nombreEmp, horasTrab,
        sueldo = horasTrab * cuotaHora
        Imprimir nombreEmp, sueldo
        totEmpleados = totEmpleados + 1
        totSueldos = totSueldos + sueldo
        Preguntar ¿desea procesar otro
                    empleado (S/N)?
        Leer desea
    }

    Imprimir totEmpleados, totSueldos
}
Fin

```

## 5.2 La repetición for.

La repetición for se representa mediante el siguiente formato:

```

for
contador=valorInicial; condición; incremento
{
    Acción 1
    Acción 2
    Acción 3
    -
    -
    Acción N
}

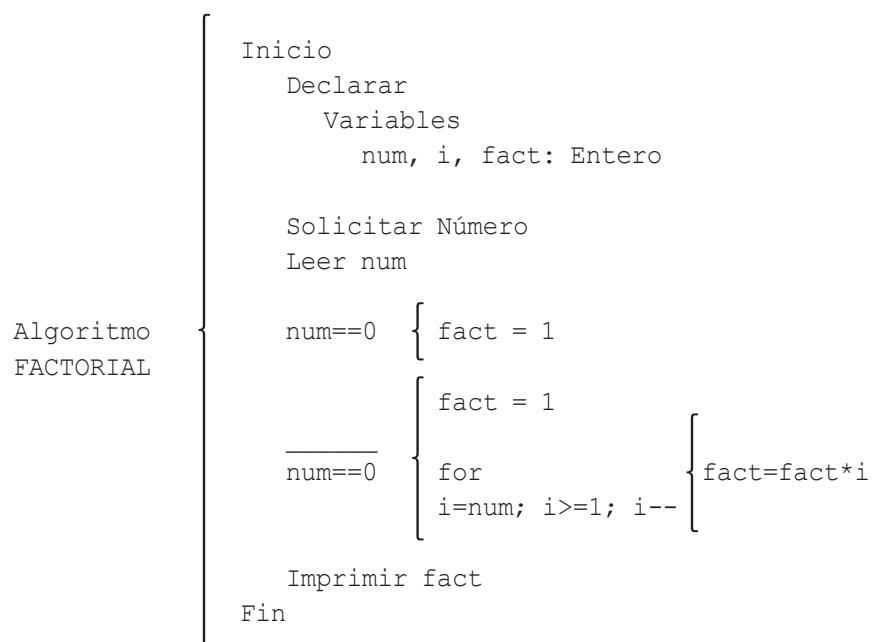
```

Donde:

Se plantea el ciclo for desde que el contador (contador) toma un valor inicial (valorIncial), con una condición que evalúa si no ha llegado a un valor final y con un incremento (incremento); si la condición se cumple, entra al ciclo; si no se va a la siguiente acción después del ciclo.

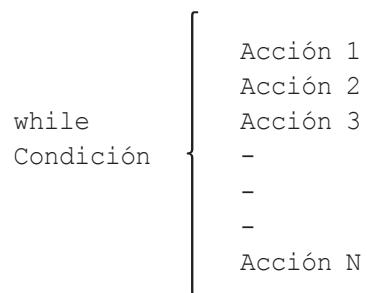
Ejemplo:

Algoritmo del capítulo 5 (página 133).



### 5.3. La repetición while

La repetición while se representa mediante el siguiente formato:



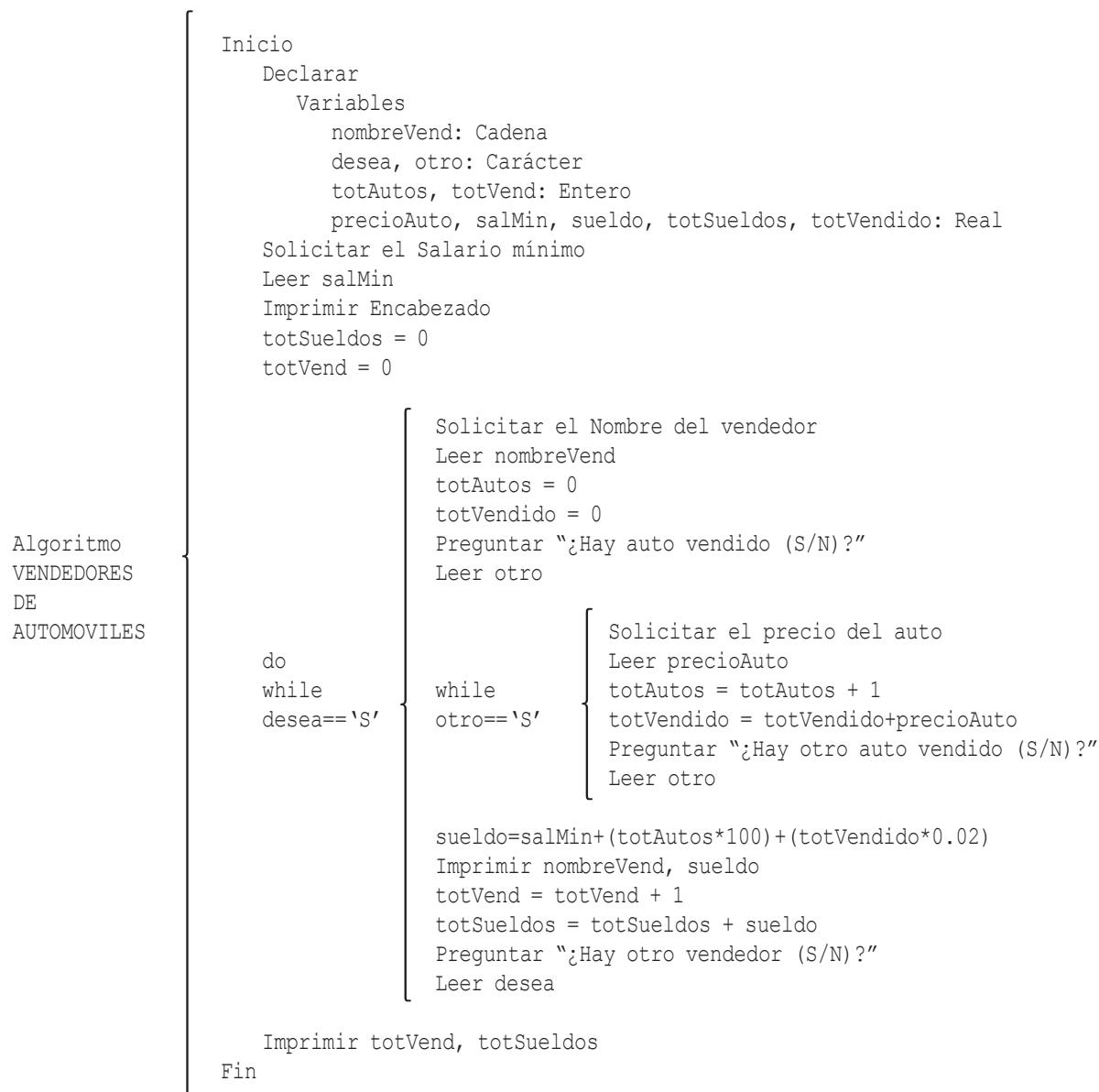
Donde:

while Condición.

Se establece el conjunto repetitivo; las acciones de la derecha es lo que se ejecutará dentro del ciclo, mientras se cumple la condición.

Ejemplo:

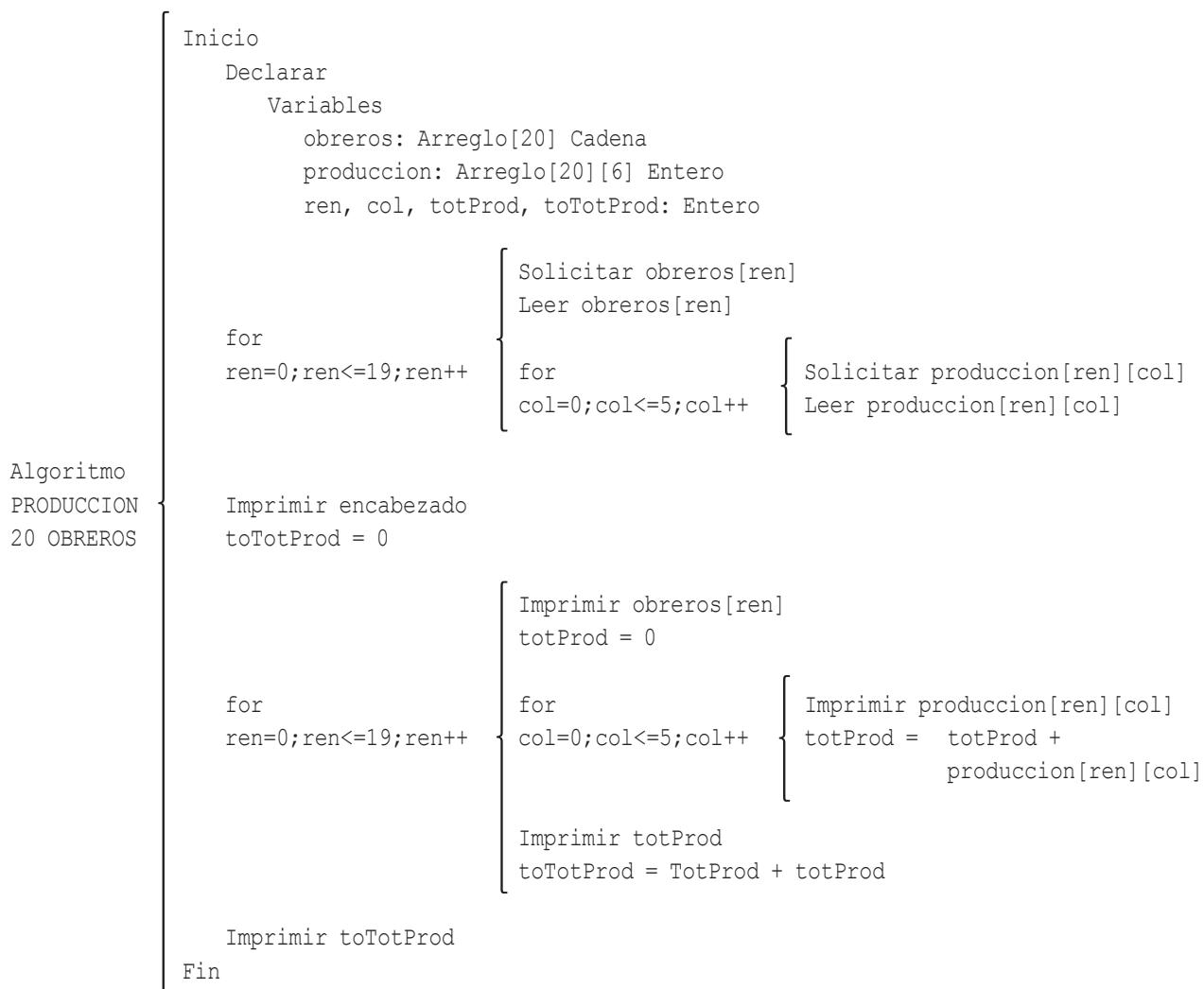
Algoritmo del capítulo 5 (página 150).



## 6. Arreglos con diagramas Warnier

Ejemplo:

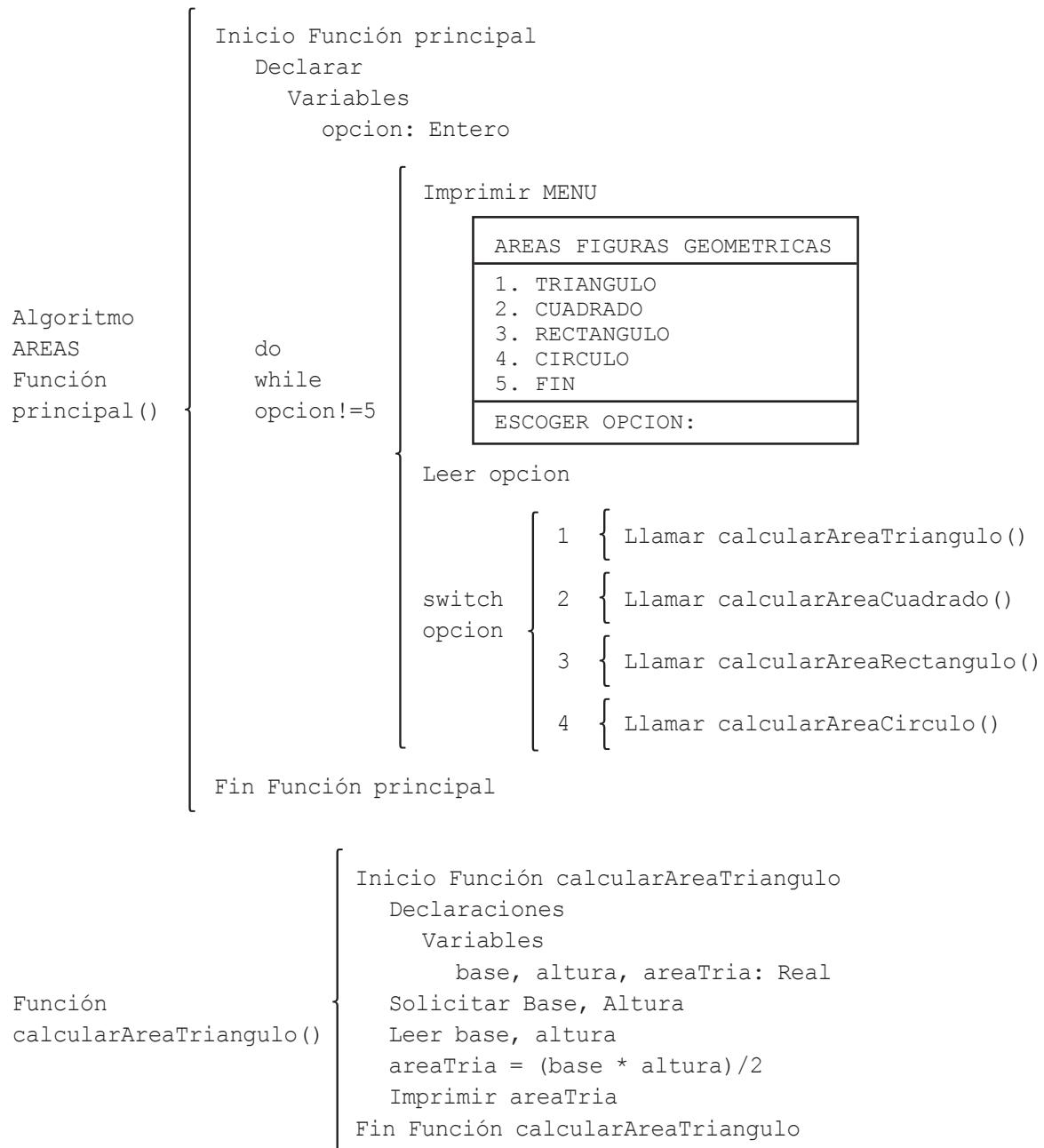
Algoritmo del capítulo 6 (página 177).



## 7. Diseño descendente (Top Down Design) con diagramas Warnier

## Ejemplo:

Algoritmo del capítulo 7 (página 213).



```
Inicio Función calcularAreaCuadrado
    Declarar
        Variables
            lado, areaCuad: Real
    Solicitar Lado
    Leer lado
    areaCuad = Potencia(lado,2)
    Imprimir areaCuad
Fin Función calcularAreaCuadrado

Función calcularAreaRectangulo()
{
    Inicio Función calcularAreaRectangulo
    Declarar
        Variables
            areaRec, base, altura: Real
    Solicitar Base, Altura
    Leer base, altura
    areaRec = base * altura
    Imprimir areaRec
Fin Función calcularAreaRectangulo

Función calcularAreaCirculo()
{
    Inicio Función calcularAreaCirculo
    Declarar
        Constantes
            PI = 3.14159265
        Variables
            areaCirc, radio: Real
    Solicitar Radio
    Leer radio
    areaCirc = PI * Potencia(radio,2)
    Imprimir areaCirc
Fin Función calcularAreaCirculo
```

Nota: Se recomienda elaborar algunos de los algoritmos de los ejercicios resueltos o propuestos de los capítulos 3,4,5,6 y 7.

# D

## Apéndice D: Diagramas Chapin (Nassi-Schneiderman)

### Contenido

- 1 El diagrama Chapin (Nassi-Schneiderman)
- 2 Elementos para solucionar problemas en diagramas Chapin (Nassi-Schneiderman)
- 3 La secuenciación
- 4 La selección

En este apéndice se presenta otra técnica adicional para el diseño de programas, es decir, para elaborar algoritmos, se trata de los diagramas Chapin (Nassi-Schneiderman). Dicha técnica tiene el nombre de sus autores, y son ellos quienes tienen todo el mérito de este invento. En este libro, se han tomado esas ideas como base, conjuntamente con los conceptos de nuestra metodología poniendo a su disposición otro método adicional de diseño de programas (algoritmos). En este trabajo se hacen algunas adaptaciones con el propósito de aprovechar los conceptos de la metodología original de este autor conjuntamente con los conceptos de Chapin (Nassi-Schneiderman), mostrando una técnica que permitirá al estudiante tener otra alternativa para el diseño de programas (algoritmos). El tratamiento que se le ha dado a esta técnica, presupone que el lector ya ha estudiado la metodología original del libro, es decir, en seudocódigo.

## 1. El diagrama Chapin (Nassi-Schneiderman)

El diagrama Chapin (Nassi-Schneiderman) utiliza el bloque de proceso:

Acción

Adentro del bloque se indica la acción a realizar, la cual puede ser leer datos, calcular, imprimir datos, etc.

## 2. Elementos para solucionar problemas en diagramas Chapin (Nassi-Schneiderman)

Los tipos de datos, la forma de dar nombres de identificadores y las operaciones primitivas elementales que utilizaremos aquí, para explicar el uso de los diagramas Chapin (Nassi-Schneiderman), serán los mismos que explicamos en el capítulo 2 de nuestra metodología original:

Ejemplos:

```
Declarar
    Constantes
    ---
    ---
    Tipos
    ---
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
```

Ler nombreEmp, horasTrab, cuotaHora

sueldo = horasTrab \* cuotaHora

Imprimir nombreEmp, sueldo

Adentro del bloque se indica la acción a realizar, la cual puede ser leer datos, calcular, imprimir datos, etc.

### 3. La secuenciación

La secuenciación se representa:

```
Acción 1  
Acción 2  
Acción 3  
...  
Acción N
```

Explicación:

Es una secuencia de bloques de proceso ordenada de arriba hacia abajo, donde cada bloque representa una acción.

Ejemplo:

Nuestro primer algoritmo del capítulo 3 (página 55).

Algoritmo CALCULA SUELDO DE UN EMPLEADO

```
Inicio  
    Declarar  
        Variables  
            nombreEmp: Cadena  
            horasTrab: Entero  
            cuotaHora, sueldo: Real  
    Solicitar    Nombre del empleado,  
                Número de horas trabajadas  
                y Cuota por hora  
    Leer nombreEmp, horasTrab, cuotaHora  
    sueldo = horasTrab * cuotaHora  
    Imprimir nombreEmp, sueldo  
Fin
```

Explicación:

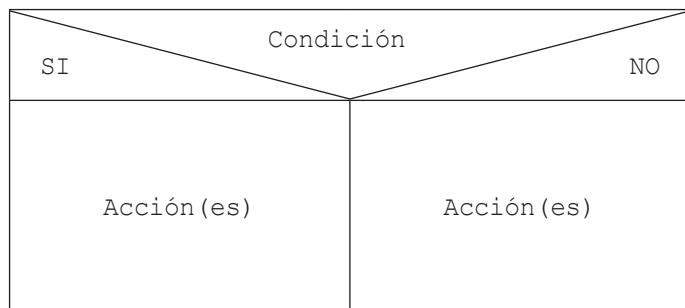
Observe que al inicio se coloca una primera línea de encabezado donde se identifica lo que hace el algoritmo. Luego se coloca un bloque para definir las variables, otro para leer los datos, calcular e imprimir los datos, por último se tiene el fin.

## 4. La selección

La selección tiene tres formas: la doble (if-then-else), la simple (if-then) y la múltiple (switch).

### 4.1 La selección doble (if-then-else)

La selección doble se representa:



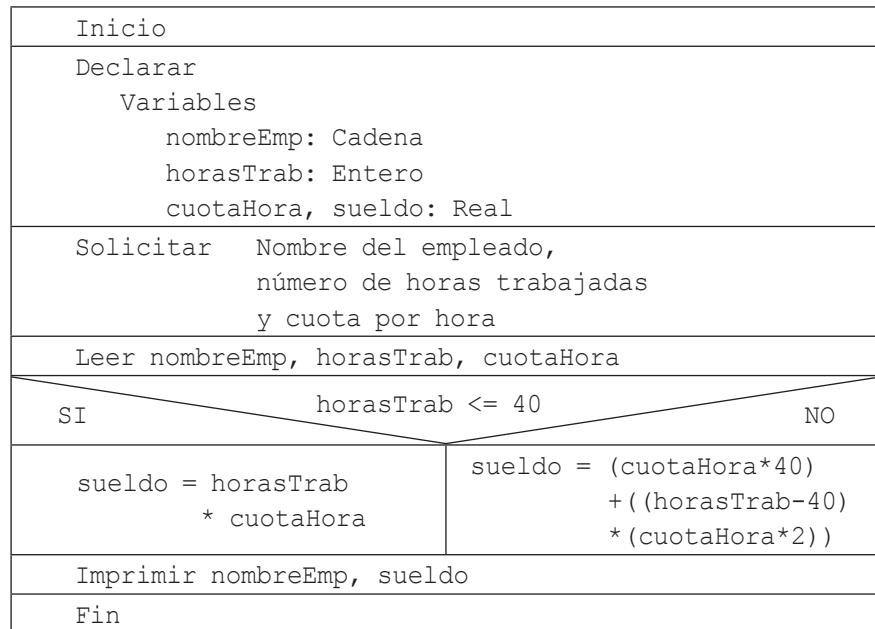
Donde:

- Condición Es una expresión lógica mediante la que se establece la condición que controla la selección.
- SI Es la salida hacia el bloque verdadero (THEN).
- NO Es la salida hacia el bloque falso (ELSE).

Ejemplo:

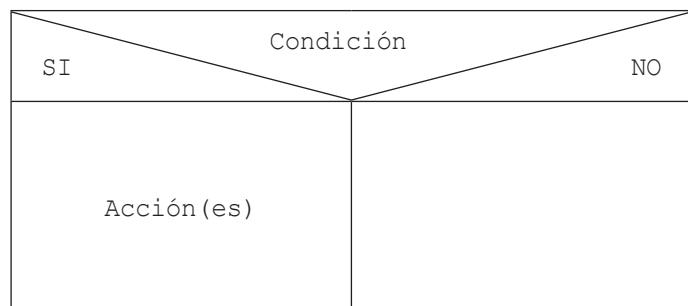
Algoritmo del capítulo 4 (página 73).

Algoritmo CALCULO SUELDO DOBLE



## 4.2 La selección simple (if-then)

La selección simple se representa:

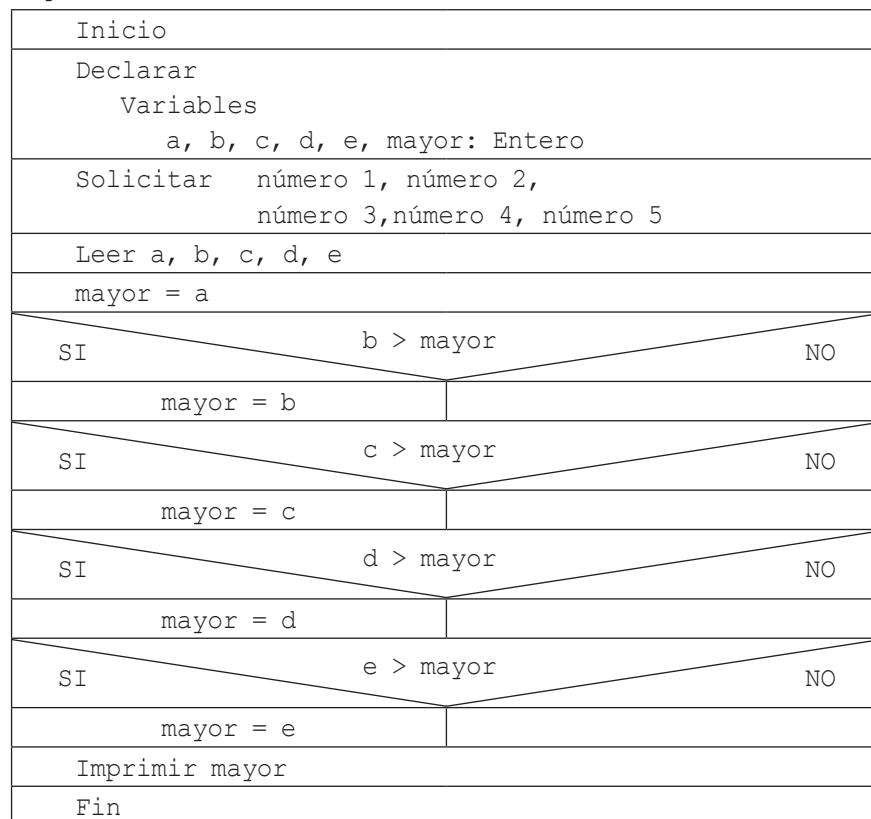


Es decir, se deja en blanco el bloque falso (NO).

Ejemplo:

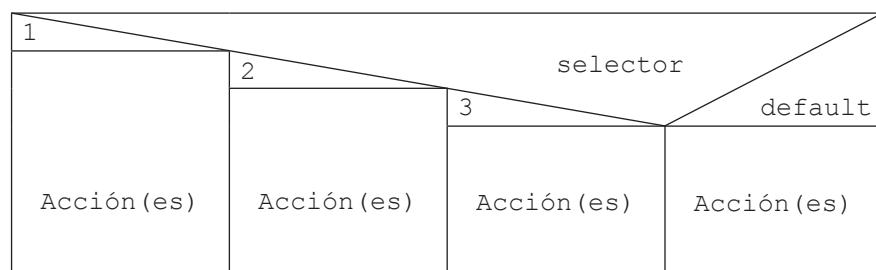
Algoritmo del capítulo 4 (página 88).

#### Algoritmo MAYOR 5 NUMEROS

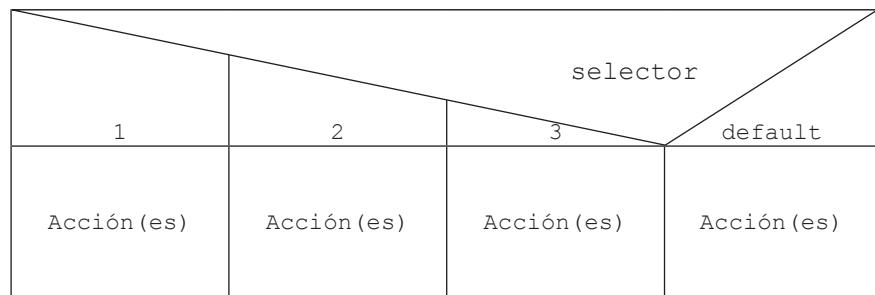


### 4.3 La selección múltiple (switch)

La selección múltiple se representa:



O el formato:



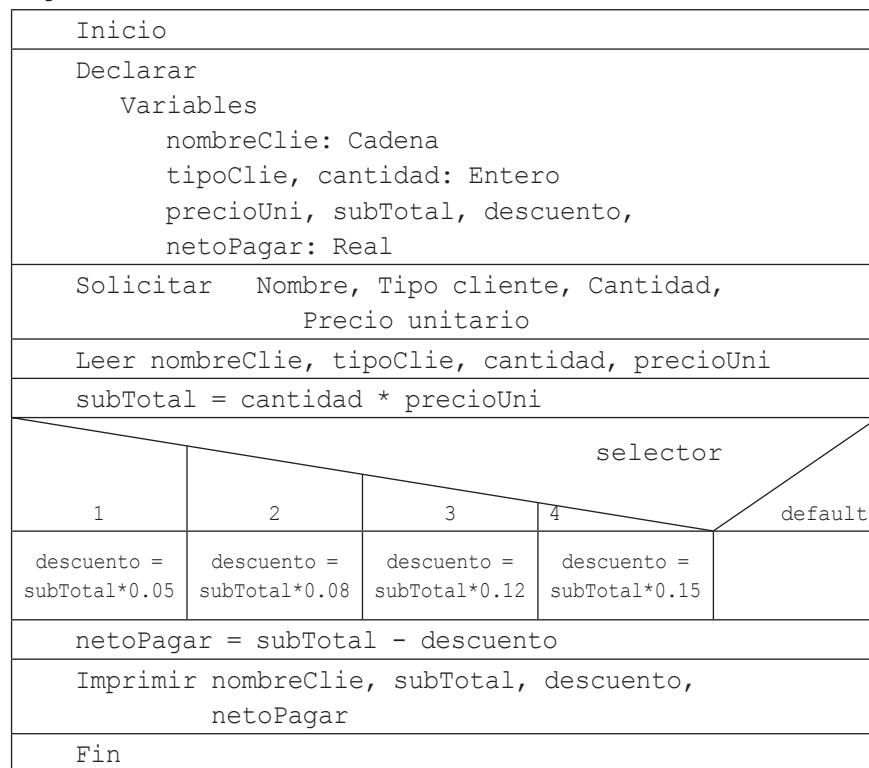
Donde:

- |           |                                                                                                                                                      |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| selector  | Es una variable que toma un valor, y de acuerdo a su valor, va al bloque 1, o al 2, o al 3, etcétera. Puede ser tipo entero, carácter u otro válido. |
| 1,2,3,... | Son las salidas hacia cada bloque.                                                                                                                   |
| default   | Es la salida hacia el bloque falso, es decir, si selector no toma ninguno de los valores indicados.                                                  |

Ejemplo:

Algoritmo del capítulo 4 (página 82).

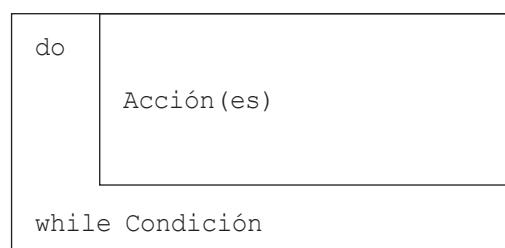
Algoritmo CLIENTE HOJAS HIELO SECO



## 5. La repetición

### 5.1 La repetición do...while

La repetición do...while se representa:



*Explicación:*

La esquera del bloque indica que se repite mientras se cumpla la condición que se indica en la palabra while. Las acciones que quedan dentro del bloque se repiten mientras se cumpla la condición.

**Ejemplo:**

Ejercicio segundo del capítulo 5 (página 92).

**Algoritmo CALCULA SUELDOS DE EMPLEADOS**

|                                                                                                                                          |                                                                                                                                                                                                                                                                                                                              |
|------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Inicio                                                                                                                                   |                                                                                                                                                                                                                                                                                                                              |
| Declarar<br>Variablest<br>nombreEmp: Cadena<br>horasTrab, totEmpleados: Entero<br>cuotaHora, sueldo, totSueldos: Real<br>desea: Carácter |                                                                                                                                                                                                                                                                                                                              |
| Imprimir encabezado                                                                                                                      |                                                                                                                                                                                                                                                                                                                              |
| totEmpleados = 0; totSueldos = 0                                                                                                         |                                                                                                                                                                                                                                                                                                                              |
| do                                                                                                                                       | Solicitar Nombre del empleado, Número de horas trabajadas<br>y Cuota por hora<br>Leer nombreEmp, horasTrab, cuotaHora<br>sueldo = horasTrab * cuotaHora<br>Imprimir nombreEmp, sueldo<br>totEmpleados = totEmpleados + 1<br>totSueldos = totSueldos + sueldo<br>Preguntar ¿Desea procesar otro empleado (S/N)?<br>Leer desea |
| while desea=='S'                                                                                                                         |                                                                                                                                                                                                                                                                                                                              |
| Imprimir totEmpleados, totSueldos                                                                                                        |                                                                                                                                                                                                                                                                                                                              |
| Fin                                                                                                                                      |                                                                                                                                                                                                                                                                                                                              |

**5.2 La repetición for**

La repetición for se representa:

|                                                  |            |
|--------------------------------------------------|------------|
| for contador=valorInicial; condición; incremento |            |
| do                                               |            |
|                                                  | Acción(es) |

Donde:

|              |                                                                                                                                                                                                    |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| contador     | Es una variable de tipo entero o real que funge como contador del ciclo.                                                                                                                           |
| valorInicial | Es el valor inicial que toma la variable.                                                                                                                                                          |
| condición    | Es una condición que evalúa si no ha llegado al valor final que deberá tomar la variable contador; si la condición se cumple, entra al ciclo; si no, se va a la siguiente acción después del ciclo |
| incremento   | Es el incremento o decremento que marcará los intervalos del conteo del contador del ciclo.                                                                                                        |

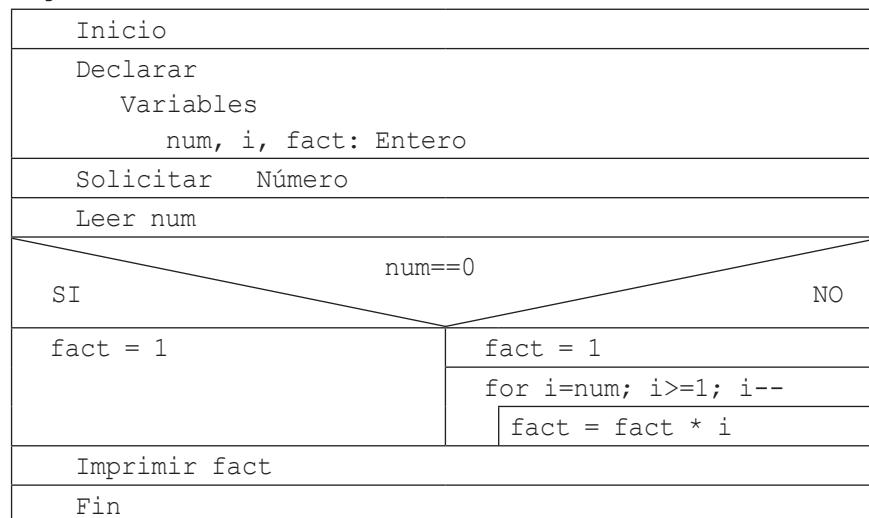
*Explicación:*

Adentro de la escuadra se colocan las acciones que se ejecutarán dentro del ciclo for.

Ejemplo:

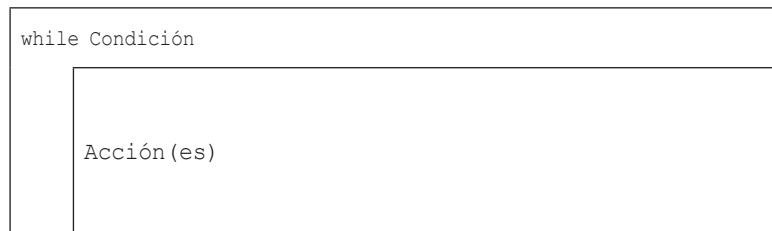
Algoritmo del capítulo 5 (página 133).

#### Algoritmo FACTORIAL



### 5.3 La repetición while

La repetición while se representa:



*Explicación:*

Adentro de la escuadra se colocan las acciones que se ejecutarán dentro del ciclo while, el cual se repetirá mientras se cumpla la condición.

Ejemplo:

Algoritmo del capítulo 5 (página 150).

Algoritmo VENDEDORES DE AUTOMOVILES

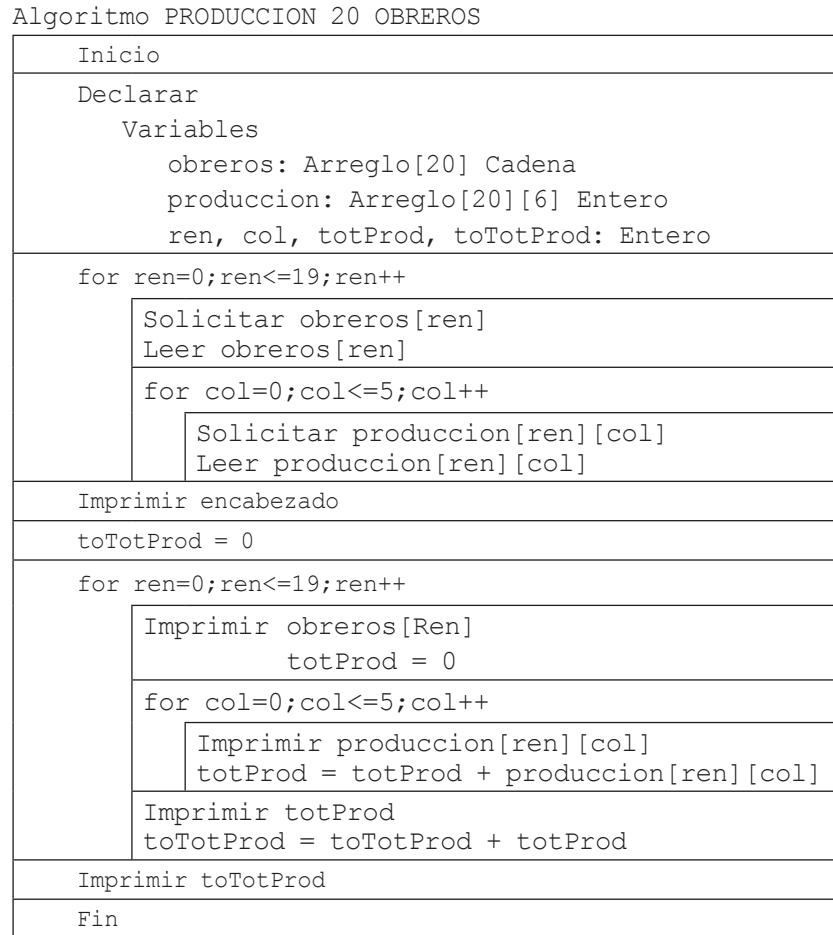
|                                                                                                                                                                            |                                                                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Inicio<br>Declarar<br>Variables<br>nombreVend: Cadena<br>desea, otro: Carácter<br>totAutos, totVend: Entero<br>precioAuto, salMin, sueldo,<br>totSueldos, totVendido: Real |                                                                                                                                                                                                     |
| Solicitar el Salario mínimo<br>Leer salMin<br>Imprimir Encabezado<br>totSueldos = 0<br>totVendido = 0                                                                      |                                                                                                                                                                                                     |
| do                                                                                                                                                                         | Solicitar el Nombre del vendedor<br>Leer nombreVend<br>totAutos = 0<br>totVendido = 0<br>Preguntar "¿Hay auto vendido (S/N)?"<br>Leer desea                                                         |
|                                                                                                                                                                            | while desea == 'S'<br>Solicitar el precio del auto<br>Leer precioAuto<br>totAutos = totAutos + 1<br>totVendido = totVendido + precioAuto<br>Preguntar "¿Hay otro auto vendido (S/N)?"<br>Leer desea |
|                                                                                                                                                                            | sueldo=salMin+(totAutos *100)+(totVendido*0.02)<br>Imprimir nombreVend, sueldo<br>totVend = totVend + 1<br>totSueldos = totSueldos + sueldo<br>Preguntar "¿Hay otro vendedor (S/N)?"<br>Leer otro   |
|                                                                                                                                                                            | while otro == 'S'<br>Imprimir totVend, totSueldos                                                                                                                                                   |
|                                                                                                                                                                            | Fin                                                                                                                                                                                                 |

## 6. Arreglos

A continuación se tiene un ejemplo usando arreglos.

Ejemplo:

Algoritmo del capítulo 6 (página 177).



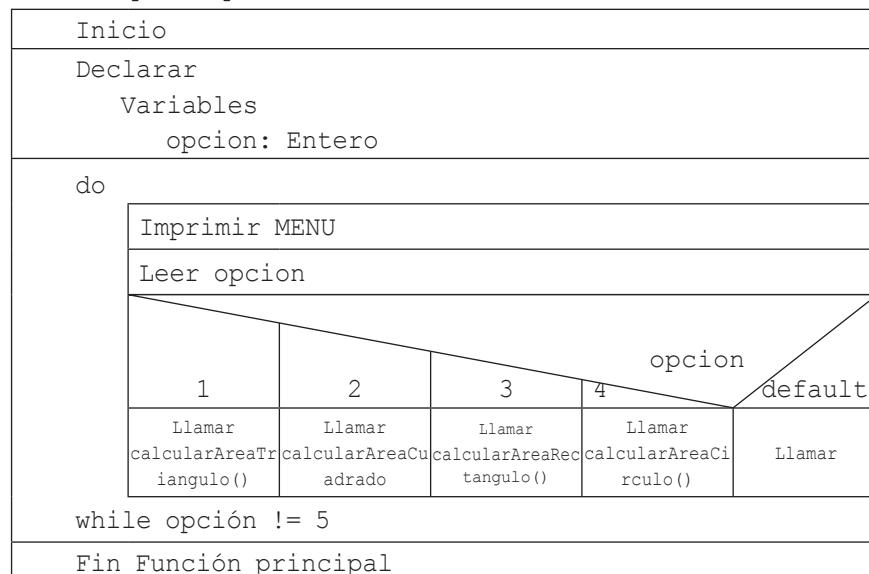
## 7. Diseño descendente

Ahora tenemos un ejemplo usando funciones. Se tiene la función principal(); y las funciones subordinadas calcularAreaTriangulo(), calcularAreaCuadrado(), calcularAreaRectangulo() y calcularAreaCirculo(). Mismas que son llamadas desde la función principal().

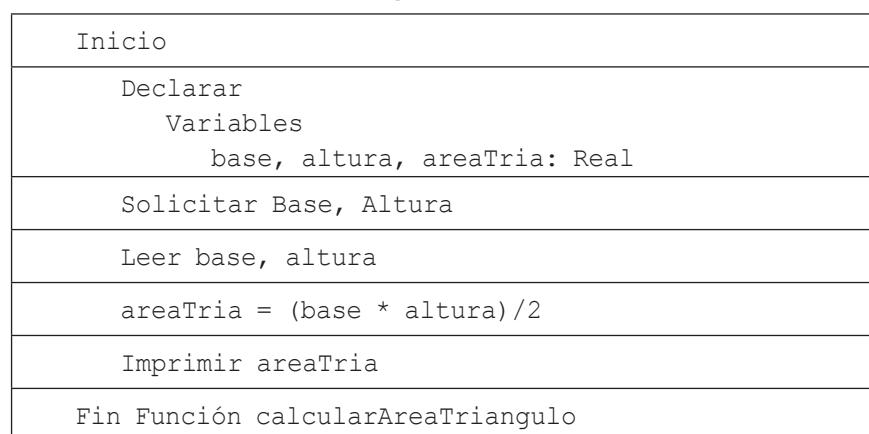
Algoritmo del capítulo 7 (página 213).

Algoritmo AREAS

Función principal()



Función calcularAreaTriangulo()



## Función calcularAreaCuadrado()

|                                   |
|-----------------------------------|
| Inicio                            |
| Declarar                          |
| Variables                         |
| lado, areaCuad: Real              |
| Solicitar Lado                    |
| Leer lado                         |
| areaCuad = Potencia(lado,2)       |
| Imprimir areaCuad                 |
| Fin Función calcularAreaTriangulo |

## Función calcularAreaRectangulo()

|                                    |
|------------------------------------|
| Inicio                             |
| Declarar                           |
| Variables                          |
| areaRec, base, altura: Real        |
| Solicitar Base, Altura             |
| Leer base, altura                  |
| areaRec = base * altura            |
| Imprimir areaRec                   |
| Fin Función calcularAreaRectangulo |

## Función calcularAreaCirculo()

|                                   |
|-----------------------------------|
| Inicio                            |
| Declarar                          |
| Constantes                        |
| PI = 3.14159265                   |
| Variables                         |
| areaCirc, radio: Real             |
| Solicitar Radio                   |
| Leer radio                        |
| areaCirc = PI * Potencia(radio,2) |
| Imprimir areaCirc                 |
| Fin Función calcularAreaCirculo   |

Nota: Se recomienda elaborar algunos de los algoritmos de los ejercicios resueltos o propuestos de los capítulos 3,4,5,6 y 7.

# E

## Apéndice E: Seudocódigo castellanizado (español estructurado)

Algunos autores y maestros utilizan seudocódigos castellanizados para representar las estructuras lógicas de control, es decir, en lugar de if-then-else utilizan si-entonces-sino, en lugar de for utilizan para o desde, en lugar de while utilizan mientras, en lugar de do while utilizan hacer hasta, en lugar de switch utilizan según; de acuerdo con mi experiencia, he comprobado que no es buena idea, porque las estructuras de la programación tienen sus palabras reservadas originales en inglés y según mi opinión, no deben ser cambiadas; el alumno debe aprender la estructura if-then-else y no si-entonces-sino; for y no para o desde; while y no mientras; switch y no según; do while y no hacer hasta.

Yo pienso que para el alumno es mejor aprender esas palabras en su forma original en inglés, porque si las aprende en castellano, después tendrá que aprenderlas en inglés, le costará más trabajo y algo de confusión.

Para que vean la diferencia, en este apéndice se presentan algunos de los algoritmos que se desarrollaron en los capítulos 3, 4, 5, 6 y 7; en una versión castellanizada de las palabras if then else do for while; para que los revisen y vean que es mejor utilizar esas palabras en su versión original en inglés y no castellanizadas. Además estos ejemplos le pueden ayudar a cambiar la forma de utilizar las palabras reservadas de la programación.

Al desarrollar éstos algoritmos, se utilizarán los lineamientos que se estudiaron en el capítulo 2 para representar los tipos de datos y las operaciones primitivas elementales como: Declarar, Solicitar, Leer, Calcular, Imprimir.

Nuestro primer algoritmo del capítulo 3 (página 55).

```
Algoritmo CALCULA SUELDO DE UN EMPLEADO
Declarar
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
    Solicitar    Nombre del empleado, número de horas
                trabajadas y cuota por hora
    Leer nombreEmp, horasTrab, cuotaHora
    sueldo = horasTrab * cuotaHora
    Imprimir nombreEmp, sueldo
Fin
```

Algoritmo del capítulo 4 (página 74).

```
Algoritmo CALCULO SUELDO DOBLE
Declarar
    Variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
    Solicitar    Nombre del empleado,
                número de horas trabajadas
                y cuota por hora
    Leer nombreEmp, horasTrab, cuotaHora
    si horasTrab <= 40 entonces
        sueldo = horasTrab * cuotaHora
    si no
        sueldo=(cuotaHora*40)+((horasTrab-40)*(cuotaHora*2))
    fin si
    Imprimir nombreEmp, sueldo
Fin
```

**Algoritmo del capítulo 4 (página 88).**

```

Algoritmo MAYOR 5 NUMEROS
Declarar
    Variables
        a, b, c, d, e, mayor: Entero
    Solicitar número 1, número 2, número 3, número 4, número 5
    Leer a, b, c, d, e
    mayor = a
    si b > mayor entonces
        mayor = b
    fin si
    si c > mayor entonces
        mayor = c
    fin si
    si d > mayor entonces
        mayor = d
    fin si
    si e > mayor entonces
        mayor = e
    fin si
    Imprimir mayor
Fin

```

**Algoritmo del capítulo 4 (página 94).**

```

Algoritmo CLIENTE HOJAS HIELO SECO
Declarar
    Variables
        nombreClie: Cadena
        tipoClie, cantidad: Entero
        precioUni, subTotal, descuento, netoPagar: Real
    Solicitar Nombre, Tipo cliente, Cantidad, Precio unitario
    Leer nombreClie, tipoClie, cantidad, precioUni
    subTotal = cantidad * precioUni
    según tipoClie
        1: descuento = subTotal * 0.05
        2: descuento = subTotal * 0.08
        3: descuento = subTotal * 0.12
        4: descuento = subTotal * 0.15
    fin según
    netoPagar = subTotal - descuento
    Imprimir nombreClie, subTotal, descuento, netoPagar
Fin

```

Ejercicio segundo del capítulo 5 (página 106).

```
Algoritmo CALCULA SUELDOS DE EMPLEADOS
Declarar
    Variables
        nombreEmp: Cadena
        horasTrab, totEmpleados: Entero
        cuotaHora, sueldo, totSueldos: Real
        desea: Carácter
    Imprimir encabezado
    totEmpleados = 0
    totSueldos = 0
    hacer
        Solicitar Nombre del empleado, Número de
            horas trabajadas y Cuota por hora
        Leer nombreEmp, horasTrab, cuotaHora
        sueldo = horasTrab * cuotaHora
        Imprimir nombreEmp, sueldo
        totEmpleados = totEmpleados + 1
        totSueldos = totSueldos + sueldo
        Preguntar ¿Desea procesar otro empleado (S/N)?
        Leer desea
        mientras desea=='S'
        Imprimir totEmpleados, totSueldos
    Fin
```

Algoritmo del capítulo 5 (página 133).

```
Algoritmo FACTORIAL
Declarar
    Variables
        num, i, fact: Entero
    Solicitar Número
    Leer num
    si num == 0 entonces
        fact = 1
    si no
        fact = 1
        para i = num hasta 1 decremento -1
            fact = fact * i
        fin para
    fin si
    Imprimir fact
Fin
```

**Algoritmo del capítulo 5 (página 150).**

```
Algoritmo VENDEDORES DE AUTOMOVILES
Declarar
    Variables
        nombreVend: Cadena
        desea, otro: Carácter
        totAutos, totVend: Entero
        precioAuto, salMin, sueldo, totSueldos,
        totVendido: Real
    Solicitar el Salario mínimo
    Leer salMin
    Imprimir Encabezado
    totSueldos = 0
    totVend = 0
    hacer
        Solicitar el Nombre del vendedor
        Leer nombreVend
        totAutos = 0
        totVendido = 0
        Preguntar "¿Hay auto vendido (S/N) ?"
        Leer desea
        mientras desea == 'S'
            Solicitar el precio del auto
            Leer precioAuto
            totAutos = totAutos + 1
            totVendido = totVendido + precioAuto
            Preguntar "¿Hay otro auto vendido (S/N) ?"
            Leer desea
            fin mientras
            sueldo=salMin+(totAutos *100)+(totVendido*0.02)
            Imprimir nombreVend, sueldo
            totVend = totVend + 1
            totSueldos = totSueldos + sueldo
            Preguntar "¿Hay otro vendedor (S/N) ?"
            Leer otro
            mientras otro == 'S'
                Imprimir totVend, totSueldos
            Fin
```

Algoritmo del capítulo 6 (página 177).

```
Algoritmo PRODUCCION 20 OBREROS
Declarar
    Variables
        obreros: Arreglo[20] Cadena
        produccion: Arreglo[20][6] Entero
        ren, col, totProd, toTotProd: Entero
    para ren = 1 hasta 20 incremento 1
        Solicitar obreros[ren]
        Leer obreros[ren]
        para col = 1 hasta 12 incremento 1
            Solicitar produccion[ren][col]
            Leer produccion[ren][col]
        fin para
    fin para
    Imprimir encabezado
    toTotProd = 0
    para ren = 1 hasta 20 incremento 1
        Imprimir obreros[ren]
        totProd = 0
        para col = 1 hasta 12 incremento 1
            Imprimir produccion[ren][col]
            totProd = totProd + produccion[ren][col]
        fin para
        Imprimir totProd
        toTotProd = toTotProd + totProd
    fin para
    Imprimir toTotProd
Fin
```

**Algoritmo del capítulo 7 (página 213).**

```

Algoritmo AREAS
    Función principal()
        Declarar
            Variables
                opcion: Entero
        hacer
            Imprimir MENU
            

AREAS FIGURAS GEOMETRICAS
1. TRIANGULO
2. CUADRADO
3. RECTANGULO
4. CIRCULO
5. FIN
ESCOGER OPCION:


            Leer opcion
            según opcion
                1: Llamar calcularAreaTriangulo()
                2: Llamar calcularAreaCuadrado()
                3: Llamar calcularAreaRectangulo()
                4: Llamar calcularAreaCirculo()
            fin según
            mientras opción != 5
        Fin Función principal

        Función calcularAreaTriangulo()
            Declarar
                Variables
                    base, altura, areaTria: Real
            Solicitar Base, Altura
            Leer base, altura
            areaTria = (base * altura) /2
            Imprimir areaTria
        Fin Función calcularAreaTriangulo
    
```

```
Función calcularAreaCuadrado()
    Declarar
        Variables
            lado, areaCuad: Real
    Solicitar Lado
    Leer lado
    areaCuad = Potencia(lado,2)
    Imprimir areaCuad
Fin Función calcularAreaCuadrado
```

```
Función calcularAreaRectangulo()
    Declarar
        Variables
            areaRec, base, altura: Real
    Solicitar Base, Altura
    Leer base, altura
    areaRec = base * altura
    Imprimir areaRec
Fin Función calcularAreaRectangulo
```

```
Función calcularAreaCirculo()
    Declarar
        Constantes
            PI = 3.14159265
        Variables
            areaCirc, radio: Real
    Solicitar Radio
    Leer radio
    areaCirc = PI * Potencia(radio,2)
    Imprimir areaCirc
Fin Función calcularAreaCirculo
```

Nota: Se recomienda elaborar algunos de los algoritmos de los ejercicios resueltos o propuestos de los capítulos 3,4,5,6 y 7.

## Bibliografía

---

- Alcalde,E. y García,M.**, Metodología de la programación, España, Mc Graw Hill, 1987.
- Bell, D. y Parr, M.**, Java para estudiantes Tercera edición, México, Prentice Hall, 2003.
- Booch, G.**, Análisis y diseño orientado a objetos con aplicaciones Segunda edición, USA, México, Addison-Wesley/Díaz de Santos, 1996.
- Booch, G., Rumbaugh, J. y Jacobson, I.**, UML El lenguaje unificado de modelado, España, Addison Wesley, 1999.
- Ceballos, F.J.**, Java 2 Curso de programación, México, Alfaomega-Rama, 2000.
- Dahl,O.J., Dijkstra, E.W., y Hoare,C.A.R.**, Structured programming, USA, Academic Press, 1972.
- Deitel, H.M. y Deitel, P.J.**, Como programar en Java Quinta edición, México, Pearson Prentice Hall, 2004.
- Horstmann, C.S. y Cornell, G.**, Core Java 2 Volume I Fundamentals 5th edition, USA, The Sun Microsystems Press Prentice Hall, 2000.
- Horton, I.**, Beginning Java 2, USA, Wrox Press Inc., 2000.
- Jacobson, I., Booch, G. y Rumbaugh, J.**, UML El proceso unificado de desarrollo de software, España, Addison Wesley, 2000.
- Joyanes, A.L.**, Fundamentos de programación Segunda edición, España, Mc Graw Hill, 1996.
- Joyanes, A.L.**, Problemas de metodología de la programación, España, Mc Graw Hill, 1990.
- Joyanes, A.L.**, Programación orientada a objetos Segunda edición, España, Osborne Mc Graw Hill, 1998.
- Joyanes, A.L. y Fernández, A.M.**, Java 2 Manual de programación, España, Mc Graw Hill, 2001.
- Korthage, R.**, Lógica y algoritmos, México, Limusa, 1967.
- Lemay, L. y Cadenhead, R.**, Aprendiendo Java en 21 días, México, Pearson Prentice Hall, 1999.
- López, R.L.**, Metodología de la programación orientada a objetos, México, Alfaomega, 2006.
- López,R.L.**, Programación estructurada en lenguaje C, México, Alfaomega, 2005.
- López,R.L.**, Programación estructurada un enfoque algorítmico Segunda edición, México, Alfaomega, 2003.
- Meyer, B.**, Construcción de software orientado a objetos Segunda edición, España, Prentice Hall, 1999.
- Rumbaugh, J., Jacobson, I. y Booch, G.**, UML El lenguaje unificado de modelado. Manual de referencia, España, Addison Wesley, 2000.
- Schildt, Herbert**, Fundamentos de programación en Java 2, Colombia, Osborne Mc Graw Hill, 2002.
- Schildt, Herbert**, Java 2 Manual de referencia, España, Osborne Mc Graw Hill, 2001.
- Spencer,D.**, Problemas para resolver con computadoras, México, Limusa, 1985.
- Warnier,J.D.**, Construcción de programas, España, Editores Técnicos Asociados, 1979.
- Watkins,R.**, Solución de problemas por medio de computadoras, México, Limusa, 1984.
- Wu, T.C.**, Introducción a la programación orientada a objetos en Java, España, Mc Graw Hill, 2001.