

# PROGRAMACION en COBOL ESTRUCTURADO

LAWRENCE R. NEWCOMER, M.S.

*Assistant professor of  
computer science  
Pennsylvania State University*

TRADUCCION:  
**Alfonso Camañó Liceras**  
Licenciado en Ciencias Matemáticas

Revisión técnica:  
**Antonio Vaquero Sánchez**  
Catedrático de Informática  
Facultad de Ciencias Físicas  
Universidad Complutense de Madrid

## McGRAW-HILL

MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • LISBOA  
MADRID • NUEVA YORK • PANAMÁ • SAN JUAN • SANTIAGO • SÃO PAULO  
AUCKLAND • HAMBURGO • LONDRES • MONTREAL • NUEVA DELHI  
PARÍS • SAN FRANCISCO • SINGAPUR • ST. LOUIS  
SIDNEY • TOKIO • TORONTO

**CONSULTORES EDITORIALES**  
**AREA DE INFORMATICA Y COMPUTACION**

**Antonio Vaquero Sánchez**

Catedrático de Informática  
Facultad de Ciencias Físicas  
Universidad Complutense de Madrid  
ESPAÑA

**María Lourdes Fournier García**

Actuaria, Facultad Ciencias UNAM  
Profesora Asociada a Tiempo Completo  
Universidad Autónoma Metropolitana  
MEXICO

**Gerardo Quiroz Vieyra**

Ingeniero en Comunicaciones y Electrónica  
Escuela Superior de Ingeniería Mecánica y Eléctrica IPN  
Carter Wallace, S. A.  
Universidad Autónoma Metropolitana  
Docente DCSA  
MEXICO

**Alfonso Pérez Gama**

Ingeniero Electrónico  
Universidad Nacional de Colombia  
COLOMBIA

**José Portillo**

Universidad de Lima  
PERU

**PROGRAMACION  
en  
COBOL ESTRUCTURADO**

LAWRENCE R. NEWCOMER es Profesor Agregado de Informática en la Universidad del Estado de Pensilvania. Es Diplomado en inglés por la Universidad de Wesleyan (Connecticut), Licenciado en Psicología por la Universidad del Estado de Millersville y Licenciado en Informática por la Universidad del Estado de Pensilvania. Además de enseñar aplicaciones comerciales de la informática, el profesor Newcomer ha publicado artículos sobre la enseñanza de la informática, ha producido diversos programas comerciales y ha dirigido numerosos seminarios y exposiciones para la industria.

#### PROGRAMACIÓN EN COBOL ESTRUCTURADO

Prohibida la reproducción total o parcial de esta obra,  
por cualquier medio, sin autorización escrita del editor.

DERECHOS RESERVADOS © 1986, respecto a la primera edición en español por  
McGRAW-HILL INTERAMERICANA DE MEXICO, S. A. de C. V.  
Atiacomulco 499-501, Fracc. Ind. San Andrés Atoto  
53500 Naucalpan de Juárez, Edo. de México  
Miembro de la Cámara Nacional de la Industria Editorial, Reg. Núm. 1890

**ISBN 968-422-007-3**  
(ISBN 968-451-769-6)

Traducido de la primera edición en inglés de  
PROGRAMMING WITH STRUCTURED COBOL

Copyright © MCMLXXXIV, by McGraw-Hill, Inc., U. S. A.

ISBN 0-07-037988-X

2345678901 Editito-85 8123456798

Impreso en México Printed in Mexico

Esta obra se terminó de  
imprimir en septiembre de 1988  
en Programas Educativos, S.A. de C.V.  
Calz. Chabacano 65-A  
Col. Asturias  
Delegación Cuauhtémoc  
06850, México, D.F.

Se tiraron 1000 ejemplares

## Prólogo

Este libro trata del COBOL (como se definió por el "American National Standards Institute" en 1974) desde el punto de vista de la *programación estructurada*. Cubre la mayor parte, aunque no todas, de las técnicas que han permitido a los programadores alcanzar mayores niveles de productividad en el desarrollo de *programas individuales*. Se pone especial énfasis en el uso del método de *diseño modular y descendente*, así como en las numerosas normas de codificación en COBOL.

El libro trata de ser exhaustivo y conciso a la vez, permitiendo al lector progresar rápidamente. Al finalizar el Capítulo 2, el lector está en disposición de escribir programas simples en COBOL. Aprender a programar computadoras es como aprender a nadar (hay que lanzarse e intentarlo), por eso el lector deberá resolver, en una máquina real, cuantos más Ejercicios de Programación, mejor. El Capítulo 9 trata de la depuración de programas y está diseñado para que pueda leerse después del Capítulo 2; consulte el Capítulo 9 cuando tenga problemas con un Ejercicio de Programación.

Si aparecen extensiones al COBOL de 1974 norma ANS, están siempre referidas al COBOL del sistema IBM OS/VS y, en todo caso, se indica. Todos los ejemplos y problemas fueron comprobados en la computadora IBM 3081 de la Universidad del Estado de Pensilvania con el COBOL del sistema IBM OS/VS. Cuando hay diferencias entre distintas computadoras, se da la información correspondiente a los sistemas del tipo IBM-370 con sistemas operativos OS/VS o similar. El "American National Standards Institute" está en la actualidad trabajando en una versión de 1980 para el COBOL ANS. Los posibles cambios a introducir en el COBOL de 1980 se dan en el Apéndice C.

Quiero agradecer su ayuda a la Universidad, en particular al "campus" de York, a Diane Anderson, su valiosa colaboración en la preparación del manuscrito; a Marge Johnson, su ayuda constante en las comunicaciones; a Frank Cable, por su oportunidad inicial; a Mike Perelman, su inquebrantable entusiasmo; a Bob Rodgers y Carol Miller, sus útiles comentarios, y a Debby Lord sus sugerencias y sus varios ejercicios de programación. Ha sido un placer trabajar con David Beckwith, Marthe Grice y el resto de las personas de McGraw-Hill. Un agradecimiento especial debo a mis padres por educarme y sobre todo a mi esposa que ha hecho posible el libro.

LAWRENCE NEWCOMER

### Nota

Para información y guía del usuario a continuación se extracta el "Government Printing Office Form Number 1965-0795689".

Cualquier organización interesada en reproducir total o parcialmente el informe y especificaciones COBOL, o en utilizar las ideas del informe como base para manuales de instrucciones o cualquier otro propósito, puede hacerlo libremente. Sin embargo, todas estas organizaciones están obligadas a reproducir esta sección en la introducción al documento. Quienes utilicen un pasaje corto, están obligados a mencionar "COBOL" para identificar la fuente, pero no deben transcribir toda esta sección.

COBOL es un lenguaje industrial y no es propiedad de ninguna compañía o grupo de compañías, ni tampoco de ninguna organización o grupo de organizaciones.

No se otorga ninguna garantía expresa ni implícita por ningún contribuyente ni por el Comité COBOL sobre la precisión y funcionamiento del lenguaje de programación. Más aún, no se asume responsabilidad por ningún contribuyente ni por el Comité relacionada con lo anterior.

Se han establecido procedimientos para el mantenimiento de COBOL. Las preguntas y las proposiciones de modificaciones deben dirigirse al "Executive Committee" de la "Conference of Data Systems Languages".

Los autores y propietarios de los derechos del material usado aquí son:

FLOW-MATIC (Trademark of Sperry Rand Corporation). Programming for the UNIVAC® I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

ellos han autorizado específicamente el uso de este material total o parcialmente en las especificaciones de COBOL. Tales autorizaciones se extienden a la reproducción y uso de las especificaciones de COBOL en manuales de programación y publicaciones similares.

## Contenido

<b>Capítulo 1 CONCEPTOS FUNDAMENTALES DE PROCESAMIENTO DE DATOS .....</b>	<b>1</b>
1.1 Funcionamiento de una computadora .....	1
1.2 Entrada .....	1
1.3 Salida .....	2
1.4 Almacenamiento auxiliar .....	2
1.5 Memoria .....	3
1.6 ALU .....	4
1.7 CU .....	4
1.8 Lenguajes de programación .....	4
1.9 Sistemas operativos .....	6
1.10 Ficheros .....	6
1.11 Códigos de datos .....	7
<hr/>	
<b>Capítulo 2 GENERALIDADES DE COBOL .....</b>	<b>9</b>
2.1 Las DIVISIONES en COBOL .....	10
2.2 Secciones (SECTION) y párrafos .....	12
2.3 Reglas para la escritura de programas COBOL .....	13
2.4 Nombres definidos por el programador y palabras reservadas .....	16
2.5 Comienzo .....	17
<hr/>	
<b>Capítulo 3 DIVISION DE IDENTIFICACION (IDENTIFICATION DIVISION). .</b>	<b>29</b>
3.1 Notación sintáctica .....	29
3.2 Sintaxis de la división de identificación .....	30
3.3 Párrafo PROGRAM-ID .....	30
3.4 Párrafos opcionales .....	31
<hr/>	
<b>Capítulo 4 DIVISION DEL ENTORNO (ENVIRONMENT DIVISION) .....</b>	<b>34</b>
4.1 Sintaxis de la división del entorno .....	34
4.2 CONFIGURATION SECTION: Párrafo SOURCE-COMPUTER ..	35
4.3 CONFIGURATION SECTION: Párrafo OBJECT-COMPUTER ..	36
4.4 CONFIGURATION SECTION: Párrafo SPECIAL-NAMES .....	36
4.5 Sección de entrada-salida (INPUT-OUTPUT SECTION) .....	38
4.6 Ejemplos de la división del entorno .....	42
<hr/>	
<b>Capítulo 5 DIVISION DE DATOS (DATA DIVISION) .....</b>	<b>49</b>
5.1 Estructura de la división de datos .....	49
5.2 Sección de ficheros (FILE SECTION) .....	50

## CONTENIDO

5.3 FD BLOCK CONTAINS...	52
5.4 FD RECORD CONTAINS...	54
5.5 FD LABEL RECORDS...	55
5.6 FD DATA RECORDS...	55
5.7 FD LINAGE...	56
5.8 Descripción de registros en la sección de ficheros	58
5.9 Sintaxis de la descripción de datos	61
5.10 Cláusula PICTURE	62
5.11 Cláusula USAGE	69
5.12 Cláusula BLANK	71
5.13 Cláusula JUSTIFIED	72
5.14 Cláusula OCCURS	72
5.15 Cláusula SIGN	73
5.16 Cláusula SYNCHRONIZED	74
5.17 Cláusula REDEFINES	74
5.18 Determinación de la longitud de un elemento de datos	76
5.19 Sección de almacenamiento de trabajo (WORKING-STORAGE SECTION)	77
5.20 Normas de codificación	80

---

<b>Capítulo 6 DIVISION DE PROCEDIMIENTOS (PROCEDURE DIVISION) ..</b>	<b>94</b>
--	-----------

6.1 Introducción: Normas de codificación	94
6.2 Entrada/salida: Instrucción OPEN	94
6.3 Entrada/salida: Instrucción CLOSE	97
6.4 Entrada: Instrucción READ	99
6.5 Salida: Instrucción WRITE	100
6.6 Entrada: Instrucción ACCEPT	106
6.7 Salida: Instrucción DISPLAY	108
6.8 Procesamiento: Instrucción MOVE	110
6.9 Fin de programa: Instrucción STOP	113
6.10 Ejecución controlada de un párrafo: Instrucción PERFORM	114
6.11 Procesamiento: Instrucción ADD	119
6.12 Procesamiento: Truncamiento y opción ROUNDED	121
6.13 Procesamiento: Desbordamiento y la cláusula ON SIZE ERROR	121
6.14 Procesamiento: Instrucción SUBTRACT	122
6.15 Procesamiento: Instrucción MULTIPLY	123
6.16 Procesamiento: Instrucción DIVIDE	125
6.17 Procesamiento: Instrucción COMPUTE	127
6.18 Eficiencia en los cálculos aritméticos	130
6.19 Normas de codificación adicionales para la división de procedimientos	130

---

<b>Capítulo 7 LOGICA DE LOS PROGRAMAS ..</b>	<b>155</b>
--	------------

7.1 Diseño lógico: Diagramas de flujo estructurados	155
7.2 Estructuras lógicas de programación	156
7.3 La estructura secuencial en COBOL	160
7.4 La estructura de selección en COBOL	160
7.5 Estructura de selección: Condición de clase de datos	161
7.6 Estructura de selección: Condición relación	162
7.7 Estructura de selección: Condición signo	164
7.8 Estructura de selección: Condición nombre-de-condición	164

## CONTENIDO

7.9 Operadores lógicos y condiciones compuestas .....	167
7.10 Abreviaturas de las condiciones compuestas .....	170
7.11 Instrucciones IF anidadas y lineales .....	170
7.12 Estructuras iterativas: Instrucción PERFORM .....	173
7.13 Instrucciones PERFORM anidadas .....	182
7.14 Diseño lógico: Pseudocódigo .....	184
7.15 Diseño lógico: Tablas de decisión .....	185
<hr/>	
<b>Capítulo 8 DISEÑO DE PROGRAMAS: EL MÉTODO DESCENDENTE Y MODULAR (TOP-DOWN) .....</b>	<b>211</b>
8.1 Paso 1.: Definición del problema .....	211
8.2 Paso 2.: Diseño general del programa .....	211
8.3 Paso 3.: Diseño detallado del programa .....	213
8.4 Paso 4.: Preparación de un plan para la codificación y comprobación del programa .....	215
8.5 Paso 5.: Ensamblaje de datos de prueba .....	216
8.6 Paso 6.: Codificación y comprobación descendente .....	216
8.7 Paso 7.: Confección de la documentación del programa .....	216
<hr/>	
<b>Capítulo 9 DEPURACION .....</b>	<b>238</b>
9.1 Depuración de errores sintácticos .....	238
9.2 Depuración de errores lógicos graves .....	241
9.3 Depuración de errores lógicos leves .....	241
9.4 Obtención de información de trazas del programa .....	241
9.5 Lectura de datos durante la ejecución del programa .....	244
9.6 Las instrucciones DECLARATIVES y USE FOR DEBUGGING .....	248
<hr/>	
<b>Capítulo 10 MANIPULACION DE TABLAS .....</b>	<b>260</b>
10.1 Tablas unidimensionales: La cláusula OCCURS .....	260
10.2 Subíndices .....	260
10.3 Manipulación de tablas unidimensionales .....	262
10.4 Tablas bidimensionales .....	266
10.5 Manipulación de tablas bidimensionales .....	268
10.6 Tablas de longitud variable: Cláusula OCCURS... DEPENDING..	273
10.7 Indices .....	276
10.8 Búsqueda secuencial en una tabla y el verbo SEARCH .....	279
10.9 Búsqueda binaria en una tabla y el verbo SEARCH ALL .....	282
<hr/>	
<b>Capítulo 11 PROCESAMIENTO SECUENCIAL DE FICHEROS .....</b>	<b>297</b>
11.1 Actualización de ficheros secuenciales .....	297
11.2 Algoritmo "maestro-transacciones" .....	298
11.3 Actualización de ficheros en disco .....	314
<hr/>	
<b>Capítulo 12 ORDENACION Y FUSION DE FICHEROS .....</b>	<b>324</b>
12.1 El vocabulario de ordenación de ficheros .....	324
12.2 Utilización de la instrucción SORT .....	325
12.3 Fusión de ficheros con el verbo MERGE .....	342

CONTENIDO

Apéndice <i>A</i>	PALABRAS RESERVADAS DE COBOL .....	351
Apéndice <i>B</i>	SECUENCIAS DE ORDEN DE CARACTERES .....	361
Apéndice <i>C</i>	CONSIDERACIONES AL COBOL DE 1980 .....	368
<hr/>		
INDICE .....		371

# Capítulo 1

## Conceptos fundamentales de procesamiento de datos

### 1.1 FUNCIONAMIENTO DE UNA COMPUTADORA

Los objetos que componen físicamente una *computadora* se denominan material (*hardware*). Desde la perspectiva del programador, un sistema de computadora tiene los componentes funcionales que se detallan en la Figura 1-1.

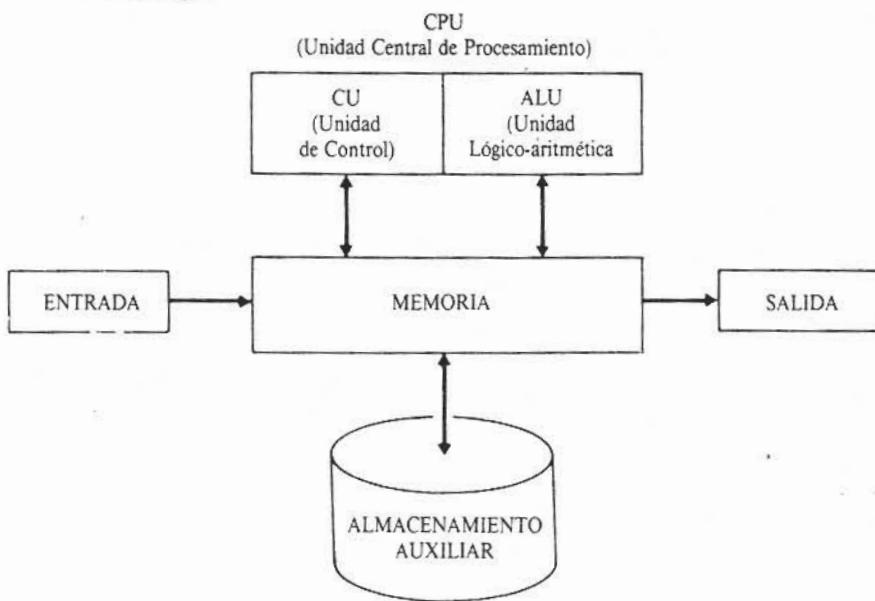


Fig. 1-1

### 1.2 ENTRADA

Las *unidades de entrada* son máquinas que transmiten información desde el mundo exterior a la memoria de la computadora. En la mayor parte de los casos, primero se tendrá que poner la información en un formato a la medida de la unidad de entrada, generalmente mecanografiando o tecleando. Los documentos a partir de los cuales se transcribe la información se denominan *documentos fuente*. Las tres principales unidades de entrada son las siguientes:

#### Lectores de tarjetas

En primer lugar se utiliza una *perforadora de tarjetas*, operada manualmente o por computadora, para registrar la información a modo de agujeros en tarjetas de un tamaño normalizado. A continuación, se coloca el paquete de tarjetas en el lector de tarjetas, que por procedimientos electromecánicos o fotomecánicos aprecia los agujeros de las tarjetas y transmite la información a la memoria de la computadora.

#### Unidad de disco

Una unidad de disco es conceptualmente similar a un sistema de tarjetas, pero registra la información magnéticamente en un *disco flexible* o en un *disco rígido*. El disco se introduce después en una *unidad de disco*, que es capaz de apreciar la información codificada magnéticamente y

transmitirla a la memoria de la computadora. La unidad de disco lee los datos mucho más rápidamente que un lector de tarjetas. Asimismo, a diferencia de las tarjetas, los discos flexible y duro pueden reutilizarse para almacenar nuevos datos.

#### Terminales de impresora y de pantalla (CRT)

Los terminales se pueden conectar a la computadora de modo que sus *teclados* sirvan de unidad de entrada. Cuando se pulsa una tecla, el carácter correspondiente se transmite a la memoria de la computadora. Los *terminales inteligentes* disponen de un *microprocesador* (una computadora en un solo chip) y de algo de memoria; de este modo el terminal es capaz de almacenar los caracteres que se pulsan y transmitirlos después a la memoria de la computadora en grupos (en lugar de uno cada vez). Los terminales inteligentes también pueden llevar a cabo algún procesamiento local de la información por sí mismos. Tanto los terminales inteligentes como los terminales no inteligentes disponen además de unidades de salida: los *terminales de impresora* tienen, como su nombre indica, una impresora, mientras que los *terminales de pantalla*, una pantalla de rayos catódicos (CRT).

### 1.3 SALIDA

Las *unidades de salida* reciben la información de la memoria de la computadora en forma capaz de ser entendida por el hombre o bien por otras máquinas.

#### Impresoras

Estas unidades registran el contenido de la memoria de la computadora en forma de caracteres impresos en papel. En las *impresoras por impacto* se utiliza una cinta entintada como en las máquinas de escribir; otras impresoras utilizan técnicas térmicas, electrostáticas o el láser para construir las imágenes en el papel. Las *impresoras de caracteres* imprimen un carácter cada vez, mientras que las *impresoras de línea* imprimen una línea completa cada vez. Las actuales impresoras láser pueden imprimir más de 20.000 líneas/minuto.

#### Terminales CRT y de impresión

Los periféricos que se describen en la Sección 1.2 funcionan a la vez como periféricos de entrada y salida.

#### Perforadoras de tarjetas

Estos periféricos funcionan bajo control directo de la CPU y perforan información en tarjetas.

### 1.4 ALMACENAMIENTO AUXILIAR

Los periféricos preparados para la entrada y salida pueden servir como *periféricos para almacenamiento auxiliar*, que son necesarios por: (i) incluso la memoria de la computadora más grande no puede almacenar toda la información que precisa un negocio relativamente pequeño y (ii) la memoria de las computadoras modernas se borra cuando se corta la corriente. Por tanto, incluso en el caso de que toda la información cupiera en la memoria, sería inaceptable asumir el riesgo de un corte en el suministro de energía eléctrica.

#### Disco rígido

Este medio es el más importante en el almacenamiento auxiliar y utiliza un soporte conductor para leer y escribir datos en el *paquete de disco*. Un paquete de disco consiste en una pila de platos planos de aluminio cubiertos con un material magnético e insertado en un eje central (Fig. 1-2). Hay un *brazo de acceso* que dispone de cabezas de lectura/escritura (una por cada superficie) y que puede desplazarse hacia dentro, o hacia fuera. Al detenerse el brazo de acceso posa las cabezas

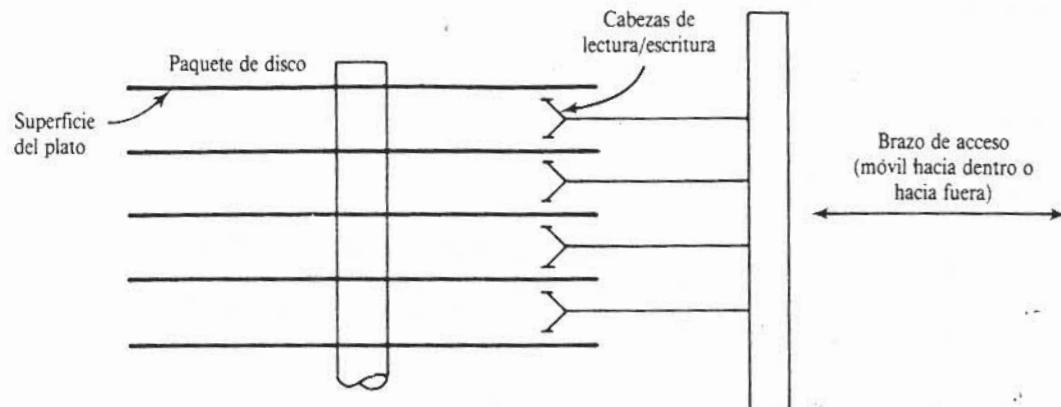


Fig. 1-2

sobre una serie de *pistas* circulares sobre las superficies del disco que está girando; los datos se registran sobre estas pistas. Las pistas se agrupan en conjuntos denominados *cilindros*, que se refieren al grupo de pistas accesibles desde una posición concreta del brazo.

El tiempo medio utilizado para desplazar el brazo de acceso hasta el cilindro deseado (*tiempo de acceso*) está comprendido entre 16 y 60 milisegundos (un milisegundo es 1/1.000 de segundo). A este tiempo deben sumarse el *tiempo rotacional* (tiempo para que un determinado lugar de una pista se sitúe debajo de la cabeza) y el pequeño tiempo para transferir los datos desde o hacia la memoria. El *tiempo total de acceso* es del orden de 30 ms para los discos IBM.

Algunos discos (denominados de *cabeza fija por pista*) disponen de una cabeza de lectura/escritura para cada pista del disco. De este modo el primer tiempo de acceso (para colocar el brazo) es nulo.

### Cinta magnética

Las cintas de una computadora son análogas a las utilizadas por un magnetofón y vienen preparadas en forma de carretes o de casetes.

La mayor desventaja de las cintas como medio de almacenamiento auxiliar es que la unidad debe leer toda la información de la cinta para localizar un dato concreto. Por tanto, la cinta es eficiente sólo si se procesa la información en el mismo orden en que está físicamente en la misma (*procesamiento secuencial*). La información contenida en un disco puede procesarse en cualquier orden que pueda desearse, secuencial o arbitrario (*procesamiento aleatorio*).

## 1.5 MEMORIA

La memoria de la computadora se fabrica con componentes electrónicos cada uno de los cuales puede almacenar un *dígito binario* (*bit*). Como un bit sólo puede tener por valor cero o uno, se agrupan en conjuntos (generalmente de ocho) denominados *bytes*. Un byte tiene suficiente capacidad para almacenar un carácter.

Los bytes de la memoria están numerados 0, 1, 2, 3, ...; el número único asociado a un byte se conoce como su *dirección*. El acceso al azar de una dirección de memoria es cuestión de *nanosegundos* (1 ns es 1/1.000.000.000 de un segundo).

La capacidad de memoria y de almacenamiento auxiliar se mide en términos de KB (*kilobyte*, aproximadamente mil bytes) y MB (*megabyte*, aproximadamente un millón de bytes). Una memoria de alrededor de 128.000 bytes se diría que tiene 128K, mientras que si tuviera 2.000.000 de bytes se diría que tiene 2 MB. Los paquetes de discos tienen entre 29 y 1.260 MB, dependiendo del modelo.

El constante flujo de información hacia la memoria desde los periféricos de entrada y/o almacenamiento auxiliar, el procesamiento de la información mientras está en la memoria y la transferencia final de los resultados a los periféricos de salida o al almacenamiento auxiliar (para registrarlos permanentemente en una forma fácilmente accesible a la computadora) —todo ello constituye lo

que se conoce como *ciclo entrada-proceso-salida* y es característico de casi todas las aplicaciones en el procesamiento de datos.

## 1.6 ALU (ARITMETIC LOGIC UNIT)

La ALU es la parte de la CPU (Central Process Unit) que realmente procesa la información. La ALU recibe los datos que hay que procesar de la memoria y los resultados son enviados a la memoria para su almacenamiento temporal. La ALU realiza dos funciones de procesamiento básicas:

**Aritmética.** Operaciones de suma, resta, multiplicación y división.

**Lógica.** Función por la cual la ALU compara los valores almacenados en dos lugares de memoria diferentes para determinar si son iguales o, si no, cuál es el mayor. Las "decisiones" simples son la base sobre las que se construyen las más complejas y la aparente actividad inteligente de la computadora.

## 1.7 CU (CONTROL UNIT)

Las computadoras modernas operan sobre el concepto de *programa almacenado*, lo que permite cambiar la tarea realizada por la computadora simplemente introduciendo el programa en la memoria. La unidad de control se encarga de *ejecutar* el programa almacenado como sigue:

- (1) Lo primero que hace la CU es *capturar* la instrucción almacenada en la memoria cuya dirección está en un área especial denominada IAR, siglas en inglés de "Instruction Address Register" (Registro de Dirección de Instrucciones).
- (2) A continuación la CU *decodifica* la instrucción, es decir, la interpreta para determinar lo que hay que hacer (*operación* a llevar a cabo) y los datos que tiene que utilizar en la operación (direcciones de memoria de los *operando*s).
- (3) La CU después sustituye la dirección contenida en la IAR con la correspondiente a la siguiente instrucción almacenada.
- (4) Por último, la CU envía las señales al resto del sistema para asegurarse que la operación se ejecuta. Así, en el caso de una suma, la unidad de control exigirá que la memoria envíe copias de los valores a sumar a la ALU. Después obligará a la ALU a ejecutar la suma y enviar el resultado a la dirección de la memoria deseada. Para una operación de entrada o salida, la CU activará al *periférico de entrada/salida* y a la memoria para transferir los datos.

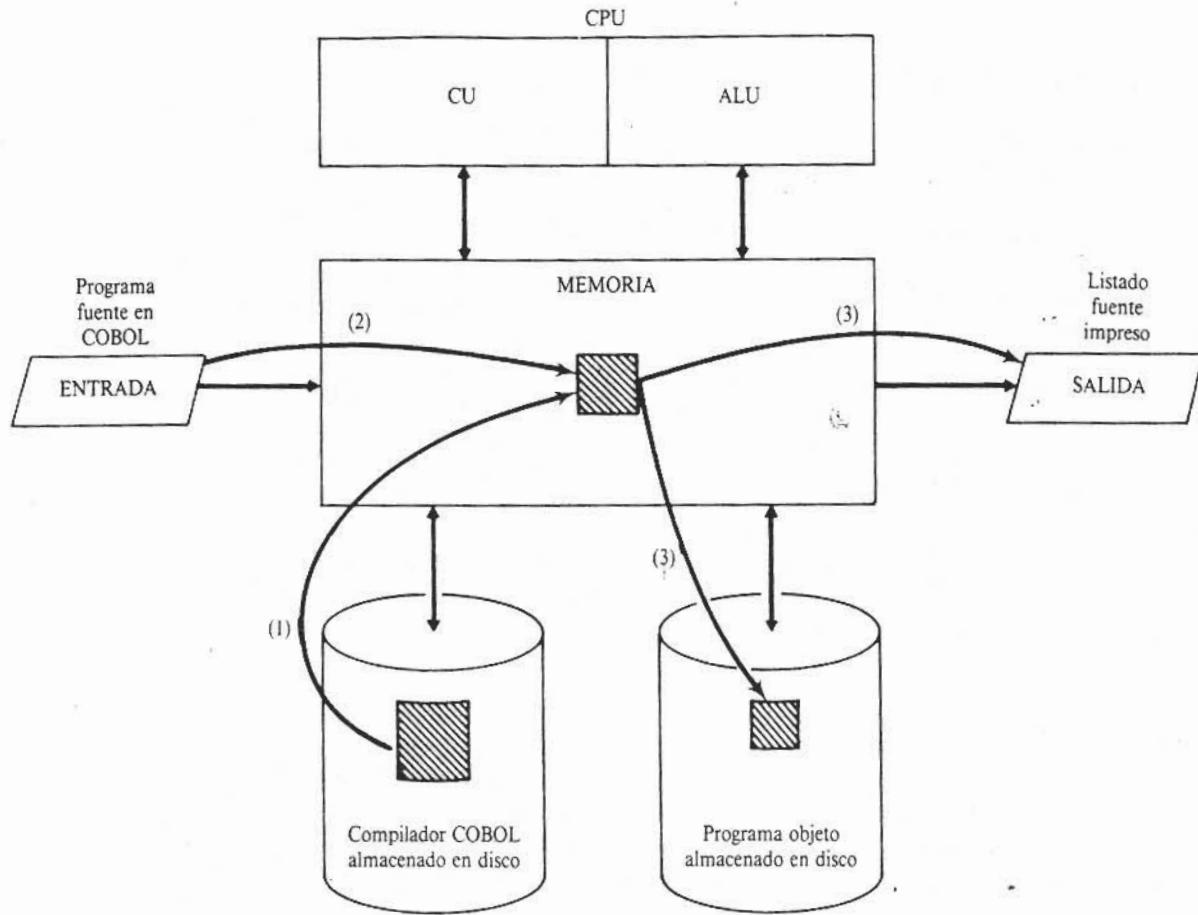
Al completar estas cuatro fases de la operación de la unidad de control, el ciclo empieza de nuevo en el punto 1. (El IAR se prepara para ello en el punto 3.)

El programador puede alterar esta *secuencia de ejecución* poniendo *instrucciones de ramificación* en su programa. Una instrucción de ramificación cambia el contenido del IAR de forma que almacene una dirección distinta de la correspondiente a la siguiente instrucción. De este modo la computadora continúa la ejecución secuencial a partir de la nueva dirección.

La velocidad de ejecución en las computadoras modernas se mide en MIPS (millones de instrucciones por segundo). Las computadoras que se comercializan en estos días operan a velocidades comprendidas entre 0,2 y 11 MIPS o incluso superiores; este ratio se incrementa continuamente.

## 1.8 LENGUAJES DE PROGRAMACION

Las instrucciones de los programas que se almacenan en memoria y se ejecutan por la CPU tienen la forma de cadenas de dígitos binarios; se dice que están expresadas en *lenguaje máquina*. Para los hombres, sin embargo, es más apropiado escribir y leer programas escritos en algún



- (1) Se transfiere el compilador desde el almacenamiento auxiliar a la memoria.
- (2) Se ejecuta el compilador, el programa fuente COBOL se lee y se transfiere.
- (3) Las instrucciones del compilador valen almacenando el programa objeto en el disco (para su posterior uso) e imprimen un listado del programa fuente.

Fig. 1-3

*lenguaje de alto nivel* como el COBOL. La traducción necesaria del programa en un lenguaje de alto nivel (*programa fuente*) a lenguaje máquina (*programa objeto*) la realiza la propia computadora bajo el control de un programa especial denominado *compilador*. Los compiladores son escritos por las compañías que producen los sistemas en general y se almacenan en formato de lenguaje máquina en el almacenamiento auxiliar.

El proceso de compilación (de un programa fuente escrito en COBOL) se esquematiza en la Figura 1-3. Si el compilador detectara errores de construcción (errores sintácticos) en el programa fuente, generaría *mensajes para el diagnóstico del error* que se imprimirían.

Si se detectan errores sintácticos, el programador debe corregirlos y volver a compilar el programa fuente modificado. Este nuevo programa fuente (o el original si no hubo errores sintácticos) tiene que *depurarse*; es decir, debe ejecutarse utilizando un conjunto de datos para los cuales los resultados de la salida se conocen de antemano. Este procedimiento se lleva a cabo para poder detectar los *errores lógicos* del programa; por ejemplo, que se instruya a la computadora a sumar, en lugar de multiplicar, las horas trabajadas y el precio por hora para obtener el valor del salario bruto. Durante la depuración, el programa fuente tendrá que recompilarse cada vez que se modifique para eliminar un error lógico.

Una vez que el programa ha sido depurado, la versión final del programa objeto se almacena permanentemente en el almacenamiento auxiliar; así puede volver a ser leído por la computadora y ejecutado cuando se necesite.

## 1.9 SISTEMAS OPERATIVOS

En lugar de encargar a *operadores* humanos la responsabilidad de *cargar* los programas objeto en la memoria para su ejecución, los fabricantes de computadoras y otros especialistas han escrito programas denominados *sistemas operativos* que permiten a la computadora realizar esta función con la rapidez que la caracteriza. Estos sistemas operativos se almacenan en disco en lenguaje máquina.

Los operadores humanos mantienen un control global del sistema de computadoras, introduciendo, para ello, información con destino al sistema operativo que le indique los programas que tiene que ejecutar. Esto se lleva a cabo de dos formas distintas. Algunos sistemas operativos están preparados para aceptar instrucciones especiales que forman parte de lo que se denomina *lenguaje de control de trabajos* (JCL), desde cualquier periférico de entrada. Las instrucciones de JCL indican al sistema operativo los *programas de aplicación* (control de inventario, cuentas de clientes, proveedores) que tiene que cargar en memoria. Otros sistemas operativos están diseñados de forma que permiten una relación *interactiva y conversacional* con los operadores humanos situados en las consolas (CRT). El usuario humano puede escribir órdenes que el sistema operativo ejecuta de inmediato; las órdenes informan al sistema el programa que se desea cargar o ejecutar. Para practicar COBOL en una computadora real es preciso conocer primero el lenguaje de órdenes o de control de trabajos para el sistema operativo con el que se esté funcionando.

## 1.10 FICHEROS

Tanto los programas como los datos se guardan en el almacenamiento auxiliar en forma de ficheros. Un *fichero* consiste en la información sobre un programa o aplicación particular y se identifica con el *nombre de fichero*. Por ejemplo, un fichero denominado ENE.FACTURAS podría tener información sobre las facturas pendientes de cobro, y otro llamado PROVEED.MAESTRO tener información sobre los pagos pendientes a proveedores. Las reglas exactas para asignar nombres a los ficheros depende del sistema operativo particular con el que se esté trabajando.

Los ficheros se componen de registros; un *registro* contiene información sobre una persona o cosa particular. Por ejemplo, en un fichero denominado EMP.DATOS los registros podrían contener información sobre los empleados, uno sobre Smith, otro sobre Jones, etc. Los registros en un fichero en cinta o disco están separados físicamente por regiones en blanco llamadas *espacios entre registros* (IRG, siglas de "Inter Register Gaps").

Las unidades de disco y cinta transfieren a la memoria de la computadora *un registro cada vez*. Por eso, cuando un programa COBOL procesa un fichero, tiene que trabajar con un registro cada

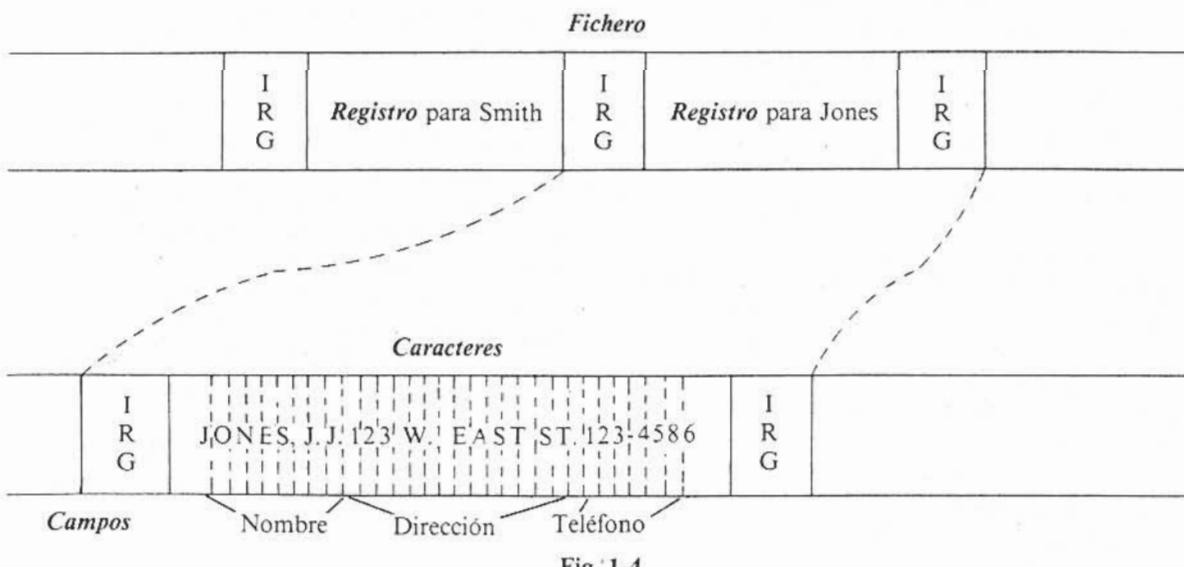


Fig. 1-4

vez. El programador está obligado a pensar en términos del ciclo entrada-proceso-salida: entrada de un registro del fichero, procesamiento del registro, salida del resultado —repetiendo el mismo ciclo hasta que todos los registros hayan sido procesados.

A su vez, un registro se compone de campos, un *campo* es un atributo como nombre, número de cuenta, salario bruto, saldo, etc. Cada campo se divide en *bytes* (o *caracteres*) que lo conforman. La jerarquía fichero/registro/campo/carácter se ilustra en la Figura 1-4.

La mayor parte de los sistemas operativos mantienen una *etiqueta de fichero* para cada fichero en disco o cinta. La etiqueta de fichero es un registro especial que contiene campos como nombre del fichero, fecha de creación, dónde se encuentra (disco), tamaño del fichero, palabra clave de seguridad del fichero, etc. Las etiquetas de fichero en cinta preceden y siguen al propio fichero en la cinta; las etiquetas de ficheros en disco se agrupan por lo general en lo que se llama *tabla de contenido del volumen (VTOC)* o *directorio*. El sistema operativo puede localizar un fichero en el disco buscando la etiqueta en el directorio. El uso de etiquetas en los ficheros permite asegurar que los programas procesan los ficheros adecuados.

## 1.11 CODIGOS DE DATOS

La unidad fundamental de memoria y de almacenamiento auxiliar es, como ya sabemos, el dígito binario o bit. Cualquier información almacenada en sistema debe *codificarse* en forma de cadena de dígitos binarios. A los efectos de su codificación, conviene clasificar los datos como sigue:

**Alfanuméricos.** datos que pueden contener cualquier carácter introducido desde un teclado, consola o impresora. En este tipo se incluyen todas las *letras* del alfabeto, los *dígitos* desde 0 a 9 y todos los *caracteres especiales* como @, #, \$, %, &, \*, (,), -, +, espacio, etc. El campo "John W. Doe" es alfanumérico (obsérvese el carácter especial ".") , lo mismo sucede con "238-9761" (obsérvese el carácter especial "-").

**Alfabéticos.** Datos compuestos por las letras del alfabeto y el carácter espacio en blanco. "John W. Doe" es alfabético, al igual que "N SPRINT ST"; sin embargo, "John W. Doe" y "43 N SPRINT ST" no son alfabéticos.

**Numéricos.** Datos compuestos por dígitos entre 0 y 9 junto con los signos + o -. En algunos casos los datos numéricos pueden contener el punto decimal ("."), pero en otras ocasiones el punto decimal se omite y se representa de otro modo. Ejemplos de datos numéricos son: "1278", "-53", "1.28" y "1.28" (donde . representa el punto decimal omitido).

Los siguientes *sistemas de codificación* se utilizan para representar datos alfanuméricos, alfabéticos y numéricos en forma de números binarios susceptibles de ser almacenados por la computadora:

**DISPLAY (EBCDIC y ASCII).** EBCDIC proviene del nombre inglés "Extended Binary-Coded Decimal Interchange" y ASCII proviene de "American Standard Code for Information Interchange". Ambos son los sistemas de codificación más usados para representar datos alfanuméricos. Los dos sistemas representan cada carácter (incluido el espacio) por medio de un número binario de 8 dígitos (por eso cada carácter requiere un byte de memoria o de almacenamiento auxiliar). Los códigos completos aparecen en el Apéndice B.

Impresoras, lectoras de tarjetas, terminales CRT y terminales con impresora intercambian información codificada en ASCII o EBCDIC con la memoria. En COBOL los datos codificados en ASCII o EBCDIC se dice que están en formato *DISPLAY*.

**COMP-3 (decimal empaquetado).** El sistema decimal empaquetado se utiliza exclusivamente con datos numéricos en memoria o en el almacenamiento auxiliar; no se utiliza por los periféricos de entrada/salida, como lectores de tarjetas, terminales o impresoras, ni tampoco todas las computadoras utilizan este sistema. En COBOL se dice que los números decimales empaquetados están en formato *COMP-3*. Este sistema de codificación tiene las siguientes ventajas sobre EBCDIC y ASCII: (i) la ALU puede calcular directamente con estos números (por el contrario tiene que convertir los números introducidos en EBCDIC o ASCII a decimal empaquetado para hacer el

cálculo y después volverlos a convertir a EBCDIC o ASCII); (ii) los números decimales empaquetados consumen menos memoria que los mismos números en EBCDIC o ASCII.

**COMP (complemento binario a dos).** El sistema complemento binario a dos tiene la misma aplicación y ventajas que el decimal empaquetado; en COBOL los números codificados en complemento binario a dos se dicen que están en formato *COMP*.

Casi todas las computadoras soportan alguna versión del sistema complemento binario a dos; algunas disponen además del sistema decimal empaquetado. Véase el Capítulo 5 para conocer en qué casos se debe usar cada sistema.

# Capítulo 2

## Generalidades de COBOL

Una de las características más importantes de COBOL es que autodocumenta, es decir, un programador puede comprender la mayor parte de los programas COBOL simplemente leyéndolos. Examine el programa COBOL del Ejemplo 2.1, al que nos referiremos repetidamente en este capítulo.

### EJEMPLO 2.1

```
00001      IDENTIFICATION DIVISION.  
00002  
00003      PROGRAM-ID. QUARTER.  
00004      AUTHOR. LARRY NEWCOMER.  
00005      INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS.  
00006      DATE-WRITTEN. MAY 1983.  
00007      DATE-COMPILED. MAY 9,1983.  
00008      SECURITY. THERE ARE NO SECURITY CONSIDERATIONS FOR QUARTER.  
00009      *  
00010      *OVERVIEW OF PROGRAM QUARTER--  
00011      *  
00012      *      QUARTER LEE UN FICHERO QUE CONTIENE LAS VENTAS MENSUALES  
00013      *      DE UN VENDEDOR DURANTE UN TRIMESTRE JUNTO CON EL NOMBRE  
00014      *      DEL VENDEDOR Y LA CUOTA TRIMESTRAL DE VENTAS. DESPUES IMPRIME  
00015      *      UN INFORME MOSTRANDO:  
00016      *  
00017      *          NOMBRE      VENTAS TRIMESTRALES      CUOTA  
00018      *  
00019      *          JONES        $42,000.98      $40,000.00  
00020      *          SMITH        $59,000.67      $60,000.00  
00021      *          YOST         $47,893.00      $45,000.00  
00022      *  
00023  
00024      ENVIRONMENT DIVISION.  
00025  
00026      CONFIGURATION SECTION.  
00027      SOURCE-COMPUTER. IBM-370.  
00028      OBJECT-COMPUTER. IBM-370.  
00029  
00030      INPUT-OUTPUT SECTION.  
00031      FILE-CONTROL.  
00032          SELECT FICHERO-VENTAS      ASSIGN TO VENTAS.  
00033          SELECT INFORME-TRIMESTRAL ASSIGN TO INFOVENT.  
00034  
00035      DATA DIVISION.  
00036  
00037      FILE SECTION.  
00038  
00039          FD FICHERO-VENTAS  
00040          LABEL RECORDS ARE STANDARD  
00041          RECORD CONTAINS 80 CHARACTERS  
00042  
00043          01 REGISTRO-VENTAS.  
00044          05 NOMBRE-REG-VENTAS      PIC X(15).  
00045          05 VENTAS-MES1-REG-VENTAS  PIC S9(4)V99.  
00046          05 VENTAS-MES2-REG-VENTAS  PIC S9(4)V99.  
00047          05 VENTAS-MES3-REG-VENTAS  PIC S9(4)V99.  
00048          05 CUOTA-REG-VENTAS     PIC S9(5)V99.  
00049          05 FILLER             PIC X(40).  
00050
```

```

00051 FD INFORME-TRIMESTRAL
00052   LABEL RECORDS ARE OMITTED
00053   RECORD CONTAINS 132 CHARACTERS.
00054
00055 01 LINEA-INFORME-TRIMESTRAL    PIC X(132).
00056
00057 WORKING-STORAGE SECTION.
00058
00059 01 INTER-Y-TOTALES.
00060   05 FIN-FICHERO-VENTAS      PIC X.
00061   05 TOTAL-TRIMESTRE      PIC S9(5)V99    COMP-3.
00062 01 LINEA-INFORME-TRABAJO.
00063   05 NOMBRE-TRABAJO        PIC X(15).
00064   05 FILLER                PIC X(5)     VALUE SPACES.
00065   05 TOTAL-TRABAJO        PIC $$,$$$$.99.
00066   05 FILLER                PIC X(5)     VALUE SPACES.
00067   05 CUOTA-TRABAJO        PIC $$,$$$$.99.
00068   05 FILLER                PIC X(87)    VALUE SPACES.
00069
00070 PROCEDURE DIVISION.
00071
00072 010-EXECUTIVE-PARA.
00073   PERFORM 020-INICIALIZACION
00074   PERFORM 040-IMPRESION-LINEAS-INFORME
00075     UNTIL FIN-FICHERO-VENTAS
00076   PERFORM 050-FIN-Y-REBOBINADO
00077   STOP RUN
00078
00079
00080 020-INICIALIZACION.
00081   OPEN INPUT FICHERO-VENTAS
00082     OUTPUT INFORME-TRIMESTRAL
00083   MOVE "F" TO FIN-FICHERO-VENTAS
00084   PERFORM 030-LEE-FICHERO-VENTAS
00085
00086
00087 030-LEE-FICHERO-VENTAS.
00088   READ FICHERO-VENTAS
00089     AT END
00090     MOVE "T" TO FIN-FICHERO-VENTAS
00091
00092
00093 040-IMPRESION-LINEAS-INFORME.
00094   MOVE NOMBRE-REG-VENTAS TO NOMBRE-TRABAJO
00095   COMPUTE TOTAL-TRIMESTRE = VENTAS-MES1-REG-VENTAS
00096     + VENTAS-MES2-REG-VENTAS
00097     + VENTAS-MES3-REG-VENTAS
00098   MOVE TOTAL-TRIMESTRE TO TOTAL-TRABAJO
00099   MOVE CUOTA-REG-VENTAS TO CUOTA-TRABAJO
00100   WRITE LINEA-INFORME-TRIMESTRAL
00101     FROM LINEA-INFORME-TRIMESTRAL
00102   PERFORM 030-LEE-FICHERO-VENTAS
00103
00104
00105 050-FIN-Y-REBOBINADO.
00106   CLOSE FICHERO-VENTAS
00107     INFORME-TRIMESTRAL
00108

```

## 2.1 LAS DIVISIONES EN COBOL

Un programa COBOL siempre consta de cuatro partes principales denominadas DIVISIONES, en el siguiente orden: identificación (IDENTIFICACION), entorno (ENVIRONMENT), datos (DATA) y procedimientos (PROCEDURE). Al principio de cada DIVISION hay una instrucción especial COBOL llamada *cabecera de división*.

**EJEMPLO 2.2** Las cabeceras de división del ejemplo 2.1 son:

- Línea 1: IDENTIFICATION DIVISION.
- Línea 24: ENVIRONMENT DIVISION.
- Línea 35: DATA DIVISION.
- Línea 70: PROCEDURE DIVISION.

La división de identificación contiene el nombre del programa, quién lo escribió, cuándo y dónde fue escrito, cuándo se compiló y las precauciones que por motivos de seguridad deben tomarse para impedir el acceso al programa o a los ficheros que el programa procesa. La DIVISION debe finalizar con un conjunto de frases en inglés dando una breve descripción de lo que hace el programa (en el Ejemplo 2.1 desde la línea 9 a la 22).

La división del entorno contiene información sobre la(s) computadora(s) en las que el programa COBOL se va a compilar y en las que el programa resultante en lenguaje máquina se ejecutará. También se da un nombre COBOL a cada uno de los ficheros que se van a procesar y asigna cada fichero a una unidad I/O específica. Esta información determina el entorno y configuración en el que se ejecutará el programa.

**EJEMPLO 2.3** En el Ejemplo 2.1:

- Línea 27: SOURCE-COMPUTER. IBM-370.  
Especifica la computadora usada para compilar el programa.
- Línea 28: OBJECT-COMPUTER. IBM-370.  
Especifica la computadora usada para la ejecución del programa objeto.
- Líneas 32-33: SELECT FICHERO-VENTAS ASSIGN TO VENTAS.  
SELECT INFORME-TRIMESTRAL ASSIGN TO INFOVENT.  
Asigna los ficheros a sus respectivas unidades I/O.

La división de datos (DATA DIVISION) da una breve descripción de cada fichero que va a procesarse y una descripción más detallada de los registros del fichero. Cada campo del registro se describe en términos de su longitud y tipo de datos. La división de datos también contiene detalles sobre los campos de datos usados por el programa que no son registros de ficheros, estos campos se almacenan en un área denominada WORKING-STORAGE. Todos los *campos de datos* (de registros o del WORKING STORAGE) tienen que describirse en la división de datos.

**EJEMPLO 2.4** Las líneas 39-49 del Ejemplo 2.1 describen el fichero de entrada de ventas y los registros que contiene:

```

FD  FICHERO-VENTAS
LABEL RECORDS ARE STANDARD
RECORD CONTAINS 80 CHARACTERS

01  REGISTRO-VENTAS.
    05 NOMBRE-REG-VENTAS      PIC X(15).
    05 VENTAS-MES1-REG-VENTAS PIC S9(4)V99.
    05 VENTAS-MES2-REG-VENTAS PIC S9(4)V99.
    05 VENTAS-MES3-REG-VENTAS PIC S9(4)V99.
    05 CUOTA-REG-VENTAS      PIC S9(5)V99.
    05 FILLER                 PIC X(40).

```

Las líneas 57-68 describen los datos del WORKING-STORAGE que no forman parte de ningún registro.

WORKING-STORAGE SECTION.

01 INTER-Y-TOTALES.	
05 FIN-FICHERO-VENTAS	PIC X.
05 TOTAL-TRIMESTRE	PIC S9(5)V99 COMP-3.

01 LINEA-INFORME-TRABAJO.	
05 NOMBRE-TRABAJO	PIC X(15).
05 FILLER	PIC X(5) VALUE SPACES.
05 TOTAL-TRABAJO	PIC \$ \$\$\$.99.
05 FILLER	PIC X(5) VALUE SPACES.
05 CUOTA-TRABAJO	PIC \$ \$\$\$.99.
05 FILLER	PIC X(87) VALUE SPACES.

Es en la división de procedimientos (PROCEDURE DIVISION) donde se instruye a la computadora respecto del proceso que tiene que llevar a cabo. Las instrucciones son semejantes a frases en inglés.

**EJEMPLO 2.5** Del Ejemplo 2.1 se han extrapolado las siguientes instrucciones de la PROCEDURE DIVISION:

- PERFORM 020-INICIALIZACION  
(línea 73) obliga a que todas las instrucciones en el párrafo COBOL denominado “020-INICIALIZACION” (líneas 80-85) se ejecuten una vez antes de que la computadora proceda a ejecutar la instrucción siguiente “PERFORM 040-IMPRESION-LINEAS-INFORME” (línea 74).
- PERFORM 040-IMPRESION-LINEAS-INFORME UNTIL FIN-FICHERO-VENTAS = “T”  
(líneas 74-75) obliga a que la computadora ejecute las instrucciones del párrafo llamado “040-IMPRESION-LINEAS-INFORME” (líneas 93-103) una y otra vez, secuencialmente, hasta que el dato denominado “FIN-FICHERO-VENTAS” contenga el valor “T”. Esto obliga a la computadora a proceder con la instrucción siguiente (línea 76).
- OPEN INPUT FICHERO-VENTAS OUTPUT INFORME-TRIMESTRAL  
(línea 81-82) la instrucción OPEN prepara para su procesamiento un fichero e indica si se va a utilizar como entrada o salida. Un fichero tiene que ser abierto (OPENed) antes de utilizarse por un programa.
- MOVE NOMBRE-REG-VENTAS TO NOMBRE-TRABAJO  
(línea 94) la instrucción MOVE copia el contenido de una variable (en este caso NOMBRE-REG-VENTAS) en otra variable (en este caso NOMBRE-TRABAJO).
- COMPUTE TOTAL-TRIMESTRE = VENTAS-MES1-REG-VENTAS  
                               + VENTAS-MES2-REG-VENTAS  
                               + VENTAS-MES3-REG-VENTAS  
(líneas 95-97) la instrucción COMPUTE se utiliza para realizar un cálculo numérico. El resultado del cálculo se almacena en la variable situada a la izquierda del signo igual.

## 2.2 SECCIONES (SECTION) Y PARRAFOS

Cada división de un programa COBOL se divide en secciones (SECTION) que comienzan con unas líneas especiales denominadas *cabecera de sección*; a su vez, cada sección se divide en *párrafos*.

**EJEMPLO 2.6** Las cabeceras de sección en el Ejemplo 2.1 son:

- línea 26: CONFIGURATION SECTION.
- línea 30: INPUT-OUTPUT SECTION.
- línea 37: FILE SECTION.
- línea 57: WORKING-STORAGE SECTION.

Las cabeceras de sección contienen el nombre de la sección seguido de la palabra “SECTION” y un punto. Los nombres de cada SECTION en las divisiones del entorno y configuración y datos son una parte fija del lenguaje COBOL y no pueden cambiarse (estas palabras se conocen como *palabras reservadas*). El uso de SECTION en la PROCEDURE DIVISION es opcional; cuando se utilizan, sus nombres deben seguir las reglas de nomenclatura de COBOL (dichos nombres se conocen como *nombres definidos por el programador*). En la división de identificación no hay secciones.

Los párrafos en COBOL pueden formar parte de una SECTION o bien figurar solos en una DIVISION. En cualquier caso, los párrafos se identifican por la *cabecera de párrafo*, que consiste simplemente en el nombre del mismo seguido de un punto. Los nombres de párrafo en las divisiones

de identificación, entorno y configuración y datos son parte fija de COBOL y no pueden cambiarse. Por el contrario, los párrafos de la división de procedimientos son opcionales y sus nombres dependen del programador.

**EJEMPLO 2.7** La división de identificación se compone únicamente de párrafos (no hay SECTION). Los nombres de párrafo de la división de identificación del Ejemplo 2.1 son:

PROGRAM-ID	AUTHOR	INSTALLATION	DATE-WRITTEN	DATE-COMPILED
SECURITY				

Estos párrafos son parte fija del lenguaje COBOL.

**EJEMPLO 2.8** La división de procedimientos de un programa COBOL puede o no estar compuesta de secciones. En el Ejemplo 2.1 sólo se utilizan párrafos en la división de procedimientos, sus nombres definidos por el programador son:

010-EXECUTIVE-PARA	020-INICIALIZACION	030-LEE-FICHERO-VENTAS
040-IMPRESION-LINEAS-INFORME	050-FIN-Y-REBOBINADO	

Aunque no es obligatorio en el lenguaje COBOL, estos párrafos empiezan con números —práctica que facilita encontrar el párrafo deseado en el listado del programa (*dando por supuesto que los números están ordenados*).

### Uso del punto

Los párrafos en la división de procedimientos se componen de frases, y como sucede en inglés, las frases se delimitan con el uso de puntos. Aunque COBOL permite el uso del punto al final de cada frase, es buena idea utilizar los puntos en la división de procedimientos sólo cuando es absolutamente necesario. Obsérvese que en la división de procedimientos del Ejemplo 2.1 los puntos ocupan ellos solos una línea. Este uso del punto (*punto estructurado*) se recomienda porque permite visualizarlo fácilmente, hace posible la inserción de nuevas frases a continuación del punto y ayuda en la depuración de errores.

**EJEMPLO 2.9** En COBOL es obligatorio el uso de puntos al final de cada párrafo. Algunas veces también son necesarios para separar una frase de otra dentro de un mismo párrafo. La división de procedimientos del Ejemplo 2.1 utiliza (i) punto después de la cabecera de DIVISION (línea 70); (ii) punto después de cada cabecera de párrafo (líneas 72, 86, 87, 93 y 105); y (iii) punto estructurado al final de cada párrafo (líneas 78, 85, 91, 103 y 108).

**EJEMPLO 2.10** Las líneas 42 y 54 del Ejemplo 2.1 contienen puntos estructurados. Estos forman parte de la división de datos que puede precisar la inserción o borrado de líneas COBOL conforme se hagan modificaciones en el programa; el punto estructurado facilita dichos cambios. Sin embargo, la regla general es que, fuera de la división de procedimientos, los puntos se colocan en la misma línea que la frase que concluyen.

## 2.3 REGLAS PARA LA ESCRITURA DE PROGRAMAS COBOL

Los programadores en COBOL escriben sus programas en unos *formularios de codificación COBOL* especiales (Fig. 2-1). Las columnas 1-6 de cada línea son por lo general utilizadas para poner unos números de secuencia que permiten ordenar fácilmente el programa. Las columnas 73-80 de cada línea se utilizan para la identificación del programa; en estas columnas se escribe una abreviatura del nombre del programa. Los números de secuencia y las columnas de identificación fueron importantes cuando los programas se perforaban en tarjetas, porque un bloque de tarjetas podía caerse y desordenarse. Hoy en día, que los programas se escriben desde terminales CRT y se almacenan en disco, son menos importantes estos dos puntos.

SISTEMA		INSTRUCCION DE PERFORACION										PAGINA DE	
PROGRAMA PROGRAMADOR		FECHA		GRAFICO					PERFORACION				
SECUENCIA (PAG)	(SERIE)	A	B	12	16	20	24	28	32	36	40	44	48
1	3	0 1											
2	4	0 2											
3	5	0 3											
4	6	0 4											
5	7	0 5											
6	8	0 6											
7	9	0 7											
8	0	0 8											
9	1	0 9											
10	2	1 0											
11	3	1 1											
12	4	1 2											
13	5	1 3											
14	6	1 4											
15	7	1 5											
16	8	1 6											
17	9	1 7											
18	0	1 8											
19	1	1 9											
20	2	2 0											
1	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54	55	56	57
58	59	60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83	84	85

Fig. 2-1

Las instrucciones COBOL se escriben entre las columnas 8 y 72. Las columnas 8-11 conforman el *margen A* y las columnas 12-72 el *margen B*. En el margen A comenzará lo siguiente:

- cabeceras de división
- cabeceras de sección
- cabeceras de párrafo

y, dentro de la división de datos,

- cabeceras de descripción de los ficheros (comenzando con las letras "FD" como en las líneas 39 y 51 del Ejemplo 2.1)
- cabeceras de descripción de los registros (datos con el número de nivel "01" como en las líneas 43, 55, 59 y 62 del Ejemplo 2.1).

Todo lo demás debe comenzar a escribirse en el margen B. Aunque COBOL permite que las entradas en el margen B comiencen en la columna 12, es fundamental en la programación estructurada que dentro del margen B se clarifiquen las relaciones entre las líneas.

**EJEMPLO 2.11** En las frases siguientes, extraídas de la división de procedimientos del Ejemplo 2.1, cada línea de la misma frase comienza un poco desplazada de la anterior:

- líneas 88-90: READ FICHERO-VENTAS  
AT END  
MOVE "T" TO FIN-FICHERO-VENTAS
- líneas 100-101: WRITE LINEA-INFORME-TRIMESTRAL  
FROM LINEA-INFORME-TRABAJO

#### Usos especiales de la columna 7

COBOL asume que cada línea sigue a la anterior dentro de una misma frase hasta que se encuentra un punto. Sólo se necesitan reglas especiales cuando una línea concluye en medio de una palabra o en medio de una *cadena entre comillas* (Sección 5.19, literales no numéricos):

- (1) Si la línea anterior concluye en medio de una palabra, debe escribirse un "-" (guión) en la columna 7 de la línea siguiente y continuar en el margen B.
- (2) Si la línea anterior concluye en medio de un literal entre comillas, debe escribirse un "-" (guión) en la columna 7 de la línea siguiente, repetir las comillas de apertura en el margen B y continuar con la cadena.

#### EJEMPLO 2.12

```

      MOVE "GRACIAS POR SU
      "PRONTO PAGO" TO
      CUSTOMER-MESSAGE
      -
      col.   col.   col.
      7       12      16
  
```

En el ejemplo de arriba se usan estas reglas, si bien se producirán muchos espacios en blanco involuntarios porque para la primera linea de la cadena entre comillas se asume que continúa hasta la columna 72.

**EJEMPLO 2.13** ¿Qué es incorrecto en lo que sigue?

```

      MOVE "OVER
      "PAYMENT" TO MENSAJE-SALDO-NEGATIVO
      -
      col.   col.   col.
      7       12      16
  
```

(a) En primer lugar podría haberse evitado partir la cadena:

```
MOVE "OVERPAYMENT"
      TO MENSAJE-SALDO-NEGATIVO
```

(b) Puesto que la primera línea se extiende hasta la columna 72, el mensaje aparecerá así:

```
"OVER
      PAYMENT"
```

que no es lo que se pretendía.

Cuando se escribe un "\*" en la columna 7, el resto de la línea se trata como un *comentario*. Esto significa que aunque el compilador imprimirá la línea en el listado del programa fuente, no traducirá su contenido al programa objeto. Los comentarios se utilizan para explicar lo que hace el programa y cómo lo hace. Son útiles para que otros programadores en COBOL puedan realizar modificaciones en un programa. Recuerde: puesto que COBOL es un lenguaje autodocumentado, los comentarios son supérfluos y sólo deben usarse donde realmente es necesario clarificar algo.

**EJEMPLO 2.14** Hay comentarios en las líneas 9-22 del Ejemplo 2.1.

La columna 7 también se utiliza para insertar *líneas de depuración* en un programa COBOL. Las líneas de depuración son simplemente líneas COBOL con una "D" escrita en la columna 7. Como su nombre indica, se utilizan para localizar y corregir errores en un programa. (Véase la Sección 4.2.)

#### Líneas en blanco

En un programa COBOL pueden aparecer líneas en blanco; se utilizan para hacer más legible el listado del programa fuente. (Líneas 2, 23, 25, 29,..., 104 del Ejemplo 2.1.)

Algunos programadores prefieren las *líneas de comentario en blanco* (líneas con un "\*" en la columna 7 únicamente), porque el compilador procesa más rápidamente una linea de comentarios que una línea en blanco. Sin embargo, la ventaja es minúscula y por tanto emplearemos líneas en blanco.

## 2.4 NOMBRES DEFINIDOS POR EL PROGRAMADOR Y PALABRAS RESERVADAS

Las palabras individuales que conforman un programa COBOL se forman de acuerdo con las siguientes reglas:

- (1) Inferiores a 30 caracteres de longitud.
- (2) Los caracteres permitidos son las letras de la A a la Z, los dígitos de 0 a 9 y el guión.
- (3) No pueden empezar ni acabar con un guión.
- (4) No pueden contener espacios dentro.

Algunas de las palabras usadas en un programa COBOL se consideran parte del lenguaje COBOL y sirven para un propósito determinado. Entre estas *palabras reservadas* están "IDENTIFICATION", "DIVISION", "DATE-WRITTEN", "INPUT-OUTPUT", "SECTION", "WORKING-STORAGE", "MOVE", "OPEN" y "COMPUTE"; hay una lista completa en el Apéndice A. Los nombres definidos por el programador se construyen de acuerdo con las reglas anteriores y *no pueden* duplicar una palabra reservada. Entre las palabras definidas por el programador están:

- nombres de fichero usados en las instrucciones SELECT y en las descripciones de fichero (FD) (véanse las líneas 32, 33, 39 y 51 del Ejemplo 2.1),
- nombres de registros y campos en la división de datos (líneas 43-49, 55, 59-68),
- nombres de párrafos y secciones en la división de procedimientos (líneas 72, 80, 87, 93, 105).

**EJEMPLO 2.15**

(a) Compare las siguientes parejas de nombres definidos por el programador:

- MVHF o MEDIA-VENTAS-HASTA-FECHA
- MP o MEDIA-PONDERADA
- EXTRA o PAGA-EXTRA-DICIEMBRE

Cuando haya posibilidad de duda es preferible ser descriptivos.

(b) Los siguientes nombres definidos por el programador son inválidos o absurdos:

- RECORD (inválido: duplica una palabra reservada)
- STUDENT-BODY-CUMULATIVE-GRADE-POINT-AVERAGE (inválido: más de 30 caracteres)
- -OUT-OF-STOCK (inválido: comienza con un guión)
- N (absurdo: no es descriptivo)
- TOTAL-\$SALES (inválido: contiene un carácter prohibido)

**2.5 COMIENZO**

Hasta el momento se ha examinado superficialmente la programación en COBOL. Sin embargo, cualquier lenguaje de programación se aprende *practicándolo*. Los dos programas COBOL que siguen sirven como guías para resolver los ejercicios de programación que se dan al final del capítulo. Para comprobar los programas que escriba deberá consultar con un instructor o programador con experiencia en el lenguaje de control de trabajos o lenguaje de órdenes que utilice su sistema de computadoras.

**EJEMPLO 2.16** Un programa COBOL completo. Todos los comentarios aparecen dentro del programa (utilizando líneas de comentario). Después del listado del programa aparece un ejemplo de la salida.

```

00001      IDENTIFICATION DIVISION.
00002
00003      PROGRAM-ID. PHONELST.
00004
00005      AUTHOR. LARRY NEWCOMER.
00006      INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS.
00007
00008      DATE-WRITTEN. MAY 1983.
00009      DATE-COMPILED. MAY 9,1983.
00011      *  PHONELST PRODUCE UN LISTADO IMPRESO DE TODOS LOS NOMBRES
00012      *  Y TELEFONOS DE EMPLEADOS. NOMBRES Y TELEFONOS ENTRAN
00013      *  DESDE UN FICHERO DE TARJETAS COLOCADAS POR ORDEN ALFABETICO.
00014      *  LA UNICA SALIDA ES UNA COPIA DE NOMBRES Y TELEFONOS, SIN
00015      *  CABECERAS Y SIN NUMEROS DE PAGINA (SE TRATA DE QUE SEA
00016      *  LO MAS SIMPLE POSIBLE)
00018      ENVIRONMENT DIVISION.
00020      CONFIGURATION SECTION.
00022      *  LA SECCION DE CONFIGURACION DESCRIBE EL SISTEMA QUE
00023      *  SE VA A UTILIZAR EN LA COMPILEACION DEL PROGRAMA
00024      *  (SOURCE-COMPUTER) Y EL SISTEMA QUE SE USARA EN LA
00025      *  EJECUCION DEL PROGRAMA OBJETO (OBJECT-COMPUTER)
00026
00027      SOURCE-COMPUTER. IBM-370.
00028      OBJECT-COMPUTER. IBM-370.

```

00030 INPUT-OUTPUT SECTION.

00032 FILE-CONTROL.

00034 \* EL PARRAFO FILE-CONTROL DE LA INPUT-OUTPUT SECTION  
 00035 \* DEFINE LOS FICHEROS QUE SE VAN A PROCESAR:

00036     SELECT FICHERO-TARJETAS                 ASSIGN TO TARJETAS.  
 00038     SELECT FICHERO-IMPRESION                 ASSIGN TO IMPRES.

00039

00040 \* ESTAS INSTRUCCIONES DAN NOMBRES COBOL A LOS FICHEROS  
 00041 \* A PROCESAR. EN ESTE CASO EL FICHERO DE ENTRADA  
 00042 \* ES FICHERO-TARJETAS Y EL DE SALIDA FICHERO-IMPRESION

00043

00044 \* LAS INSTRUCCIONES SELECT TAMBIEN DAN A CADA FICHERO  
 00045 \* UNA INSTRUCCION DEL LENG. DE CONTROL DE TRABAJOS  
 00046 \* DONDE HAY MAS INFORMACION SOBRE EL DISPOSITIVO I/O.  
 00047 \* CONSULTE A SU INSTRUCTOR SOBRE LAS INSTRUCCIONES DEL  
 00048 \* LENGUAJE DE CONTROL DE TRABAJOS NECESARIAS PARA  
 00049 \* EJECUTAR EL PROGRAMA.

00051 DATA DIVISION.

00053 \* EN LA DIVISION DE DATOS SE DESCRIBEN TODOS LOS  
 00054 \* ELEMENTOS PROCESADOS POR EL PROGRAMA.

00056 FILE SECTION.

00058 \* LA SECCION DE FICHEROS CONTIENE INFORMACION ADICIONAL  
 00059 \* SOBRE CADA FICHERO. OBSERVE QUE LOS NOMBRES DE  
 00060 \* SELECT Y FD COINCIDEN EXACTAMENTE. DEBE HABER  
 00061 \* UNA INSTRUCCION FD DE CADA FICHERO QUE APAREZCA EN  
 00062 \* UNA INSTRUCCION SELECT.

00063

00064 FD FICHERO-TARJETAS  
 00065 RECORD CONTAINS 80 CHARACTERS  
 00066 LABEL RECORDS ARE OMITTED

00067

00068 \* EN "FD" SE INDICA EL NUMERO DE CARACTERES POR  
 00069 \* REGISTRO Y SI TIENE O NO ETIQUETAS. EN UNA TARJETA  
 00070 \* NORMAL LA LONGITUD ES 80 COLUMNAS. NO SE  
 00071 \* ADMITEN ETIQUETAS EN UN FICHERO DE TARJETAS.

00072

00073 \* EL RESTO DE LA INST. "FD" DESCRIBE EL CONTENIDO DEL  
 00074 \* REGISTRO. PIC X(80) SIGNIFICA QUE CADA REGISTRO  
 00075 \* TIENE 80 CARACTERES ALFANUMERICOS.

00076

00077

00078     01 ENTRADA-NOMBRE-TELEFONO                 PIC X(80).

00080 \* LA INST. FD PARA UN FICHERO DE IMPRESION NO PUEDE  
 00081 \* TENER ETIQUETAS. LAS IMPRESORAS TIENEN LINEAS DE 132 CAR.

00082

00083 FD FICHERO-IMPRESION  
 00084 RECORD CONTAINS 132 CHARACTERS  
 00085 LABEL RECORDS ARE OMITTED

00086

00087

00088 \* UN REGISTRO DE UN FICHERO DE IMPRESION ES UNA LINEA.  
 00089 \* EN ESTE CASO CADA LINEA CONTIENE 132 CARACTERES  
 00090 \* ALFANUMERICOS ("PIC X(132)").

00091

00092     01 LINEA-IMPRESION                         PIC X(132).

00094 WORKING-STORAGE SECTION.

00096 \* LAS AREAS DEL WORKING-STORAGE NO FORMAN PARTE  
 00097 \* DE REGISTROS DE FICHEROS. AQUI SE HA DEFINIDO UN

00098 \* INTERRUPTOR CON TRES CARACTERES ("PIC X(3)") QUE  
 00099 \* CONTIENE "SI" O "NO" DEPENDIENDO DE QUE SE HAYA  
 00100 \* O NO ALCANZADO EL FINAL DEL FICHERO, ES DECIR,  
 00101 \* SI QUEDAN O NO REGISTROS POR PROCESAR.  
 00102  
 00103      01 INTER-FIN-TARJETAS                            PIC X(3).  
  
 00105 \* ESTA AREA CONTIENE UNA TARJETA DE ENTRADA CON  
 00106 \* EL NOMBRE Y TELEFONO DEL EMPLEADO -- EL NOMBRE  
 00107 \* TIENE 20 CARACTERES ALFANUMERICOS Y EL TELEFONO  
 00108 \* TIENE 8 CARACTERES ALFANUMERICOS. "FILLER" SE UTILIZA  
 00109 \* PARA RELLENAR ESPACIOS INOPERANTES.  
 00110  
 00111      01 NOMBRE-TELEFONO-TARJETA.  
 00112        05 NOMBRE-EMPLEADO                            PIC X(20).  
 00113        05 TELEFONO-EMPLEADO                        PIC X(8).  
 00114        05 FILLER                                      PIC X(52).  
  
 00116 \* EN ESTA AREA SE CONSTRUYEN LAS LINEAS DE IMPRESION  
 00117 \* OBSERVE QUE HAY AREAS INTERMEDIAS CON "FILLER"  
 00118 \* RELLENADAS CON ESPACIOS EN BLANCO MEDIANTE "VALUE  
 00119 \* SPACES". ESTOS ESPACIOS SIRVEN PARA LA SEPARACION  
 00120 \* DE LOS CAMPOS EN LA LINEA DE IMPRESION.  
 00121  
 00122      01 LINEA-NOMBRE-TELEFONO.  
 00123        05 NOMBRE-IMPRESO                            PIC X(20).  
 00124        05 FILLER                                      PIC X(5)    VALUE SPACES.  
 00125        05 TELEFONO-IMPRESO                        PIC X(8).  
 00126        05 FILLER                                      PIC X(99)    VALUE SPACES.  
  
 00128      PROCEDURE DIVISION.  
  
 00130 \*          LA DIVISION DE PROCEDIMIENTOS CONTIENE LAS  
 00131 \*          INSTRUCCIONES QUE DICEN A LA COMP. LO QUE DEBE HACER.  
 00132  
 00133      000-PROD-LISTADO-TELEFONOS.  
  
 00135      MOVE "NO" TO INTER-FIN-TARJETAS  
 00136  
 00137 \*          DA A INTER-FIN-TARJETAS EL VALOR "NO".  
 00138  
 00139 \*          LA INSTRUCCION OPEN DEBE UTILIZARSE ANTES DE QUE  
 00140 \*          SE PUEDA PROCESAR UN FICHERO. SE INDICA SI EL  
 00141 \*          FICHERO ES DE ENTRADA O SALIDA.  
 00142  
 00143      OPEN     INPUT                                    FICHERO-TARJETAS  
 00144               OUTPUT                                    FICHERO-IMPRESION  
 00145  
 00146      PERFORM 100-ENTRADA-REGISTRO  
 00147  
 00148 \*          LA INSTRUCCION PERFORM HACE QUE TODAS LAS INST.  
 00149 \*          DEL PARRAFO "100-ENTRADA-REGISTRO" SE EJECUTEN.  
 00150 \*          OBSERVE QUE ASI SE DA ENTRADA AL PRIMER REGISTRO.  
 00151 \*          DEL FICHERO. EL PARRAFO "PROD-LISTADO-  
 00152 \*          NOMBRE-TEL" PROCESA ESTE PRIMER REGISTRO.  
 00153 \*          DESPUES SE EJECUTA EL PARRAFO "100-ENTRADA-  
 00154 \*          REGISTRO" CON EL QUE SE DA ENTRADA AL  
 00155 \*          SIGUIENTE REGISTRO DEL FICHERO Y ASI  
 00156 \*          SUCESIVAMENTE.  
 00157  
 00158  
 00159  
 00160      PERFORM 200-PROD-LISTADO-NOMBRE-TEL  
 00161        UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"  
 00162  
 00163 \*          ESTA VERSION DE LA INSTRUCCION PERFORM HACE  
 00164 \*          QUE SE EJECUTE REPETITIVAMENTE EL PARRAFO  
 00165 \*          LLAMADO "200-PROD-LISTADO-NOMBRE-TEL". LA

00166 \* REPETICION DEL PARRAFO CONTINUA HASTA QUE  
 00167 \* INTER-FIN-TARJETAS CONTIENE EL VALOR "SI".  
 00168  
 00169  
 00170 CLOSE FICHERO-TARJETAS  
 00171 FICHERO-IMPRESION  
 00172  
 00173 \* LA INSTRUCCION CLOSE DEBE UTILIZARSE CUANDO EL  
 00174 \* PROGRAMA ACABA DE PROCESAR UN FICHERO.  
 00175 \* DESCONECTA EL FICHERO DEL PROGRAMA COBOL.  
 00176  
 00177 STOP RUN  
 00178  
 00179 \* LA INSTRUCCION STOP OBLIGA A LA COMPUTADORA A  
 00180 \* PARAR LA EJECUCION DEL PROGRAMA.  
 00181  
 00182  
 00184 100-ENTRADA-REGISTRO.  
 00185  
 00186 \* ESTE PARRAFO DA ENTRADA A UNA TARJETA. LA  
 00187 \* INSTRUCCION READ LEE UNA TARJETA DEL FICHERO Y  
 00188 \* MUEVE EL CONTENIDO AL AREA DEL WORKING-STORAGE  
 00189 \* LLAMADA "NOMBRE-TELEFONO-TARJETA". SI NO HUBIERA  
 00190 \* MAS TARJETAS QUE LEER, EL INTER-FIN-TARJETAS TOMA  
 00191 \* EL VALOR "SI". SE EJECUTAN A CONTINUACION LAS  
 00192 \* INSTRUCCIONES CLOSE Y STOP RUN. ESTA ULTIMA  
 00193 \* DETIENE LA EJECUCION DEL PROGRAMA.  
 00194  
 00195 READ FICHERO-TARJETAS RECORD  
 00196 INTO NOMBRE-TELEFONO-TARJETA  
 00197 AT END  
 00198 MOVE "SI" TO INTER-FIN-TARJETAS  
 00199  
 00201 200-PROD-LISTADO-NOMBRE-TEL.  
 00202  
 00203 \* ESTE PARRAFO MUEVE LOS DATOS DE NOMBRE-TELEFONO-  
 00204 \* TARJETA AL AREA DESDE LA CUAL SE IMPRIMIRA UNA  
 00205 \* LINEA DEL INFORME. LA INSTRUCCION "WRITE" IMPRIME  
 00206 \* UNA LINEA CUYOS CONTENIDOS ESTAN EN "LINEA-NOMBRE-  
 00207 \* TELEFONO". DESPUES DE ESTO LA INSTRUCCION PERFORM  
 00208 \* 100-ENTRADA-REGISTRO HACE QUE SE EJECUTE ESTE  
 00209 \* PARRAFO PONIENDO A DISPOSICION DEL PROGRAMA EL  
 00210 \* SIGUIENTE REGISTRO.  
 00211  
 00212 MOVE NOMBRE-EMPLEADO TO NOMBRE-IMPRESO  
 00213 MOVE TELEFONO-EMPLEADO TO TELEFONO-IMPRESO  
 00214  
 00215 WRITE LINEA-IMPRESION  
 00216 FROM LINEA-NOMBRE-TELEFONO  
 00217  
 00218 PERFORM 100-ENTRADA-REGISTRO  
 00219

ABEL FRED	123-4567
BAKER SUE	111-2222
CHARLIE SUE	453-7822
PERELMAN BARNEY	UNLISTED
PERELMAN MIKE	UNKNOWN
RODGERS BOB	234-5678
RODGERS BOBB	345-6789
RODGERS LEE	456-7890
FOLKERS DICK	567-8901
FOLKERS RUTH	678-9012
SUE PEGGY	231-7856
ZOTZ ZAPPA	999-9999

**EJEMPLO 2.17** Al igual que en el Ejemplo 2.16, sólo se comentan aquellas características que aparecen por primera vez.

```

00001      IDENTIFICATION DIVISION.
00002
00003      PROGRAM-ID. TIMELIST.
00004
00005      AUTHOR. LARRY NEWCOMER.
00006      INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS..
00007
00008      DATE-WRITTEN. MAY 1983.
00009      DATE-COMPILED. MAY 9,1983.
00011      *      TIMELIST IMPRIME UNA LINEA POR EMPLEADO CON:
00012      *      NUM. DE HORAS NORMALES TRABAJADAS, NUM. DE HORAS
00013      *      EXTRAS TRABAJADAS Y TOTAL HORAS TRABAJADAS. AL
00014      *      FINAL DEL INFORME SE IMPRIME EL NUM. TOTAL DE
00015      *      EMPLEADOS PROCESADOS. LA ENTRADA ES UN PAQUETE
00016      *      DE TARJETAS ORDENADAS ALFABETICAMENTE.

00018      ENVIRONMENT DIVISION.

00020      CONFIGURATION SECTION.

00022      SOURCE-COMPUTER. IBM-370.
00023      OBJECT-COMPUTER. IBM-370.

00025      INPUT-OUTPUT SECTION.

00027      FILE-CONTROL.

00029          SELECT FICHERO-TARJETAS      ASSIGN TO TARJETAS.
00030          SELECT FICHERO-IMPRESION    ASSIGN TO IMPRES.

00032      DATA DIVISION.

00034      FILE SECTION.

00036          FD  FICHERO-TARJETAS
00037              RECORD CONTAINS 80 CHARACTERS
00038              LABEL RECORDS ARE OMITTED
00039
00040          01  ENTRADA-TARJETA-HORA        PIC X(80).

00043          FD  FICHERO-IMPRESION
00044              RECORD CONTAINS 132 CHARACTERS
00045              LABEL RECORDS ARE OMITTED
00046
00047          01  LINEA-IMPRESION           PIC X(132).

00050      WORKING-STORAGE SECTION.

00052          *      OBSERVE QUE HAY VARIOS CAMPOS NUMERICOS DEFINIDOS
00053          *      EN EL WORKING-STORAGE CON "PIC 99" O BIEN
00054          *      "PIC 999". CADA 9 REPRESENTA UN DIGITO DECIMAL.
00055          *      ASI "PIC 99" DESCRIBE UN ELEMENTO QUE PUEDE
00056          *      CONTENER UN NUMERO DE 2 DIGITOS.
00057
00058          01  INTER-FIN-TARJETAS       PIC X(3).

00060          01  TARJETA-HORA-TRABAJO.
00061
00062          *      CONTIENE LA INFORMACION ENTRADA DESDE EL FICHERO TARJ.
00063
00064          05  NOMBRE-TRABAJO          PIC X(20).
00065          05  HORAS-NORMALES-TRABAJO  PIC 99.
00066          05  HORAS-EXTRA-TRABAJO   PIC 99.

```

```

00067      05 FILLER          PIC X(56).

00069      01 LINEA-LISTADO-HORAS.

00070
00071      * AREA PARA CONST. UNA LINEA DEL INFORME
00072
00073          05 NOMBRE-LISTADO      PIC X(20).
00074          05 FILLER            PIC X(5)      VALUE SPACES.
00075          05 HORAS-NORMALES-LISTADO  PIC 99.
00076          05 FILLER            PIC X(5)      VALUE SPACES.
00077          05 HORAS-EXTRA-LISTADO   PIC 99.
00078          05 FILLER            PIC X(5)      VALUE SPACES.
00079          05 TOTAL-HORAS-LISTADO  PIC 999.
00080          05 FILLER            PIC X(90)     VALUE SPACES.

00082      01 LINEA-TOTAL-EMPLEADO.

00083
00084      * AREA UTILIZADA PARA CONSTRUIR LA LINEA DEL FINAL
00085      * DEL INFORME. CONTIENE UN CONTADOR DEL NUMERO
00086      * DE EMPLEADOS.
00087
00088          05 FILLER            PIC X(25)
00089                      VALUE "NUMERO DE EMPLEADOS".
00090          05 NUM-EMPLEADOS      PIC 999.
00091          05 FILLER            PIC X(104)    VALUE SPACES.

00093      PROCEDURE DIVISION.

00095      000-PROD-INFORME-HORAS.

00096
00097      * LOS INTERRUPTORES Y CONTADORES DEBEN INICIALIZARSE AL
00098      * PRINCIPIO DEL PROGRAMA. EL INTERRUPTOR TOMA EL VALOR
00099      * "NO" Y EL CONTADOR EL VALOR CERO. SE SUMARA UNO
00100      * CADA VEZ QUE SE PROCESSE UN REGISTRO DE EMPLEADO
00101      * (VEASE EL CONTENIDO DEL PARRAFO 200-PROD.-LISTADO-HORAS).
00102
00103
00104      MOVE "NO" TO INTER-FIN-TARJETAS
00105      MOVE ZERO TO NUM-EMPLEADOS
00106
00107      OPEN   INPUT           FICHERO-TARJETAS
00108      OUTPUT          FICHERO-IMPRESION
00109
00110      PERFORM 100-ENTRADA-REGISTRO
00111
00112      PERFORM 200-PROD.-LISTADO-HORAS
00113      UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00114
00115      * DESPUES DE PROCESAR TODAS LAS TARJETAS (AL CAMBIAR
00116      * INTER-FIN-TARJETAS A "SI"), TIENE QUE
00117      * IMPRIMIRSE UNA LINEA CON EL NUMERO DE EMPLEADOS
00118      * PROCESADOS.
00119      * "WRITE LINEA-IMPRESION FROM LINEA-TOTAL-EMPLEADO"
00120      * IMPRIME EL CONTENIDO DE LINEA-TOTAL-EMPLEADO
00121
00122      WRITE LINEA-IMPRESION
00123          FROM LINEA-TOTAL-EMPLEADO
00124
00125      CLOSE   FICHERO-TARJETAS
00126          FICHERO-IMPRESION
00127      STOP RUN
00128

00130      100-ENTRADA-REGISTRO.
00131
00132      READ FICHERO-TARJETAS RECORD
00133          INTO TARJETA-HORA-TRABAJO
00134          AT END

```

```

00135      MOVE "SI" TO INTER-FIN TARJETAS
00136

00138      200-PROD-LISTADO-HORAS.
00139
00140      *      INCREMENTA CONTADOR AL PROCESAR UN EMPLEADO
00141
00142      ADD 1 TO NUM-EMPLEADOS
00143
00144      *      MUEVE DATOS DESDE LA TARJETA A DONDE SE CONST. LA LINEA
00145
00146      MOVE NOMBRE-TRABAJO          TO NOMBRE-LISTADO
00147      MOVE HORAS-NORMALES-TRABAJO   TO HORAS-NORMALES-LISTADO
00148      MOVE HORAS-EXTRA-TRABAJO     TO HORAS-EXTRA-LISTADO
00149
00150      *      CALCULAR TOTAL HORAS SUMANDO HORAS NORMALES Y
00151      *      EXTRAS. RESULTADO EN LINEA-LISTADO-HORAS.
00152
00153      ADD HORAS-NORMALES-TRABAJO HORAS-EXTRA-TRABAJO
00154      GIVING TOTAL-HORAS-LISTADO
00155
00156      WRITE LINEA-IMPRESION
00157      FROM LINEA-LISTADO-HORAS
00158
00159      PERFORM 100-ENTRADA-REGISTRO
00160

```

ABEL FRED	40	03	043
BAKER SUE	30	00	030
CHARLIE CHUCK	40	12	052
FLECKSTEINER CINDY	40	00	040
FLECKSTEINER DON	40	20	060
FLECKSTEINER STACY	15	00	015
PERELMAN BARNEY	40	15	055
PERELMAN MIKE	03	00	003
NUM. DE EMPLEADOS	008		

### Preguntas de repaso

- 2.1 ¿Por qué se dice que COBOL es un lenguaje autodocumentado?
- 2.2 ¿Cuáles son las cuatro divisiones de un programa COBOL?
- 2.3 ¿Cuál es el propósito de la división de identificación?
- 2.4 ¿Cuál es el propósito de la división del entorno y configuración?
- 2.5 ¿Cuál es el propósito de la división de datos?
- 2.6 ¿Qué es el WORKING-STORAGE?
- 2.7 ¿En qué división se especifica el procesamiento a realizar?

- 2.8 ¿Qué es una sección en COBOL?
- 2.9 ¿Qué son los párrafos en COBOL?
- 2.10 ¿Por qué se incluyen números de secuencia como parte del nombre de los párrafos en la división de procedimientos?
- 2.11 ¿Qué es un punto estructurado?
- 2.12 Explique el significado de los márgenes A y B.
- 2.13 ¿Qué debe empezarse en el margen A?
- 2.14 Explique la importancia del desplazamiento jerárquico de las líneas en el margen B.
- 2.15 Explique las reglas de continuación de líneas en COBOL.
- 2.16 Explique las reglas de escritura de programas COBOL.
- 2.17 ¿Cuál es el uso especial de la columna 7 en COBOL?
- 2.18 ¿Qué son las palabras reservadas en COBOL?
- 2.19 ¿Cuáles son las reglas para los nombres definidos por el programador en COBOL?
- 2.20 Explique la función desempeñada por las líneas en blanco en un programa COBOL.

## Programas resueltos

- 2.21 ¿Qué tipo de datos pertenecen al WORKING-STORAGE?

El WORKING-STORAGE (almacenamiento de trabajo) se utiliza para aquellos datos que no corresponden a los campos de registros de ficheros. Estos datos se calculan, por lo general, por el programa o se usan como zonas de almacenamiento temporal durante el procesamiento. Ejemplo de cosas que se asignan al WORKING-STORAGE son:

**Contadores** o posiciones de memoria utilizadas para representar un dato numérico que sirve para contar (por ejemplo, el número de empleados). Los contadores se inicializan por lo general a cero.

**Acumuladores** o posiciones de memoria utilizadas para acumular un total (por ejemplo, el saldo de una cuenta). Los acumuladores se suelen inicializar a cero.

**Banderas e interruptores de programa.** Estas posiciones de memoria contienen información que indica si ciertos sucesos han ocurrido durante la ejecución del programa. Las banderas e interruptores de programa frecuentemente admiten los valores verdadero o falso (por ejemplo, si todos los registros de un fichero han sido o no procesados). Las banderas se fijan en uno u otro valor durante la ejecución del programa.

**Líneas de información.** Con frecuencia conviene construir en el WORKING-STORAGE líneas de información para después transferirlas al área de registros del fichero de impresión antes de ser impresas con la instrucción WRITE.

- 2.22 ¿Qué hace la computadora cuando ejecuta la versión en lenguaje máquina de "PERFORM 090-CALCULO-IMPUESTOS"?

La computadora enviará el control a la primera instrucción en la versión en lenguaje máquina del párrafo denominado "090-CALCULO-IMPUESTOS". Despues procederá a ejecutar el resto de las

instrucciones del párrafo, uno después de otro, y por último se devuelve el control a la instrucción inmediatamente posterior a "PERFORM 090-CALCULO-IMPUESTOS".

- 2.23 ¿Qué hace la computadora cuando ejecuta la versión en lenguaje máquina de "MOVE EXISTENCIAS TO EXISTENCIAS-INFORME"?

En términos generales, copiará el contenido de la posición de memoria (*campo de envío*) "EXISTENCIAS" en la posición de memoria (*campo de recepción*) "EXISTENCIAS-INFORME". El valor de EXISTENCIAS permanecerá inalterado. El contenido exacto del campo de recepción dependerá de la descripción de la división de datos de los dos campos.

- 2.24 Indicar si los siguientes nombres del párrafo de la división de procedimientos son incorrectos, correctos, pero faltos de descripción o totalmente aceptables: (a) INPUT, (b) INPUT-INVENTORY-RECORD, (c) B-INPUT-INVENTORY-RECORD-040, (d) PROCESS-DATA.

- (a) Incorrecto: duplica una palabra reservada.
- (b) Correcto (se puede mejorar añadiendo un número de secuencia al nombre).
- (c) Correcto: la combinación de secuencia de letras y números, junto con el nombre que describe el párrafo, es óptima.
- (d) Correcto, pero el nombre es vago y no describe lo que el párrafo realiza.

- 2.25 Explicar los siguientes usos especiales de la columna 7:

- (a) MOVE SPACES TO LINEA-DIRECCION-NOMBRE
- 
- (b) \* EL PARRAFO SIGUIENTE CALCULA LOS IMPUESTOS ESTATALES
- 
- (c) MOVE "LA LIBRERIA DE LA UNIVERSIDAD DE ← col. 72  
"PENSILVANIA" TO CABECERA-INFORME
- 
- (d) D EXHIBIT NAMED NUM-DE-EMPLEADOS-PROCESADOS
- ↑      ↑
- col. 7 Margen B

- (a) Puesto que se divide en dos líneas se necesita "-" en la columna 7 para indicar que se continúa. Téngase en cuenta que esta situación debe evitarse en la medida de lo posible:

MOVE SPACES TO  
LINEA-DIRECCION-NOMBRE

- (b) El "\*" en la columna 7 indica que el resto de la línea es un comentario, usado para explicar lo que lleva a cabo una parte dada del programa o bien cómo lo realiza.
- (c) El "-" en la columna 7 se utiliza aquí para expresar la continuación de una cadena entre comillas (*literal no numérico*) de una línea a otra. Obsérvese que no hay cierre de comillas en la primera línea (que acaba en la columna 72) y la apertura de comillas se repite en el margen B de la segunda línea. Podría haberse evitado la partición escribiendo:

MOVE  
"LA LIBRERIA DE LA UNIVERSIDAD DEL ESTADO DE PENSILVANIA"  
TO LINEA-INFORME

- (d) La "D" en la columna 7 califica la línea como línea de depuración, usada para detectar errores en el programa. Las líneas de depuración deben eliminarse después de que el programa haya sido corregido (véase la Sección 4.2).

- 2.26 Comentar los siguientes nombres de datos definidos por el usuario: (a) TNWI, (b) TOTAL-NET-WORTH-OF-INVENTORY-FOR-JULY, (c) PHONE NUMBER, (d) #-ON-HAND, (e) ON-HAND-.

- (a) Los nombres de los datos *deben* ser descriptivos. Aunque TNWI es legal, TOTAL-NET-WORTH-OF-INVENTORY es mucho mejor.

- (b) Los nombres definidos por el usuario no pueden tener más de 30 caracteres; como consecuencia este nombre es ilegal y constituye un error sintáctico.
- (c) Los nombres definidos por el usuario no pueden contener espacios en blanco (ni cualquier otro carácter que no sea letra, dígito o guión): error sintáctico.
- (d) Carácter ilegal "#": error sintáctico.
- (e) No es posible comenzar o terminar una palabra con un guión: error sintáctico.

### Problemas complementarios

- 2.27 Entrevístese con un programador en COBOL profesional. Entérese de sus normas y organización para la escritura de programas en materia de (a) líneas de comentarios, (b) continuación, (c) nombres definidos por el programador, (d) punto estructurado, (e) desplazamiento de líneas, (f) líneas en blanco.
- 2.28 Vuelva a leer el Ejemplo 2.1 y compruebe si entiende ahora la mayor parte del programa. ¿Podría comprender el programa sin comentarios, líneas en blanco, desplazamiento de líneas, nombres descriptivos y nombres de párrafos?

### Ejercicios de programación

En todo lo que sigue debe asumir que todos los importes son enteros (véase el Capítulo 5 para el procesamiento de decimales).

- 2.29 Escriba, compruebe y depure un programa COBOL completo que imprima el siguiente informe. Diseñe sus propios datos de comprobación y verifique que la salida es correcta.
1. Una línea del informe contendrá 132 caracteres, distribuidos del siguiente modo: número de cliente, 5 caracteres alfanuméricos; 10 espacios; nombre de cliente, 30 caracteres alfanuméricos; 10 espacios; número de producto, 7 caracteres alfanuméricos; 5 espacios; cantidad pedida, 4 dígitos; 5 espacios; cantidad enviada, 4 dígitos; resto de la línea en blanco.
  2. Los datos de entrada están perforados en tarjetas como sigue: número de cliente, 5 primeras columnas; nombre de cliente, siguientes 30 columnas; número de producto, siguientes 7 columnas; cantidad pedida, siguientes 4 columnas; cantidad enviada, siguientes 4 columnas; resto de la tarjeta, sin usarse.
  3. Al final del informe, imprima una línea con la cantidad total solicitada y la cantidad total enviada (a todos los clientes).
  4. No se preocupe por los títulos, números de página, espacios entre líneas, etc.

- 2.30 Escriba, compruebe y depure un programa COBOL completo para imprimir el siguiente informe:

número de cliente:	5 espacios	saldo anterior:	5 espacios	importe del pago:	5 dígitos	saldo nuevo:
8 caracteres		5 dígitos		4 dígitos		5 dígitos

La entrada se producirá mediante tarjetas perforadas como sigue:

número de cliente	saldo anterior:	importe del pago:	63 columnas sin uso
8 caracteres	5 dígitos	4 dígitos	

El saldo nuevo se calcula como saldo anterior menos importe del pago.

Al final, imprima una línea con el importe total pagado.

- 2.31** Escriba, compruebe y depure un programa COBOL completo que imprima el siguiente informe:
- La salida consistirá en los siguientes campos espaciados adecuadamente: número de producto, 8 caracteres; existencias, 5 dígitos; pedido, 5 dígitos; cantidad disponible, 6 dígitos.
- La entrada se producirá mediante tarjetas perforadas como sigue: número de producto, primeras 8 columnas; existencias, 5 columnas; pedido, 5 columnas; resto de la tarjeta, sin uso.
- La cantidad disponible se calcula con existencias + pedido.
- Al final, imprima una línea que muestre la cantidad total en existencia (para todos los productos) y el pedido total (para todos los productos).
- 2.32** Escriba, compruebe y depure un programa COBOL completo que imprima el siguiente informe:
- Salida (espaciada adecuadamente): número de cuenta en el libro mayor, 4 caracteres; descripción, 20 caracteres; importe del abono, 5 dígitos; importe del débito, 5 dígitos.
- La entrada se producirá por medio de tarjetas perforadas ordenadas de acuerdo con el número de cuenta del libro mayor: número de cuenta en el libro mayor, 4 caracteres; descripción, 20 caracteres; importe del abono, 5 dígitos; importe del débito, 5 dígitos; resto de la tarjeta, sin uso.
- Al final, imprimir una línea que muestre el importe total de los abonos y de los débitos.
- 2.33** Escriba, compruebe y depure un programa que imprima la siguiente lista desde un fichero maestro de empleados:
- La salida consiste en los siguientes campos, con 5 espacios entre cada campo: identificativo del empleado, 6 caracteres; número de deducciones, 2 dígitos; salario anual del año anterior, 5 dígitos; salario anual del año en curso, 5 dígitos.
- La entrada se producirá mediante tarjetas perforadas como sigue: identificativo del empleado, columnas 1-6; número de deducciones, 7 y 8; salario del año anterior, columnas 9-13; salario del año en curso, columnas 14-18.
- Al final del informe, escriba una línea mostrando los salarios totales del año anterior, del año en curso y el número de empleados procesados.
- 2.34** Escriba, compruebe y depure un programa que imprima el siguiente listado desde el fichero de cuentas a pagar:
- La salida (con 8 espacios entre cada campo) será: número de factura, 4 caracteres; fecha de la factura (mm aa dd), 6 dígitos; importe de la factura, 5 dígitos; descuento, 4 dígitos; importe neto (deducido el descuento), 5 dígitos.
- La entrada se producirá mediante tarjetas perforadas como sigue: número de factura, 1-4; importe de la factura, 5-9; descuento, 10-13; fecha de la factura (mm aa dd), 14-19.
- Al final del informe, imprima una línea mostrando el total de los importes de las facturas, descuento total e importe neto total.
- 2.35** Imprima un informe con los siguientes campos del fichero de estudiantes de una universidad:
- |              |        |     |                         |
|--------------|--------|-----|-------------------------|
| Estudiante # | Nombre | Año | # Asignaturas aprobadas |
|--------------|--------|-----|-------------------------|
- La entrada se producirá mediante tarjetas perforadas como sigue: número de estudiante, 1-9; nombre, 10-29; año (1, 2, 3 ó 4), 30; número de asignaturas pasadas, 31-33.
- Al final, imprima el número total de asignaturas pasadas por todos los estudiantes.
- 2.36** Imprima etiquetas de correo con el siguiente formato:
- M. MOUSE  
1031 EDGECOMB AVE.  
YORK, PA 17403
- La entrada se producirá mediante tarjetas perforadas como sigue: nombre, 1-20; calle, 21-40; ciudad, 41-51; estado, 52-53; código postal, 54-58.
- [Nota: (1) Se necesitarán tres líneas de información (Problema 2.21). (2) Cada registro de entrada generará tres líneas de salida. (3) Use

```
WRITE IMPRIME-LINEA  
    FROM NOMBRE-LINEA-TRABAJO  
    AFTER ADVANCING 2 LINES
```

para dejar una línea en blanco después de cada etiqueta.]

- 2.37 Escriba un programa que imprima cheques con el siguiente formato:

CHEQUE #	
LIBRADO	IMPORTE (dólares)

La entrada se producirá en forma de tarjetas perforadas que contendrán el nombre del librado en las columnas 1-20 y el importe [PIC 9(5)] en las columnas 21-25. Los números de los cheques los *debe generar el propio programa*, comenzando por # 100, después 110, 120, 130, etc. Después de imprimir todos los cheques, debe imprimirse una línea con el importe total desembolsado.

- 2.38 Escriba un programa que imprima un informe bancario a partir de la siguiente entrada mediante tarjetas: nombre de cliente, 1-20; saldo en cuenta de ahorro, 21-25; saldo en cuenta corriente, 26-30; saldo en certificados de depósitos, 31-35. La salida debe consistir en una línea para cada cliente, mostrando el nombre del cliente y los diversos saldos. Al final del informe, imprima una línea que muestre el saldo total en cuentas de ahorro, corrientes y certificados de depósito. El formato del informe se deja a su elección.

# Capítulo 3

## División de identificación (IDENTIFICATION DIVISION)

### 3.1 NOTACION SINTACTICA

Esta sección no sólo se aplica a la división de identificación, sino también a todo el programa en COBOL.

La *sintaxis* es la estructura gramatical del lenguaje. Por ejemplo, la frase en español “camina hombre el calle por” tiene varios *errores sintácticos* en cuanto al orden de las palabras que la conforman. Si un programa en COBOL contuviera errores sintácticos, el compilador no podría traducir las frases inválidas al lenguaje máquina. Se imprimirían mensajes de diagnóstico del error en el listado del programa fuente, de esta manera el programador puede encontrar y corregir los errores.

Se utiliza una notación común para representar la sintaxis correcta en COBOL. Los convenios aceptados son los siguientes:

- (1) *Corchetes* [ ] encierran un grupo de elementos *opcionales*. Se puede utilizar algún elemento o ninguno.
- (2) *Llaves* {} encierran un grupo de elementos o bien uno sólo. *Necesariamente* hay que utilizar *uno* de ellos.
- (3) *Letras mayúsculas* se utilizan para las palabras reservadas y las *letras minúsculas* se utilizan para los nombres definidos por el programador.
- (4) Las palabras reservadas (en mayúsculas) subrayadas deben aparecer en la frase; las palabras reservadas sin subrayar pueden omitirse.
- (5) Los puntos suspensivos (...) se usan para identificar elementos que pueden repetirse cualquier número de veces.

**EJEMPLO 3.1** Examine la notación de la instrucción MOVE de la división de procedimientos:

MOVE  $\left\{ \begin{array}{l} \text{identificador-1} \\ \text{literal} \end{array} \right\}$  TO identificador-2 [identificador-3] ...

La palabra “MOVE” está subrayada y no encerrada entre corchetes; por tanto es obligatoria. A continuación, el programador puede elegir entre usar un identificativo *o* un literal (los únicos literales vistos hasta el momento están encerrados entre comillas). Puesto que estos dos elementos están en minúsculas, la elección entre uno u otro es a gusto del programador. Después la palabra clave “TO” es obligatoria y también el identificativo que la ha de seguir (nombre definido por el programador). De hecho a “TO” puede seguirle cualquier número de identificativos definidos por el programador.

**EJEMPLO 3.2** Vamos a ver algunos ejemplos de esta instrucción:

- MOVE ENTRADA-NOMBRE-CLIENTE TO SALIDA-NOMBRE-CLIENTE  
Correcto: forma “MOVE identificativo-1 TO identificativo-2”.
- MOVE “EXCESO DE PAGO” TO AREA-MENSAJES-FACTURA TIPO-DE-CUENTA  
Correcto: forma “MOVE literal TO identificativo-2 identificativo-3”.
- MOVE IMPORTE-DESCUENTO TO DESCUENTO-FACTURA AND DESCUENTO-ENVIO  
Incorrecto: la palabra “AND” no debe escribirse entre los identificativos DESCUENTO-FACTURA Y DESCUENTO-ENVIO.

- MOVE SALUDOS TO "HOLA"

Incorrecto: aunque "HOLA" es un literal válido, los literales no pueden seguir a "TO".

- MOVE "HOLA" VENTAS-BRUTAS TO TOTAL-VENTAS

Incorrecto: tan sólo un literal o identificativo puede aparecer entre "MOVE" y "TO".

### 3.2 SINTAXIS DE LA DIVISION DE IDENTIFICACION

La división de identificación cuya función se describió en la Sección 2.1 tiene el formato sintáctico que aparece en la Figura 3-1.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. nombre de programa.
[uAUTHOR. comentario.]
[uINSTALLATION. comentario.]
[uDATE-WRITTEN. comentario.]
[uDATE-COMPILED.]
[uSECURITY. comentario.]
```

Fig. 3-1

Obsérvese que sólo la cabecera de división y el párrafo PROGRAM-ID son obligatorios.

**EJEMPLO 3.3** Compare las líneas 1-8 del Ejemplo 2.1 con el modelo sintáctico de la Figura 3-1.

### 3.3 PARRAFO PROGRAM-ID

El párrafo PROGRAM-ID se usa para dar un nombre al programa objeto. Cuando el programa está totalmente depurado se coloca en un fichero permanente en disco (en la *biblioteca de programas*) y se le distingue por el nombre que figuraba en el párrafo PROGRAM-ID (que figura en el *directorio de la biblioteca*). Con este nombre se accede al programa para volver a ejecutarlo, sin necesidad de compilar.

El nombre que se dé en el párrafo PROGRAM-ID está sujeto a las reglas generales de los nombres definidos por el programador (Sección 2.4). Sin embargo, en el sistema IBM OS/VS y en otros sistemas, el compilador COBOL convierte automáticamente un nombre COBOL en un nombre de programa para la biblioteca. Por tanto, para evitar confusión, es mejor restringirse a las reglas de los nombres de programa en la biblioteca. Dichas reglas en la versión COBOL de IBM OS/VS son: (i) no más de 8 caracteres, (ii) letras y dígitos solamente, (iii) el primer carácter debe ser una letra.

**EJEMPLO 3.4** En los sistemas IBM:

- INVENTORY-REPORT se transformaría en INVENTOR  
(sólo se permiten 8 caracteres, por lo cual el compilador trunca el nombre).
- INVRPT no se alteraría.
- PRINT-PAYCHECKS se transformaría en PRINT0PA  
(sólo se permiten 8 caracteres; el guión no está permitido y por ello el compilador lo transforma en 0).
- I-UPDATE-RECEIVABLES se transformaría en A0UPDATE  
(sólo se permiten 8 caracteres; los guiones se transforman en 0 y además el nombre tiene que empezar con una letra, por lo cual "I" se convierte en la letra "A").
- RECVUPDT no se alteraría.

### 3.4 PARRAFOS OPCIONALES

El resto de los párrafos de la división de identificación son opcionales y las frases de cada párrafo se tratan como si fueran comentarios. Puesto que estos párrafos pueden contener la información que deseé el programador, los nombres de párrafo deben reflejar claramente de qué información se trata.

#### EJEMPLO 3.5

- INSTALLATION. INVESTIGACION Y DESARROLLO.

El párrafo INSTALLATION indica en qué centro de cálculo se desarrolló el programa (una gran empresa dispondrá probablemente de muchos sistemas de computadoras en distintos lugares).

- DATE-COMPILED.

El párrafo DATE-COMPILED (fecha de compilación) es especial, puesto que la computadora y no el programador lo rellena.

- SECURITY. AUTORIZADO SOLO AL PERSONAL-VEASE MANUAL DE POLICIA 301.

El párrafo SECURITY (Seguridad) sirve para establecer las consideraciones en esta materia para el programa y los ficheros que procesa.

A continuación de los diversos párrafos de la división de identificación, se recomienda que el programador incluya una serie de comentarios (con "\*" en la columna 7) dando una breve descripción de lo que el programa lleva a cabo. De este modo, se ahorrará tiempo y dinero cuando otro programador tenga que realizar modificaciones en el programa COBOL (*mantenimiento de programas*). El fin de la documentación de un programa es permitir que sea más fácil leerlo, entenderlo y modificarlo.

**EJEMPLO 3.6** Critique la siguiente división de identificación de un programa que va a ejecutarse en un sistema IBM OS/VS:

PROGRAM-ID. IMPRIME-ETIQUETAS-CORREO.  
DATE-WRITTEN. OCT 1983.

Desde un punto de vista técnico, lo único que está mal es que falta la cabecera de división ("IDENTIFICATION DIVISION"). Sin embargo: (1) IMPRIME-ETIQUETAS-CORREO no está en el formato que requiere la biblioteca y por tanto el compilador lo transformará en IMPRIME0. (2) Es una práctica no aconsejable que se omitan los párrafos AUTHOR, INSTALLATION, DATE-COMPILED, SECURITY y los comentarios con una breve descripción del programa. Documentar apropiadamente es fundamental en cualquier programa COBOL.

### Preguntas de repaso

- 3.1 ¿Cuál es el fin de la división de identificación?
- 3.2 Defina: (a) sintaxis, (b) errores sintácticos, (c) mensajes de diagnóstico de error.
- 3.3 Explique la notación sintáctica usada en este libro.
- 3.4 ¿Cuál es la sintaxis de la división de identificación?
- 3.5 ¿Qué partes de la división de identificación son obligatorias? ¿Qué otras partes son opcionales?
- 3.6 Explique las reglas de la escritura de la división de identificación.
- 3.7 ¿Cuál es la función de una biblioteca de programas?
- 3.8 ¿Cuáles son las reglas para el párrafo PROGRAM-ID en los sistemas IBM?
- 3.9 Explique la información que se da en cada uno de los párrafosopcionales de la división de identificación.
- 3.10 ¿Qué es el mantenimiento de programas? ¿Qué tiene que ver con la división de identificación?

### Problemas resueltos

- 3.11 Dado el siguiente modelo de la instrucción PERFORM para la división de procedimientos:

```
PERFORM { nombre-de-párrafo-1 } [ { THROUGH } { nombre-de-párrafo-2 } ]
          { THRU } { nombre-de-sección-2 }
          { identificador-1 } TIMES
          { entero-1 }
```

indicar cuáles de las frases siguientes son sintácticamente correctas:

- (a) PERFORM CALCULA-IMPORTE-DESCUENTO 10 TIMES
- (b) PERFORM LEE-REGISTRO-EMPLEADO THROUGH IMPRIME-CHEQUE NUMERO-DE-EMPLEADOS TIMES
- (c) PERFORM CHECK-DEPENDIENTE TO CALCULO-DEDUCCIONES NUMERO-DE-DEPENDIENTES TIMES
- (d) PERFORM THRU CODIGO-POSTAL 9 TIMES
- (e) PERFORM CHECK-CODIGO-DEPART THROUGH CONTADOR-CODIGO TIMES
- (f) PERFORM IMPRIME-LINEA-FACTURA 7

(a) Correcto. (b) Correcto. (c) Incorrecto: "THROUGH" o "THRU" deben usarse en lugar de "TO". (d) Incorrecto: se requiere bien nombre-de-párrafo-1 o nombre-de-sección-1 (PERFORM VALIDACION-NOMBRE THRU CODIGO-POSTAL 9 TIMES). (e) Incorrecto: necesariamente nombre de párrafo 2 o nombre de sección 2 deben seguir a "THROUGH" o "THRU" (PERFORM CHECK-CODIGO-DEPART THRU IMPRIME-TOTAL-DEPART CONTADOR-CODIGO TIMES). (f) Incorrecto: "TIMES" es obligatorio (PERFORM IMPRIME-LINEA-FACTURA 7 TIMES).

- 3.12 ¿Cuáles de los nombres siguientes son apropiados para el párrafo PROGRAM-ID?

- (a) INFORME-MENSUAL-VENTAS
- (b) IMPRIME-BALANCE-SITUACION-CUENTAS-DE-CLIENTES
- (a) Correcto: el nombre es descriptivo de lo que el programa hace. El compilador COBOL de IBM lo transformaría en "INFORME0", por lo que sería preferible usar una abreviatura de 8 caracteres (por ejemplo, "INMENVEN").
- (b) Incorrecto: el nombre usado en PROGRAM-ID debe ser un nombre definido por el usuario válido (máximo 30 caracteres).

- 3.13 Escriba la división de identificación de un programa COBOL que imprima cheques. Toda la información de los cheques está en un fichero en disco creado por el programa nóminas.

En la solución obsérvese el uso de líneas en blanco para destacar PROGRAM-ID.

IDENTIFICATION DIVISION.

PROGRAM-ID. IMPCHEQ.

AUTHOR. MIKE PERELMAN, PROGRAMADOR JUNIOR, NOMINAS  
 INSTALLATION. XYZ COMPANY, BALTIMORE, MD.  
 DATE-WRITTEN. 7 SEPTIEMBRE, 1983.  
 DATE-COMPILED.  
 SECURITY. VEASE MANUAL DE POLICIA 7A3 MANEJO DE CHEQUES  
 DE LA COMPAÑIA

- \* IMPCHEQ IMPRIME CHEQUES PARA LOS EMPLEADOS COPIANDO LA
- \* INFORMACION DE LAS NOMINAS YA CONTENIDA EN EL FICHERO
- \* DEL DISCO PAGOSEQ. IMAGE EN FORMULARIOS PARA CHEQUES.
- \* EL OPERADOR PUEDE DETENER LA IMPRESION EN CASO DE DIFICULTAD.
- \* IMPCHEQ TIENE ADEMAS LA SIGUIENTE INFORMACION DE CONTROL:
  - # CHEQUES.
- \* IMPRESOS, # CHEQUES DUPLICADOS, TOTAL IMPORTE.

### Ejercicios de programación

3.14-3.23 Modificar los Problemas 2.29-2.38 para recoger los nuevos conocimientos sobre la división de identificación.

# Capítulo 4

## **División del entorno (ENVIRONMENT DIVISION)**

### **4.1 SINTAXIS DE LA DIVISION DEL ENTORNO**

La división del entorno, cuyo propósito general se describió en la Sección 2.1, tiene la forma sintáctica que aparece en la Figura 4-1. (Los números de línea sirven tan sólo como índice de referencia.) Se aprecia que la división del entorno (ENVIRONMENT DIVISION) se compone de la sección de configuración (CONFIGURATION SECTION) y de la sección de entrada-salida (INPUT-OUTPUT SECTION). La primera sección incluye los párrafos computadora fuente (SOURCE-COMPUTER), computadora objeto (OBJECT-COMPUTER) y nombres especiales (SPECIAL-NAMES). La otra sección contiene únicamente el párrafo control de ficheros (FILE-CONTROL). Recuerde que las cabeceras de división (línea 1 en la Fig. 4-1), sección (líneas 2 y 14) y párrafo (líneas 3, 4, 6 y 15) deben escribirse comenzando en el margen A; las demás cosas deben escribirse a partir del margen B. En ocasiones todo el párrafo está en la misma línea que el nombre, mientras que en otras ocasiones esto no sucede. El cuidado aspecto visual debe servir de guía (véase el Ejemplo 4.1). Aquí sólo se comentan las entradas aplicables a los ficheros organizados secuencialmente (Sección 4.5).

- (1) ENVIRONMENT DIVISION.
- (2) CONFIGURATION SECTION.
- (3) SOURCE-COMPUTER. nombre-computadora [WITH DEBUGGING MODE].
- (4) OBJECT-COMPUTER. nombre-computadora
- (5) [PROGRAM COLLATING SEQUENCE IS nombre-alfabético].
- (6) SPECIAL-NAMES.
- (7) [nombre-función IS nombre-cobol] ...
- (8) nombre-alfabético IS
- (9)     STANDARD-1
- (10)     NATIVE
- (11)     literal-1 [ { THROUGH } literal-2 ] ... } }
- (12)     [ { THRU } literal-3 [ ALSO literal-4 ] ] ... } }
- (13) .]
- (14) [INPUT-OUTPUT SECTION.
- (15) FILE-CONTROL.
- (16)     SELLECT nombre-fichero
- (17)         ASSIGN TO nombre-externo
- (18)         [ORGANIZATIONS IS SEQUENTIAL]
- (19)         [ACCESS MODE IS SEQUENTIAL]
- (20)         [FILE STATUS IS nombre-datos]
- (21)         [RESERVE entero [ AREAS ] ] AREA ] }
- (22) .]

Fig. 4-1

**EJEMPLO 4.1**

```

00031      ENVIRONMENT DIVISION.
00032
00033      CONFIGURATION SECTION.
00034
00035      SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.
00036      OBJECT-COMPUTER. IBM-370
00037          PROGRAM COLLATING SEQUENCE IS
00038              CUSTOMER-NUMBER-CODE-SEQUENCE.
00039      SPECIAL-NAMES.
00040
00041          C01 IS PARTE-SUP-PAGINA
00042          CUSTOMER-NUMBER-CODE-SEQUENCE IS
00043              "3" THRU "9"
00044              "2"
00045              "0" THRU "1"
00046
00047
00048      INPUT-OUTPUT SECTION.
00049
00050      FILE-CONTROL.
00051
00052          SELECT FICHERO-VENTAS-CLIENTES
00053              ASSIGN TO VENCLien
00054              ORGANIZATION IS SEQUENTIAL
00055              ACCESS IS SEQUENTIAL
00056              FILE STATUS IS ESTADO-VENTAS
00057
00058
00059          SELECT INFORME-VENTAS
00060              ASSIGN TO INFOVENT
00061              ORGANIZATION IS SEQUENTIAL
00062              ACCESS IS SEQUENTIAL
00063              FILE STATUS IS ESTADO-INFORME
00064

```

**4.2 CONFIGURATION SECTION: PARRAFO SOURCE COMPUTER**

La CONFIGURATION SECTION describe la computadora usada para compilar y ejecutar el programa. El párrafo SOURCE-COMPUTER empieza con el “nombre de la computadora” utilizada para la compilación. Esta frase se trata como si fuera un comentario por el compilador y sirve únicamente como documentación del programa. La forma de introducir el “nombre de la computadora” varía de acuerdo con el sistema; para los sistemas IBM-370, sería suficiente “IBM-370”.

El párrafo SOURCE-COMPUTER admite la frase *opcional* “WITH DEBUGGING MODE” (línea 3, Fig. 4-1). Con esta frase, todas las líneas de depuración (marcadas con una “D” en la columna 7) se traducen al programa objeto y por tanto se tienen en cuenta durante la ejecución. Si no se incluye esta frase, las líneas de depuración se tratan como comentarios (aunque se imprimen en el listado del programa fuente, *no* se traducen al programa objeto y por tanto *no tienen efecto* durante la ejecución). La depuración de programas se trata extensamente en el Capítulo 9.

**EJEMPLO 4.2**

- SOURCE-COMPUTER. IBM-370-158.

El número de modelo de la computadora “158” se puede añadir opcionalmente a “IBM-370”. Al no haber especificado “WITH DEBUGGING MODE”, las frases marcadas con “D” en la columna 7 se tratarán como comentarios y por tanto no se ejecutarán. El nombre de la computadora se trata como un comentario.

- SOURCE-COMPUTER. IBM-370  
WITH DEBUGGING MCDE.

Las líneas de depuración pasarán a formar parte del programa objeto (como cualquier otra instrucción) y se ejecutarán. Se escribe esta frase en una línea aparte para aumentar la apariencia visual.

### 4.3 CONFIGURATION SECTION: PARRAFO OBJECT-COMPUTER

El párrafo OBJECT-COMPUTER indica la computadora usada para ejecutar el programa objeto. El nombre de la computadora sirve sólo como documentación del programa; por lo general es el mismo que se introduce en el párrafo SOURCE-COMPUTER.

Recuerde que la computadora almacena información registrando cada carácter en forma de número binario. Al considerar el orden de los números correspondientes a cada carácter, se obtiene un orden para los caracteres propiamente dichos que se conoce con el nombre de *secuencia de orden de caracteres*. Esta secuencia depende, por supuesto, del sistema de codificación utilizado; las secuencias correspondientes a los sistemas EBCDIC y ASCII aparecen en el Apéndice B.

**EJEMPLO 4.3** En la secuencia de orden de EBCDIC,

espacio < \$ < # < A < B < Z < 0 < 9

El orden de estos mismos caracteres en ASCII es:

espacio < # < \$ < 0 < 9 < A < B < Z

Ambos sistemas, EBCDIC y ASCII, respetan en sus secuencias el orden alfabético; "ZAG" < "ZIG".

El párrafo OBJECT-COMPUTER tiene una entrada opcional (línea 5, Fig. 4-1) que permite al programador elegir la secuencia de orden de caracteres que deberá usarse en la ejecución del programa. Si se omite esta entrada, se asumirá un sistema por defecto (como por lo general se desea la secuencia que se asume por defecto, no suele hacerse uso de esta facilidad). La secuencia de orden determina el resultado que se obtendrá al comparar o clasificar datos alfanuméricos.

**EJEMPLO 4.4** Examinar el Ejemplo 4.1. En el párrafo OBJECT-COMPUTER (líneas 36-38), "IBM-370" se trata como un comentario y "CUSTOMER-NUMBER-CODE-SEQUENCE" es la secuencia de orden que utilizará el programa. Esta secuencia se *define* en las líneas 42-45 del párrafo SPECIAL-NAMES.

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

### 4.4 CONFIGURATION SECTION: PARRAFO SPECIAL-NAMES

Este párrafo tiene tres funciones principales, que se estudiarán individualmente.

#### Nomenclatura de canales de control de carro

Cuando se imprimen datos en *papel continuo*, las posiciones de los diversos elementos en el formulario se definen por medio de los *canales de control del carro*. En la Figura 4-2 estos canales se indican mediante agujeros en una cinta de papel que acompaña al propio formulario; más comúnmente, en las impresoras modernas los canales se controlan electrónicamente. En cualquier caso, la línea 7 de la Figura 4-1 se utiliza para asignar un nombre COBOL a un canal de control del carro para la impresora. En la versión de COBOL IBM OS/VS el "nombre de función" de un canal consiste en la letra "C" seguida de un número de canal con dos dígitos: C01, C02, C03,..., C09. El "nombre-cobol" de un canal tiene que ser un nombre definido por el programador.

#### EJEMPLO 4.5

```
C01 IS PARTE-SUPERIOR-FACTURA
C02 IS FECHA-FACTURA
C03 IS DIRECCION-CLIENTE
C04 IS LINEA-ELEMENTO
```

En este caso, los canales 1-4 se asocian con nombres definidos por el usuario que se usarán en la división de procedimiento para controlar el formato de la salida por impresora. Obsérvese que sólo hay *un punto* para todo el párrafo SPECIAL-NAMES (línea 13 de la Fig. 4-1) —no hay puntos entre las distintas entradas. Asimismo, obsérvese el carácter descriptivo de los nombres definidos por el programador.

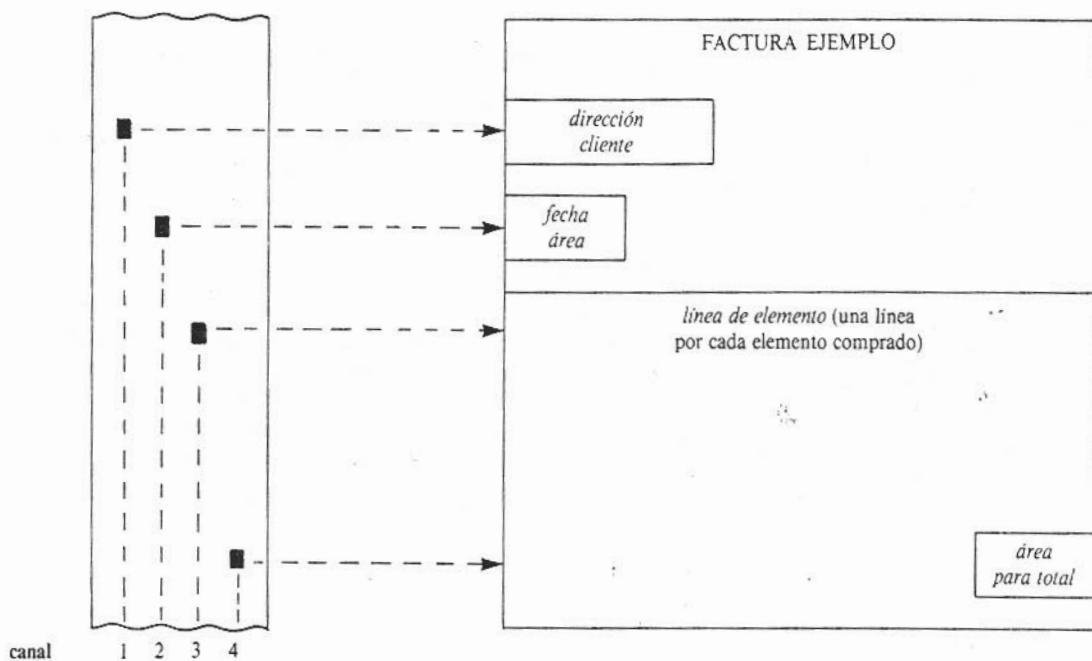


Fig. 4-2

#### Definición de la(s) secuencia(s) de orden

Si se especifica una secuencia de orden en el párrafo OBJECT-COMPUTER, debe definirse en el párrafo SPECIAL-NAMES (líneas 8-12 de la Fig. 4-1). El nombre del alfabeto (línea 8) del párrafo SPECIAL-NAMES debe coincidir con el que aparece en el párrafo OBJECT-COMPUTER (línea 5). La secuencia de orden asociada con el nombre del alfabeto se define como una de las siguientes:

- STANDARD-1  
(línea 9, Fig. 4-1), secuencia de orden ASCII
- NATIVE  
(línea 10, Fig. 4-1), secuencia de orden EBCDIC (sistema por defecto en las computadoras IBM-370)
- definido por el programador  
(líneas 11-12, Fig. 4-1). El programador puede crear su propia secuencia de orden escribiendo para ello los caracteres en el orden deseado. Para abreviar se pueden especificar rangos de caracteres ("literal-1 THROUGH literal-2"), la paridad entre dos caracteres se indica con la opción "ALSO".

Nuevamente, remarcaremos que con independencia de los canales de control del carro y la secuencia de orden que se especifique en el párrafo SPECIAL-NAMES, sólo hay un punto al final del párrafo total.

#### EJEMPLO 4.6

.....

OBJECT-COMPUTER. IBM-370 COLLATING SEQUENCE IS CODIGO-ORDEN-RAPIDO.

C01 IS PARTE-SUP-FORMULARIO  
C02 IS AREA-DOMICILIO-CORREO

CODIGO-ORDEN-RAPIDO IS

"I" THRU "N"  
"D" "G" "F" "E"  
"A" THRU "C"  
"H" ALSO "O"

Aquí, en el párrafo SPECIAL-NAMES, se define la secuencia de orden CODIGO-ORDEN-RAPIDO como sigue:

I < J < K < L < M < N < D < G < F < E < A < B < C < H=O

“H” y “O” se consideran equivalentes en esta secuencia de orden. Los caracteres no especificados se supone que siguen a “H” y “O” en el orden normal por defecto, es decir, vendrían los caracteres especiales, letras minúsculas, después “P”, etc.

#### **La consola del operador y las palabras SYSIN y SYSOUT**

En el párrafo SPECIAL-NAMES se puede opcionalmente asignar (línea 7, Fig. 4-1) nombres COBOL a ciertos ficheros del sistema procesados en la división de procedimientos por las instrucciones ACCEPT y DISPLAY (véase el Capítulo 6). Los detalles exactos de este mecanismo varían de un sistema operativo a otro, pero en el sistema IBM OS/VS COBOL “ACCEPT” permite la entrada de un registro desde la *consola del operador* (pantalla CRT o terminal de impresión utilizado por el operador de la computadora para comunicarse con el sistema operativo) o desde el fichero SYSIN. La instrucción “DISPLAY” permite la salida de un registro hacia la consola del operador o hacia el fichero SYSOUT.

#### **EJEMPLO 4.7**

```
SPECIAL-NAMES.  
C01 IS PARTE-SUP-FACTURA  
CONSOLE IS DISPOSITIVO-MENS-OPERADOR.
```

El nombre-de-función “CONSOLE” es una palabra reservada de IBM. “DISPOSITIVO-MENS-OPERADOR” es el nombre mediante el cual el resto del programa fuente en COBOL se referirá a la consola del operador en las instrucciones ACCEPT y DISPLAY:

```
DISPLAY...UPON DISPOSITIVO-MENS-OPERADOR  
ACCEPT...FROM DISPOSITIVO-MENS-OPERADOR
```

#### **EJEMPLO 4.8**

```
SPECIAL-NAMES.  
C01 IS AREA-CABECERA  
C02 IS AREA-TOTAL  
SYSIN IS FUNCION-SEL-ENTRADA  
BACK-ORDER-CODING-SEQUENCE IS STANDARD-1
```

“FUNCION-SEL-ENTRADA” es el nombre elegido en COBOL para la entrada en el fichero especial SYSIN (palabra reservada IBM). “FUNCION-SEL-ENTRADA” se utilizará en las instrucciones ACCEPT de la división de procedimientos:

```
ACCEPT...FROM FUNCION-SEL-ENTRADA
```

#### **EJEMPLO 4.9**

```
SPECIAL-NAMES.  
SYSOUT IS ERROR-LOG.
```

La salida de la instrucción DISPLAY en la división de procedimiento irá al fichero de salida especial del sistema SYSOUT (palabra reservada del COBOL de IBM). El nombre COBOL para SYSOUT será “ERROR-LOG”; por ejemplo:

```
DISPLAY...UPON ERROR-LOG
```

### **4.5 SECCION DE ENTRADA-SALIDA (INPUT-OUTPUT SECTION)**

Todo el procesamiento de datos en COBOL se basa en la manipulación de *registros* en ficheros. La información de la entrada (con independencia del dispositivo que se utilice) se analiza como si se compusiera de *registros de entrada* procedentes de un *fichero de entrada*. La información de salida (con independencia del dispositivo que se utilice) se contempla como un conjunto discreto de

*registros de salida* que se envían a un *fichero de salida*. A causa de la posibilidad de acceso a cualquier registro en cuestión de milisegundos (Sección 1.4), las unidades de almacenamiento auxiliar en disco (generalmente llamadas *periféricos de acceso directo*) pueden soportar ficheros cuyos registros son de entrada y salida en el mismo programa COBOL (*ficheros de entrada-salida*). Un registro con posibilidad de acceso directo puede primero ser tratado como entrada, después modificarse y por último devolverse como salida a su primitiva posición en el disco. Esta operación se conocé como *actualización en su lugar*.

**EJEMPLO 4.10** Un programa COBOL que imprima los cheques con el salario de los empleados procesará por lo menos 3 ficheros:

- (1) Un fichero de entrada cuyos registros (tarjetas) contengan como campos el identificativo del empleado y las horas semanales trabajadas.
- (2) La información de los cheques se destinará a la impresora para que sea impresa en formularios preparados al efecto. Cada línea impresa se considera un registro del fichero de salida.
- (3) El salario-hora de cada empleado podría obtenerse de un fichero maestro de empleados. Conforme se da entrada a la tarjeta de un empleado, el registro correspondiente en el fichero maestro se traerá desde el disco. Probablemente, algunos campos del fichero maestro de empleados se actualicen durante el proceso; por ejemplo, salario anual bruto hasta la fecha se verá incrementado adecuadamente. El registro de empleado una vez *actualizado* se devolverá a su posición primitiva en el disco (borrando el contenido anterior del registro). De este modo, el fichero en disco sirve a la vez para entrada y salida en el programa. Esta situación es común al trabajar con ficheros contenidos en dispositivos de almacenamiento auxiliar.

La sección de entrada-salida contiene un único párrafo COBOL, el párrafo FILE-CONTROL, que contiene una instrucción SELECT por cada fichero (de entrada, de salida o de entrada-salida) procesado por el programa. Cada instrucción SELECT se compone de varias cláusulas y finaliza con un punto (véanse las líneas 16-22, Fig. 4-1).

#### SELECT nombre-fichero (línea 16)

El nombre del fichero es un nombre COBOL definido por el programador. Debe ser tan descriptivo como sea posible; por ejemplo, "FICHERO-MAESTRO-EMPLEADOS" es más apropiado que "FICHERO-1". Este nombre se usará para referirse al fichero en todo el resto del programa.

#### EJEMPLO 4.11

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT FICHERO-VENTAS-CLIENTES  
    SELECT INFORME-VENTAS...
```

(extraído del Ejemplo 4.1). Este programa procesaría exactamente dos ficheros (puesto que sólo hay dos instrucciones SELECT), que aparecerían en el resto del programa bajo los nombres FICHERO-VENTAS-CLIENTES e INFORME-VENTAS.

#### ASSIGN TO nombre-externo (línea 17)

La cláusula ASSIGN TO asigna ficheros a periféricos de entrada/salida (I/O). A cada fichero se le da un "nombre externo" que asocia el fichero COBOL con las *instrucciones del lenguaje de control de trabajos* o las *órdenes del sistema operativo* (véanse la Sección 1.9 y el Problema 4.45). El contenido y formato de las instrucciones del lenguaje de control de trabajos varían mucho de una computadora o sistema operativo a otro; *debe aprender el modo de escribir nombres externos en su sistema de computadoras*. En la versión de COBOL para IBM OS/VS el nombre externo tiene que tener de 1 a 8 letras o dígitos, siendo el primer carácter una letra. El programador en COBOL debe procurar conocer cuanto más mejor sobre el lenguaje de control de trabajos.

**EJEMPLO 4.12**

```

INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT FICHERO-VENTAS-CLIENTES
    ASSIGN TO VENCLien...
  SELECT INFORME-VENTAS
    ASSIGN TO INFOVENT...

```

(extraído del Ejemplo 4.1). En este caso, al fichero denominado FICHERO-VENTAS-CLIENTES se le asigna el nombre externo VENCLien y al fichero INFORME-VENTAS se le asigna el nombre INFOVENT. Estos nombres externos se utilizarán en las instrucciones del lenguaje de control de trabajos o en las órdenes del sistema operativo, las cuales proporcionarán información adicional sobre los dispositivos I/O necesarios para procesar estos ficheros.

**Organización y modo de acceso secuenciales (líneas 18, 19; Opcional)**

El lenguaje COBOL gira en torno al procesamiento de ficheros *registro a registro*. Sin embargo, se admite flexibilidad respecto de qué registros se procesan primero, segundo, etc. Hay dos formas fundamentales de *procesamiento de ficheros o modos de acceso*.

**Acceso secuencial.** Los registros se procesan de acuerdo con su orden físico en el dispositivo de entrada-salida. No se precisa más información del fichero que la que contienen los registros que lo componen. Este tipo de procesamiento es posible con cualquier clase de dispositivo de entrada-salida.

**Acceso al azar.** Los registros se procesan en el orden previsto en el programa, que puede ser diferente del de su ubicación física en el dispositivo correspondiente. Este tipo de procesamiento sólo es posible con periféricos de acceso directo. Puede suceder que se necesite información adicional sobre el fichero para ayudar a localizar los registros de datos que requiera el programa.

*Organización de ficheros* es un término usado en relación con el método usado para almacenar registros (y cualquier información adicional precisa para la localización de los registros) en un fichero. El COBOL estándar ofrece al programador tres posibilidades:

**Organización secuencial.** Los ficheros organizados secuencialmente se pueden almacenar en cualquier dispositivo I/O. El fichero se compone sólo de los registros de datos propiamente dichos, que tienen que ser procesados en el orden de su colocación en el dispositivo I/O.

**Organización indexada.** Los ficheros indexados tienen que ubicarse en dispositivos de acceso directo. Se pueden procesar secuencialmente o al azar. Cuando se procesan secuencialmente, los registros se van capturando en orden creciente de una *clave* predefinida (campo cuyo contenido identifica unitariamente cada registro del fichero; por ejemplo, el número de cuenta). Cuando se procesan al azar, se identifica a los registros haciendo uso de la misma clave. El fichero se compone no sólo de los registros de datos propiamente dichos, sino también de *registros índice* que contienen información sobre la ubicación de cada registro en el fichero. Si se conoce el campo clave de un registro, se puede localizar rápidamente el registro buscando para ello en los registros índice del fichero. De este modo se hace posible el procesamiento al azar.

**Organización relativa.** Los ficheros relativos no se usan con la misma frecuencia que los ficheros secuenciales o indexados. El fichero se compone sólo de los registros de datos numerados 1, 2, 3,..., y por medio de este *número de registro* el proceso se puede procesar al azar. Dado que en los negocios se acostumbra a identificar la información contenida en los registros por su *clave* y no por el *número de registro*, los ficheros relativos se utilizan sólo cuando se necesita un acceso extremadamente rápido. Los ficheros relativos deben almacenarse en dispositivos de acceso directo.

La instrucción SELECT proporciona un medio para que el programador en COBOL pueda elegir la organización del fichero y el modo de acceso en cada fichero manejado por el programa. Esta elección es fundamental. Por lo general, los ficheros que requieren un procesamiento al azar se crean con organización indexada; los ficheros que sólo requieren procesamiento secuencial se crean con organización secuencial y los ficheros que pueden precisar procesamiento al azar o secuencial se crean con organización indexada (adecuada para ambos modos). El procesamiento secuencial de ficheros se estudia en el Capítulo 11; los ficheros indexados y relativos caen fuera del ámbito de este libro.

#### EJEMPLO 4.13

```

INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT FICHERO-VENTAS-CLIENTES
    ASSIGN TO VENCLien
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL...
  SELECT INFORME-VENTAS
    ASSIGN TO INFOVENT
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL...

```

(extraido del Ejemplo 4.1). Hay instrucciones SELECT para dos ficheros, FICHERO-VENTAS-CLIENTES (nombre externo, VENCLien) e INFORME-VENTAS (nombre externo, INFOVENT).

- ORGANIZATION IS SEQUENTIAL

Esta entrada se podría haber omitido porque la organización secuencial se asume por defecto. Otras posibilidades serían ORGANIZATION IS INDEXED y ORGANIZATION IS RELATIVE.

- ACCESS MODE IS SEQUENTIAL

Esta entrada también podría haberse omitido porque el modo de acceso secuencial se asume por defecto. Las otras posibilidades hubieran sido ACCESS MODE IS RANDOM y ACCESS MODE IS DYNAMIC. Los ficheros organizados secuencialmente sólo permiten el acceso secuencial; otros modos de acceso exigen que la organización sea indexada o relativa.

#### FILE STATUS IS nombre-de-datos (línea 20; Opcional)

Conceptualmente, la manipulación de los registros de un fichero que se hace en la división de procedimientos en COBOL es muy simple. La instrucción READ trae un registro a la CPU para su procesamiento desde el dispositivo de entrada o desde el almacenamiento auxiliar; WRITE transfiere un registro desde la memoria de la CPU al dispositivo de salida o al almacenamiento auxiliar. Antes de que puedan usarse las instrucciones READ o WRITE con los registros de un fichero, tiene que utilizarse la instrucción "OPEN..." para preparar (abrir) el fichero para su procesamiento (véase el Ejemplo 2.5). De forma análoga, tiene que usarse la instrucción "CLOSE..." para desactivar (cerrar) el fichero después de haber ejecutado todas las instrucciones READ y WRITE.

En la vida real, sin embargo, es fácil que se cometan equivocaciones. Por ejemplo, una instrucción OPEN puede fallar porque la descripción del fichero en el programa fuente en COBOL es inconsistente con la descripción en las instrucciones correspondientes en el lenguaje de control de trabajo. O bien una instrucción WRITE para añadir un registro a un fichero organizado secuencialmente en disco puede fallar porque no quede espacio vacío en el disco. De hecho, en los programas en COBOL se prevé que ciertas instrucciones puedan fallar. Así, para procesar los registros de un fichero secuencial, se diseñará un programa que permanezca leyendo (READ) registros, procesándolos y sacando los resultados hasta que la instrucción READ falle porque todos los registros hayan sido leídos (*condición de fin de fichero*).

Por tanto, los programas deben estar preparados para manejar las *condiciones excepcionales* que puedan tener lugar durante el procesamiento del fichero. La instrucción SELECT proporciona un medio para obtener *respuesta* sobre el éxito o fallo de cada instrucción que manipula un fichero.

Esta respuesta se produce en forma de un código que se sitúa en un área de datos después de cada instrucción OPEN, CLOSE, READ y WRITE. El código indica si la ejecución de la instrucción tuvo o no éxito, y si no lo tuvo, las razones del fallo.

#### EJEMPLO 4.14

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
  SELECT VENTAS-MENSUALES  
    ASSIGN TO MENSVEN  
    FILE STATUS IS ESTADO-VENTAS-MENSUALES
```

En la cláusula FILE STATUS, el programador ha elegido el nombre "ESTADO-VENTAS-MENSUALES" para el área de datos que se usará para obtener respuesta con relación al resultado de las instrucciones OPEN, READ, WRITE y CLOSE ejecutadas en la división de procedimientos. "ESTADO-VENTAS-MENSUALES" debe además definirse en la división de datos como un área que puede contener dos caracteres cuyo significado será el éxito o fracaso de cualquier operación realizada con el fichero. Consulte su manual para conocer el significado de FILE STATUS en su sistema.

Muchos programadores en COBOL con experiencia no definen área para el FILE STATUS en los ficheros secuenciales, puesto que hay otros modos de comprobar el resultado (éxito o fracaso) de las operaciones de entrada/salida (véase el Capítulo 11).

#### RESERVE entero AREAS (línea 21; Opcional)

Esta cláusula se puede incluir para aumentar la velocidad de procedimiento cuando el acceso es secuencial (véase la Sección 5.8).

### 4.6 EJEMPLOS DE LA DIVISION DEL ENTORNO

#### EJEMPLO 4.15

```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-370.  
OBJECT-COMPUTER. IBM-370.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
  SELECT VIEJO-MAESTRO-INVENTARIO  
    ASSIGN TO VIEJOMAE.  
  SELECT NUEVO-MAESTRO-INVENTARIO  
    ASSIGN TO NUEVOMAE.  
  SELECT FICHERO-TRANSACCIONES  
    ASSIGN TO TRANSACC.
```

Esta es la división del entorno de un programa para procesar un fichero secuencial. Los registros del FICHERO-TRANSACCIONES se usarán para actualizar la información de los registros de VIEJO-MAESTRO-INVENTARIO. Los registros contenido la información actualizada se ubicarán en el fichero NUEVO-MAESTRO-INVENTARIO. Los tres ficheros son secuenciales (asunción por defecto cuando se omite la cláusula ORGANIZATION IS...) y se procesan secuencialmente (asunción por defecto cuando se omite la cláusula ACCESS MODE IS...). Puesto que todos los ficheros son secuenciales, las áreas determinadas en la cláusula FILE STATUS son opcionales y aquí no se utilizan (las condiciones excepcionales se comprueban durante los procesos de entrada/salida con métodos de la división de procedimientos). La secuencia de orden de caracteres es la que se asume por defecto en el sistema de que se trate (EBCDIC en los sistemas IBM-370). Como no se especifica DEBUGGING MODE, las líneas con una "D" en la columna 7 se tratarán como comentarios y no tendrán efectos en la ejecución del programa. Los nombres de las cláusulas ASSIGN TO tienen el formato apropiado para los sistemas IBM OS/VS ("VIEJOMAE", "NUEVOMAE" y "TRANSACC" contienen de 1 a 8 caracteres, letras o dígitos, el primero de los cuales es una letra).

**EJEMPLO 4.16**

```

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370
    WITH DEBUGGING MODE.
OBJECT-COMPUTER. IBM-370.
SPECIAL-NAMES.
    C01 IS PARTE-SUP-CHEQUE
    C02 IS AREA LIBRADO
    C05 IS AREA-IMPORTE

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FICHERO-NOMINAS
        ASSIGN TO NOMINAS
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL
        FILE STATUS IS ESTADO-NOMINAS

    SELECT FICHERO-CHEQUES
        ASSIGN TO CHEQUES
        ORGANIZATION IS SEQUENTIAL
        ACCESS MODE IS SEQUENTIAL
        FILE STATUS IS ESTADO-CHEQUES

```

## Comentarios:

- (1) DEBUGGING MODE está “activado”; por tanto, las líneas con “D” en la columna 7 se traducirán y tendrán efecto durante la ejecución.
- (2) Obsérvese el uso de los puntos estructurados al final de las instrucciones SPECIAL-NAMES y SELECT.
- (3) Los canales 1, 2 y 5 del mecanismo de control de carro de la impresora se usarán para posicionar el carro en las tres áreas de los cheques impresos por este programa.
- (4) El programa usa dos ficheros. La información de entrada la toma del FICHERO-NOMINAS y los cheques los imprime en el FICHERO-CHEQUES. Los nombres externos (NOMINAS y CHEQUES) tienen formato IBM OS/VS.
- (5) Las cláusulas ORGANIZATION y ACCESS MODE se incluyen, aunque su valor por defecto es SEQUENTIAL. Se actúa así para mejorar la documentación del programa.
- (6) Ambos ficheros tienen asociadas áreas para el código de FILE STATUS; estas áreas pueden comprobarse después de cada operación en el fichero realizada en la división de procedimientos. ESTADO-NOMINAS y ESTADO-CHEQUES deben definirse en la división de datos. Estas áreas podrían haberse omitido y manejarse las condiciones excepcionales de otro modo (véase el Capítulo 11).

**Preguntas de repaso**

- 4.1** Describa la sintaxis de la división del entorno.
- 4.2** ¿Cuál es el propósito de la CONFIGURATION SECTION?
- 4.3** ¿Cuál es el propósito de la INPUT-OUTPUT SECTION?
- 4.4** Describa las reglas de escritura de la división del entorno.
- 4.5** Explique el propósito de las “líneas para la depuración”. ¿Cómo se ven afectadas por el párrafo SOURCE-COMPUTER?

- 4.6 ¿Cuál es el significado de “secuencia de orden de caracteres”?
- 4.7 ¿Cómo se implementa la secuencia de orden por defecto?
- 4.8 ¿Qué debe hacerse para utilizar una secuencia de orden distinta de la que se asume por defecto?
- 4.9 ¿Qué tres funciones tiene el párrafo SPECIAL-NAMES?
- 4.10 ¿Qué es el “papel continuo”?
- 4.11 ¿Qué es una cinta para el control del carro de la impresora?
- 4.12 Explique cómo se relacionan los canales de control del carro con las distintas áreas en un formulario en papel continuo.
- 4.13 ¿Qué utilizan las impresoras modernas en lugar de las cintas de control del carro?
- 4.14 En los sistemas IBM, ¿qué secuencias de orden de caracteres se llaman STANDARD-1 y NATIVE?
- 4.15 ¿Qué es un terminal con consola para el operador de la computadora?
- 4.16 ¿Qué función realizan las instrucciones ACCEPT y DISPLAY de la división de procedimientos? ¿Cómo se relacionan con el párrafo SPECIAL-NAMES de la división del entorno y configuración?
- 4.17 Explicar el significado de los siguientes “nombres de función” en el sistema IBM OS/VS: (a) SYSIN, (b) CONSOLE, (c) SYSOUT, (d) C01, (e) C04.
- 4.18 ¿Qué son los dispositivos de acceso directo?
- 4.19 ¿Qué es la operación “actualización en su lugar”?
- 4.20 Dé algunos ejemplos de registros y ficheros.
- 4.21 Describa cada parte de la instrucción SELECT y explique lo que realiza.
- 4.22 ¿Qué es el lenguaje de control de trabajos (JCL)?
- 4.23 ¿Qué es el lenguaje de órdenes del sistema operativo?
- 4.24 ¿Cómo se relaciona la cláusula ASSIGN TO de la instrucción SELECT con el lenguaje de control de trabajos?
- 4.25 Explique en que consiste el procesamiento (acceso) aleatorio de un fichero.
- 4.26 Explique en que consiste el procesamiento (acceso) secuencial de un fichero.
- 4.27 Defina el concepto “organización de un fichero”.
- 4.28 Explique brevemente el significado de fichero organizado secuencialmente.
- 4.29 Explique brevemente el significado de organización indexada de un fichero.
- 4.30 Explique brevemente el significado de organización relativa de un fichero.
- 4.31 ¿Por qué es tan importante la elección de la organización de un fichero?
- 4.32 Explique cuándo debe elegirse organización secuencial y cuándo organización indexada para un fichero.
- 4.33 Explique brevemente la función de las instrucciones OPEN, READ, WRITE y CLOSE de la división de procedimientos.
- 4.34 Diga algunas “condiciones excepcionales” que puedan producirse durante el procesamiento de ficheros.

- 4.35 ¿De qué modo la cláusula FILE STATUS de la instrucción SELECT se relaciona con el manejo de condiciones excepcionales?
- 4.36 ¿Qué es la condición "fin de fichero"?

### Problemas resueltos

- 4.37 Escriba la sección de configuración (CONFIGURATION SECTION) de un programa que correrá en un sistema IBM-370. Las líneas de depuración serán traducidas al programa objeto y se deberá usar la secuencia de orden de caracteres EBCDIC:

```
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-370 WITH DEBUGGING MODE.  
OBJECT-COMPUTER. IBM-370.
```

o

```
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-370  
WITH DEBUGGING MODE.  
OBJECT-COMPUTER. IBM-370  
PROGRAM COLLATING SEQUENCE IS EBCDIC-ORDER.  
SPECIAL-NAMES.  
EBCDIC-ORDER IS NATIVE.
```

- 4.38 Dar todas las variaciones legales de la línea 5, Figura 4-1.

(1) PROGRAM COLLATING SEQUENCE IS nombre-alfabeto; (2) PROGRAM SEQUENCE IS nombre-alfabeto; (3) PROGRAM SEQUENCE nombre-alfabeto; (4) COLLATING SEQUENCE IS nombre-alfabeto; (5) SEQUENCE IS nombre-alfabeto; (6) PROGRAM COLLATING SEQUENCE nombre-alfabeto; (7) COLLATING SEQUENCE nombre-alfabeto; (8) SEQUENCE nombre-alfabeto; (9) línea en blanco.

- 4.39 La impresora tiene que llenar cheques. Los canales de control de carro se preparan para los cheques como sigue: canal 1, fecha; canal 2, librado; canal 5, importe. El programa también imprime en la consola del operador el número total de cheques impresos. Escriba el párrafo SPECIAL-NAMES para este programa.

```
SPECIAL-NAMES.  
C01 IS AREA-FECHA-CHEQUE  
C02 IS AREA-LIBRADO-CHEQUE  
C05 IS AREA-IMPORTE-CHEQUE  
CONSOLE IS DISP-CONTROL-NUM-CHEQUES
```

- 4.40 Escriba la CONFIGURATION SECTION para un programa que utilice la siguiente secuencia de orden de caracteres: B, J, A, K,..., W, C,..., I, 7, 1, 0, 2,..., 6, 9. El programa está totalmente depurado y se va a compilar y ejecutar en una computadora IBM-370.

```
CONFIGURATION SECTION.  
SOURCE-COMPUTER. IBM-370.  
OBJECT-COMPUTER. IBM-370  
COLLATING SEQUENCE IS  
ORDEN-BIN-ALMACEN.
```

## SPECIAL-NAMES.

```

ORDEN-BIN-ALMACEN IS
  "B" "J" "A"
  "K" THRU "W"
  "C" THRU "I"
  "7" "1" "0"
  "2" THRU "6"
  "9"

```

- 4.41 Escriba la CONFIGURATION SECTION de un programa que va a ejecutarse en un sistema IBM-370 con la secuencia de orden de caracteres ASCII en lugar de la EBCDIC que se asumirá por defecto. Las líneas de depuración deben traducirse.

```

CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370 DEBUGGING MODE.
OBJECT-COMPUTER. IBM-370 SEQUENCE ASCII-ORDER.
SPECIAL-NAMES.
  ASCII-ORDER IS STANDARD-1.

```

- 4.42 Las impresoras, perforadoras de tarjetas y los lectores de tarjetas se denominan *periféricos de registro único*. Explique los conceptos campo, registro y fichero en relación con (a) un paquete de tarjetas; (b) un informe de ventas impreso en papel continuo.

- (a) Cada tarjeta es un *registro*; el paquete completo es un *fichero*; cada parte de información por separado es un *campo* (por ejemplo, número de artículo, cantidad, etc.).
- (b) El conjunto de todas las líneas impresas en el informe es el *fichero*; cada línea es un *registro*; cada porción de información de una línea es un *campo* (incluyendo los espacios en blanco de separación).

- 4.43 Describa cómo se puede actualizar en su lugar un registro de un fichero en disco relativo a una cuenta de ahorro en un banco.

Considérese un banco que disponga de terminales en línea (los terminales están conectados a la computadora del banco por medio de cable o mediante la telecomunicación). Si un cliente desea realizar un ingreso o reintegro de una cuenta de ahorro, el operador escribirá en el terminal el número de cuenta, el tipo de transacción y el importe. Esta información es procesada por un programa que accede al azar al registro del fichero que contiene el saldo de la cuenta de ahorro de ese cliente (para permitir el acceso al azar, el fichero probablemente estará indexado por el número de la cuenta de ahorro). El programa actualizará después el saldo sumando o restando el importe de la transacción. Por último, el registro actualizado volverá a ocupar su primitivo lugar en el disco.

- 4.44 Clasifique los siguientes nombres de ficheros en COBOL:

- (a) SELECT REPORT ASSIGN TO...
- (b) SELECT FICHERO-INFORME ASSIGN TO...
- (c) SELECT INFORME-PROD-ESCASOS ASSIGN TO...
- (d) SELECT FICHERO-ENTRADA ASSIGN TO...

indicando si son correctos, correctos pero incompletos o incorrectos.

(a) Incorrecto: "REPORT" es una palabra reservada. (b) Correcto, aunque pobre: "FICHERO-INFORME" no es muy descriptivo. (c) Correcto. (d) Correcto, aunque pobre: probablemente habrá varios ficheros de entrada.

- 4.45 Cada fichero procesado por un programa COBOL debe describirse con una instrucción SELECT en la sección de entrada/salida. ¿Cuál es el propósito de ASSIGN TO ... en la instrucción SELECT?

La división del entorno y configuración (al tratar del equipo que rodea al programa) es la parte del programa COBOL que *más posiblemente cambie* al pasar de un sistema a otro. Daremos dos ejemplos:

- (a) La versión de COBOL IBM OS/VS (para los sistemas IBM-370) precisa que cada fichero sea descrito adicionalmente en la instrucción DD (definición de datos) del lenguaje de control de trabajos. El nombre que aparece en la cláusula ASSIGN TO es el nexo de unión entre la instrucción DD y el programa fuente en COBOL. Así, en

```
SELECT MAESTRO-INVENTARIO ASSIGN TO MAESINVE...
```

la instrucción DD asociada será de la forma:

<i>Instrucción DD</i>	<i>Explicación</i>
//MAESINVE DD UNIT=DISK,	el diagnóstico es un disco
// VOL=SER=ABC123,	nombre del disco ABC123
// DISP=OLD,	el fichero ya existe
// DSNAME=MAESTRO.INV.A17	nombre del fichero en la etiqueta

La presencia del nombre ("MAESINVE") al principio de la instrucción DD es obligatoria.

- (b) La versión de COBOL Burroughs (para el sistema B1910) emplea un lenguaje de órdenes al sistema operativo en vez de un lenguaje de control de trabajos. Un usuario situado ante una consola CRT puede escribir un programa COBOL y después mediante las órdenes al sistema operativo hacer que se compile. Suponga que el programa fuente tiene la instrucción SELECT:

```
SELECT INVENTARIO ASSIGN TO DISK
```

Los nombres de fichero en el sistema B1910 están limitados a un máximo de 10 caracteres ("INVENTARIO"). La palabra "DISK" que aparece después de ASSIGN TO es reservada e indica al sistema qué "INVENTARIO" reside en el dispositivo del disco magnético. En el COBOL del sistema B1910 es el nombre del fichero "INVENTARIO" lo que sirve de nexo de unión entre la definición del fichero en el programa y las instrucciones en el lenguaje de órdenes del sistema operativo. Si el PROGRAM-ID fuera "ACTUALINVE", se podría escribir la siguiente orden para ejecutar el programa compilado:

```
EXECUTE ACTUALINVE; FILE INVENTARIO NAME INVMASA17
```

La orden EXECUTE dice al sistema operativo que ejecute el programa objeto indicado. La cláusula FILE conecta el fichero "INVENTARIO" descrito en el programa COBOL con el fichero en disco llamado "INVMASA17". Al ejecutarse el programa se actualizará el fichero en disco denominado INVMASA17.

- 4.46** Indique la organización y modo de acceso que elegiría para los ficheros que se manipularán en las siguientes aplicaciones: (a) un fichero maestro de límites de crédito de una compañía que permite a otros comerciantes llamar y alterar el límite de créditos de una tarjeta de cliente; (b) un fichero en cinta de nombres y direcciones de una lista de correos; (c) un fichero de cobros pendientes con registros de la cantidad adeudada por cada cliente.

- (a) Organización: indexada (por el número de tarjeta de crédito). Acceso: al azar.  
 (b) Organización: secuencial (*única posible cuando no se dispone de un periférico de acceso directo*). Acceso secuencial (*único posible con cinta*).  
 (c) Organización: indexada (por número de cliente) o bien secuencial (clasificado por número de cliente).

Acceso: El fichero debe actualizarse con información sobre pagos y nuevas compras. Si la organización es indexada, la actualización puede hacerse al azar. Si los pagos y compras también se clasifican por número de cliente, la actualización se puede llevar a cabo con acceso al azar o secuencial. Los informes impresos (como un inventario por fechas) se llevarán a cabo con acceso secuencial.

- 4.47** ¿Qué es la condición "fin de fichero"?

El concepto "fin de fichero" se aplica cuando un fichero se procesa secuencialmente. El primer registro del fichero se lee primero, después el segundo, etc. Si el fichero tuviera 7.000 registros, la condición "fin de fichero" tendría lugar cuando falla el intento de leer el registro 7.001.

- 4.48** Escriba la división del entorno completa para un programa que imprima cheques. Usará un fichero de entrada en cinta con registros conteniendo el tiempo trabajado semanal clasificado

por número de empleado y otro fichero de entrada en disco con registros maestros para cada empleado también clasificado por número de empleado. Todos los ficheros se procesan secuencialmente y el programa ya está depurado. Los canales de control del carro no se usan en este programa, aunque se pueden imprimir algunos mensajes en la consola del operador de la computadora.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-370.

OBJECT-COMPUTER. IBM-370.

SPECIAL-NAMES.

CONSOLE IS DISPOSITIVO-MENSAJES-OPERADOR

INPUT-OUTPUT SECTION.

FILE-CONTROL.

```
SELECT FICHERO-TIEMPO-SEMANAL  
      ASSIGN TO TIEMSEMA  
      ORGANIZATION IS SEQUENTIAL  
      ACCESS IS SEQUENTIAL  
      FILE STATUS IS ESTADO-FICHERO-TIEMPO
```

```
SELECT FICHERO-MAESTRO-EMPLEADOS  
      ASSIGN TO MAESEMPL  
      ORGANIZATION IS SEQUENTIAL  
      ACCESS IS SEQUENTIAL
```

```
SELECT FICHERO-CHEQUES  
      ASSIGN TO CHEQUES  
      ORGANIZATION IS SEQUENTIAL  
      ACCESS IS SEQUENTIAL
```

(¿Olvidó que los cheques impresos constituyen un fichero y por tanto requieren una instrucción SELECT?). *Observe:* (i) los puntos estructurados y las líneas en blanco; (ii) la omisión de las cláusulas DEBUGGING MODE y COLLATING SEQUENCE; (iii) la asociación del nombre COBOL "DISPOSITIVO-MENSAJES-OPERADOR" con la consola (CONSOLE) del sistema; (iv) que al FICHERO-TIEMPO-SEMANAL se le asocia un área (denominada ESTADO-FICHERO-TIEMPO y más tarde definida en la división de datos) en la cual se crea un código después de cada operación con el fichero.

## Ejercicios de programación

4.49-4.58 Modifique los Problemas 2.29-2.38 para reflejar su mejor conocimiento de la división del entorno.

# Capítulo 5

## División de datos (DATA DIVISION)

### 5.1 ESTRUCTURA DE LA DIVISION DE DATOS

La Figura 5-1 indica la estructura de la división de datos (DATA DIVISION), cuya función principal se describió en la Sección 2.1.

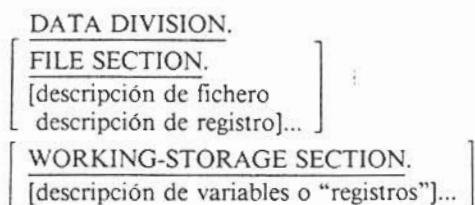


Fig. 5-1

Las dos secciones (FILE SECTION y WORKING-STORAGE SECTION) son opcionales. En la práctica, las dos son necesarias porque casi todos los programas utilizan al menos un fichero y algunos contadores, variables, banderas, etc., de la WORKING-STORAGE SECTION. En el Ejemplo 5.1 se da la división de datos correspondiente a la división del entorno que aparecía en el Ejemplo 4.1.

#### EJEMPLO 5.1

```
00066      DATA DIVISION.
00067
00068      FILE SECTION.
00069
00070      FD  FICHERO-VENTAS-CLIENTES
00071          BLOCK CONTAINS 0 RECORDS
00072          RECORD CONTAINS 80 CHARACTERS
00073          LABEL RECORDS ARE STANDARD
00074
00075
00076      01  REGISTRO-VENTAS-CLIENTES.
00077          05  NUM-CLIENTE-VENTAS      PIC X(4).
00078          05  IDENT-CLIENTE        PIC X(6).
00079          05  IMPORTE-VENTAS      PIC S9(7)V99.
00080          05  RELLENO             PIC X(61).
00081
00082      FD  INFORME-VENTAS
00083          RECORD CONTAINS 132 CHARACTERS
00084          LABEL RECORDS ARE OMITTED
00085
00086
00087      01  LINEA-INFORME-VENTAS      PIC X(132).
00088
00089      WORKING-STORAGE SECTION.
00090
00091      01  AREA-FIN-FICHERO-ESTADO.
00092
00093          05  FIN-FICHERO-VENTAS-CLIENTES  PIC X.
00094          88  NO-MAS-REGISTROS-CLIENTES    VALUE "T".
00095          05  ESTADO-VENTAS            PIC XX.
00096          05  ESTADO-INFORME          PIC XX.
00097
```

```

00098      01 AREAS-TOTAL-Y-COMPARACION.
00099
00100          05 NUM-VIEJO-CLIENTE           PIC X(4).
00101          05 TOTAL-CLIENTE             PIC S9(9)V99  COMP-3.
00102          05 GRAN-TOTAL              PIC S9(9)V99  COMP-3.
00103
00104      01 AREA-CONTADORES.
00105
00106          05 CONTADOR-LINEAS          PIC S9(3)    COMP-3.
00107          05 CONTADOR-PAGINAS        PIC S9(5)    COMP-3.
00108          05 SALTO-DE-PAGINA         PIC S9(3)    COMP-3.
00109
00110      01 CABECERA-LINEA.
00111
00112          05 FILLER                 PIC X(10)   VALUE SPACES.
00113          05 FILLER                 PIC X(20)
00114                           VALUE "VENTAS POR CLIENTES".
00115          05 FILLER                 PIC X(10)   VALUE SPACES.
00116          05 CONTADOR-PAGINAS        PIC ZZ,ZZ9.
00117          05 FILLER                 PIC X(86)   VALUE SPACES.
00118
00119      01 LINEA-ERRORES.
00120
00121          05 ERROR-NUM-CLIENTE        PIC X(4).
00122          05 FILLER                 PIC X(2)    VALUE SPACES.
00123          05 FILLER                 PIC X(22)
00124                           VALUE "*** DESORDENADO ***".
00125          05 FILLER                 PIC X(104)  VALUE SPACES.
00126
00127      01 LINEA-TOTAL.
00128
00129          05 FILLER                 PIC X(30)   VALUE SPACES.
00130          05 TOTAL-CLIENTE          PIC $ $$,$ $$,$ $.99-
00131          05 FILLER                 PIC XX     VALUE " *".
00132          05 BANDERA-GRAN-TOTAL    PIC X.
00133          05 FILLER                 PIC X(83)   VALUE SPACES.
00134
00135      01 LINEA-SALIDA.
00136
00137          05 NUM-CLIENTE-SALIDA      PIC X(4).
00138          05 FILLER                 PIC X(10)   VALUE SPACES.
00139          05 ID-VENTAS              PIC X(6).
00140          05 FILLER                 PIC X(10)   VALUE SPACES.
00141          05 IMPORTE-SALIDA        PIC $ $$,$ $$,$ $.99-
00142          05 FILLER                 PIC X(88)   VALUE SPACES.

```

## 5.2 SECCION DE FICHEROS (FILE SECTION)

La sección de ficheros debe contener una *entrada de descripción de fichero* ("FD") por cada fichero que aparezca en una instrucción SELECT de la división del entorno.

**EJEMPLO 5.2** En el Ejemplo 4.1,

línea 52: SELECT FICHERO-VENTAS-CLIENTES  
 línea 59: SELECT INFORME-VENTAS

se refieren a dos ficheros que son procesados por el programa. Las cláusulas FD correspondientes en el Ejemplo 5.1 están en las líneas 70 y 82.

A continuación de la información contenida en FD, cada fichero debe tener al menos al nivel 01 una *entrada de descripción del registro*, describiendo los registros del fichero. La estructura sintáctica de la cláusula FD aparece en la Figura 5-2.

```

FD  nombre-fichero
    [ BLOCK CONTAINS [entero-1 TO] entero-2 {CHARACTERS}
        [RECORD CONTAINS [entero-3 TO] entero-4 CHARACTERS]
        LABEL {RECORD IS } {STANDARD}
            [RECORDS ARE ] { OMITTED }
        [ DATA { RECORD IS } nombre-de-datos-3 [nombre-de-datos-4]...
            [RECORDS ARE ] ]
    [ LINAGE IS { nombre-de-datos-5 } entero-5 LINES
        [ WITH FOOTING AT { nombre-de-datos-6 } entero-6 ]
        [ LINES AT TOP { nombre-de-datos-7 } entero-7 ]
        [ LINES AT BOTTOM { nombre-de-datos-8 } entero-8 ]
    ]

```

Fig. 5-2

La cabecera de descripción del fichero ("FD") debe escribirse comenzando en el margen A; el resto de las entradas se escribirán en el margen B. El "nombre de fichero" (el mismo que aparece en la instrucción SELECT para el fichero) es la primera entrada. El resto de las entradas se pueden escribir en cualquier orden. Las líneas en blanco (o líneas de comentarios en blanco) y los desplazamientos deben usarse para aumentar la facilidad de lectura. Hay *un punto único*, situado al final de la instrucción FD, y es mejor que dicho punto sea estructurado.

### EJEMPLO 5.3

DATA DIVISION.

FILE SECTION.

FD INFORME-SEMANAL-PERSONAL

```

BLOCK CONTAINS 1 RECORD
RECORD CONTAINS 132 CHARACTERS
LABEL RECORDS ARE OMITTED
DATA RECORD IS REGISTRO-INFORME-PERSONAL
LINAGE IS 50 LINES
    WITH FOOTING AT 45
    LINES AT TOP 8
    LINES AT BOTTOM 8

```

01 REGISTRO-INFORME-PERSONAL...

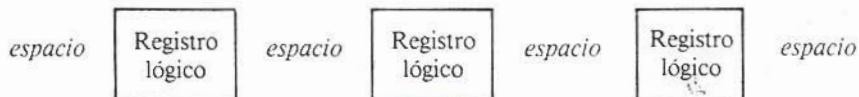
Observe el uso de líneas en blanco, desplazamiento de líneas y punto estructurado. En la descripción del registro que sigue a la instrucción FD se detalla cada campo del fichero. El nombre que sigue a FD ("INFORME-SEMANAL-PERSONAL") debe ser el mismo que se utilizó en la instrucción SELECT para el fichero.

En las Secciones 5.3 a 5.7 que siguen, se comentan las cláusulas de la instrucción FD que aparecen en la Figura 5-2, una por una. Tenga en cuenta que el objeto de cada cláusula es proporcionar información con vistas al almacenamiento de los registros del fichero en el dispositivo correspondiente.

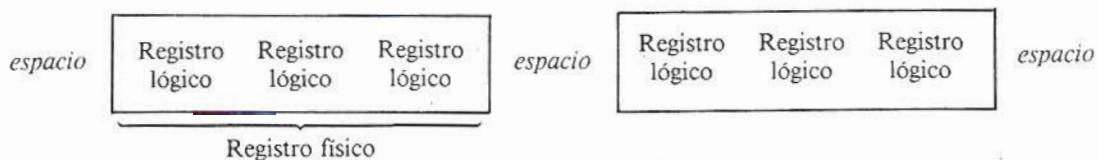
### 5.3 FD BLOCK CONTAINS...

Los ficheros almacenados en disco o cinta frecuentemente se *agrupan en bloques* para hacer más eficiente la utilización del espacio y aumentar la velocidad de procesamiento. En la operación de bloqueo se distingue entre *registros lógicos* (procesados uno cada vez por el programa) y *registros físicos* (unidad de información transferida desde o hacia el dispositivo de entrada/salida).

**EJEMPLO 5.4** Un *fichero no bloqueado* en cinta o en disco se compone de *registros lógicos* separados por *espacios interregistros*:



En un *fichero bloqueado*, los registros lógicos se agrupan en *registros físicos* o *bloques* separados por *espacios interbloques*:



El fichero bloqueado indicado tiene tres registros lógicos por cada registro físico; se dice que el *factor de bloqueo* es 3 (BF=3), en el fichero no bloqueado BF=1. En cualquier fichero, todos los caracteres entre dos espacios se manipulan de una vez. Por tanto, durante un procesamiento secuencial, el fichero bloqueado requerirá sólo 1/3 de las operaciones de entrada/salida que hubiera requerido el fichero no bloqueado —con la consecuencia de que el tiempo de posicionamiento en la cinta o disco se reducirá aproximadamente a 1/3 del tiempo empleado en el otro caso—. Como en el fichero bloqueado hay 1/3 de los espacios en blanco del fichero no bloqueado, también se optimiza el uso del espacio disponible.

**EJEMPLO 5.5** Supóngase que se desea procesar aleatoriamente un fichero no secuencial de 20.000 registros. Mas aún, supóngase que por término medio sólo la mitad de los registros del fichero son procesados en una ejecución. Si el fichero no estuviera bloqueado, se necesitarían alrededor de 10.000 operaciones de entrada para obtener los 10.000 registros lógicos que han de procesarse.

Si el fichero tuviera un factor de bloqueo de 10, todavía necesitaría aproximadamente 10.000 operaciones de entrada para conseguir los 10.000 registros lógicos que tienen que procesarse (porque en el procesamiento al azar el siguiente registro lógico no tiene que estar necesariamente en el bloque que acaba de leerse). El tiempo utilizado para transferir los datos que no van a utilizarse se malgasta. Aunque el bloqueo puede aumentar la velocidad de un procesamiento secuencial, sin embargo disminuye la velocidad en un procesamiento al azar. No obstante, algunos ficheros que vayan a procesarse al azar pueden que permanezcan bloqueados para ahorrar espacio en el disco.

La cláusula BLOCK CONTAINS indica si el fichero está o no bloqueado y especifica el factor de bloqueo. El sistema operativo automáticamente ensamblará los registros lógicos en bloques para la salida y analizará los bloques, distinguiendo los registros lógicos en el proceso de entrada. La división de procedimientos *siempre se escribe para trabajar con registros lógicos individuales sin tener en cuenta el factor de bloqueo BF* (véase la Sección 5.8). Existen tres posibilidades:

**Fichero no bloqueado.** En este caso, la cláusula BLOCK CONTAINS se omite en la instrucción FD:

```
FD INFORME-SEMANAL-PERSONAL
    RECORD CONTAINS 132 CHARACTERS
    LABEL RECORDS ARE OMITTED
```

```
01 REGISTRO-INFORME-PERSONAL...
```

Como el bloqueo sólo es posible en medios magnéticos como la cinta o el disco, los ficheros de la impresora no son bloqueados.

**Bloques de longitud fija.** Utilizar la cláusula BLOCK CONTAINS entero-2 CHARACTERS:

FD FICHERO-MAESTRO-INVENTARIO  
 BLOCK CONTAINS 500 CHARACTERS  
 RECORD CONTAINS 100 CHARACTERS  
 LABEL RECORDS ARE STANDARD

o utilizar BLOCK CONTAINS entero-2 RECORDS:

FD FICHERO-MAESTRO-INVENTARIO  
 BLOCK CONTAINS 5 RECORDS  
 RECORD CONTAINS 100 CHARACTERS  
 LABEL RECORDS ARE STANDARD

En el primer ejemplo, el factor de bloqueo BF es

$$\frac{500 \text{ bytes/bloque}}{100 \text{ bytes/negativo}} = 5 \text{ registros/bloque}$$

mientras que en el segundo ejemplo la longitud de bloque es

$$(5 \text{ registros/bloque})(100 \text{ bytes/registro}) = 500 \text{ bytes/bloque}$$

Las dos formas son completamente equivalentes.

**Bloques de longitud variable.** (¿Qué ocurre cuando los registros lógicos de un fichero son de longitud variable?). Utilizar:

BLOCK CONTAINS entero-1 TO entero-2 CHARACTERS  
 RECORD CONTAINS entero-3 TO entero-4 CHARACTERS

(véase el Ejemplo 5.6) o use

BLOCK CONTAINS entero-2 RECORDS  
 RECORD CONTAINS entero-3 TO entero-4 CHARACTERS

(véase el Ejemplo 5.7).

#### EJEMPLO 5.6

FD FICHERO-MAESTRO-CLIENTES  
 BLOCK CONTAINS 220 TO 420 CHARACTERS  
 RECORD CONTAINS 50 TO 100 CHARACTERS  
 LABEL RECORDS ARE STANDARD

La Figura 5-3 indica la estructura de un bloque de longitud variable (BF=3) en la versión de COBOL para el sistema IBM OS/VS.

...	espacio	BDW	RDW	Registro lógico	RDW	Registro lógico	RDW	Registro lógico	espacio	...
-----	---------	-----	-----	-----------------	-----	-----------------	-----	-----------------	---------	-----

BDW  $\equiv$  palabra de descripción de bloque (insertada automáticamente por el compilador) que especifica la longitud, incluyendo los 4 bytes del BDW.

RDW  $\equiv$  palabra de descripción del registro (automáticamente insertada por el compilador) que especifica la longitud del registro lógico al que precede, incluyendo los 4 bytes de la RDW.

Fig. 5-3

La cláusula RECORD CONTAINS *excluye* la RDW, pero la cláusula BLOCK CONTAINS...CHARACTERS *incluye* la BDW y la RDW. Por tanto, el bloque más corto del fichero contiene

$$\frac{220-4}{50+4} = 4 \text{ registros}$$

y el bloque más largo contiene

$$\frac{420-4}{100+4} = 4 \text{ registros}$$

En resumen, BF=4 para el fichero.

#### EJEMPLO 5.7

```
FD FICHERO-MAESTRO-CLIENTES
BLOCK CONTAINS 4 RECORDS
RECORD CONTAINS 50 TO 100 CHARACTERS
LABEL RECORDS ARE STANDARD
```

En este caso, al contrario que en el Ejemplo 5.6, el factor de bloqueo se fija explícitamente: BF=4. Las longitudes máxima y mínima del bloque son

$$4(50+4)+4=220 \text{ bytes} \quad 4(100+4)+4=420 \text{ bytes}$$

y se ve que la instrucción FD es equivalente a la del Ejemplo 5.6

#### 5.4 FD RECORD CONTAINS...

La aplicación de esta cláusula a registros de longitud fija y variable ya se ha estudiado en la Sección 5.3. La cláusula es opcional porque el compilador puede determinar la longitud del registro a partir de la descripción del mismo en el nivel 01 (que incluye la longitud de cada campo). Sin embargo, es buena idea utilizar siempre la cláusula RECORD CONTAINS... para mejorar la documentación del programa y porque algunos compiladores generan mensajes de atención si las longitudes de los campos no suman el valor especificado en la frase RECORD CONTAINS...

#### BLOCK/RECORD CONTAINS 0 CHARACTERS

En la versión de COBOL para IBM OS/VS se puede especificar

BLOCK CONTAINS 0 {CHARACTERS  
RECORDS} o RECORD CONTAINS 0 CHARACTERS

Esto es una *ampliación* de la norma ANS, que requiere un número positivo de caracteres o registros. En el COBOL de IBM, el uso de cero indica que el tamaño del bloque o registro debe tomarse de las instrucciones en el lenguaje de control de trabajos que describen el fichero (véase la Sección 4.5). Aunque RECORD CONTAINS 0 CHARACTERS tiene poca aplicación en COBOL, BLOCK CONTAINS 0... puede y *debe* usarse para hacer un programa en COBOL *independiente* del factor de bloqueo (de manera que un cambio en el factor de bloquéo no implique volver a compilar el programa fuente en COBOL). (Recuerde que los ficheros de impresora y los ficheros en tarjetas no pueden bloquearse.)

#### EJEMPLO 5.8

```
FD FICHERO-FACTURAS-ABIERTAS
BLOCK CONTAINS 0 CHARACTERS
RECORD CONTAINS 250 CHARACTERS
LABEL RECORDS ARE STANDARD
```

La longitud del bloque se tomará de la instrucción en el lenguaje de control de trabajos asociado al fichero en el momento de la ejecución. Si el fichero ya existe y dispone de una etiqueta, la longitud del bloque también

se puede tomar de la misma (eliminando así la necesidad de especificar la longitud en la instrucción del lenguaje de control de trabajos).

### 5.5 FD LABEL RECORDS...

Recuerde que como se indicó en la Sección 1.10, cada fichero en un dispositivo de acceso directo dispone de una *etiqueta de fichero* (automáticamente creada por el sistema operativo) donde se indican: el nombre por el que el sistema operativo reconoce al fichero, la organización (secuencial, indexada o relativa), la dirección del área de índices (si existe), la dirección de los registros de datos, tamaño del bloque, tipo de registro (longitud fija o variable), la longitud del registro, la palabra clave para el acceso, el tiempo durante el cual no podrá borrarse, etc. Las etiquetas de ficheros en un disco se agrupan en un fichero especial del sistema operativo denominado *tabla de contenido del volumen (VTOC)* o *directorio*.

Las etiquetas de ficheros son opcionales en los ficheros en cinta magnética, pero como consecuencia de las ventajas que ofrecen casi siempre se utilizan. Las etiquetas de fichero no existen en los ficheros contenidos en dispositivos de registro único (lectores de tarjetas, perforadoras e impresoras).

LABEL RECORDS... es la única cláusula obligatoria en la instrucción FD. Utilice

LABEL RECORDS ARE OMITTED

si el fichero está en un dispositivo de registro único o si el fichero está en cinta pero no tiene etiquetas. Utilice

LABEL RECORDS ARE STANDARD

si el fichero está en un dispositivo de acceso directo (disco, disco flexible, tambor, etc.) o si está en una cinta magnética con etiquetas.

### 5.6 FD DATA RECORDS...

La cláusula DATA RECORDS lista los nombres definidos por el programador con los que se denomina a los registros del fichero en la división de procedimientos. Esta cláusula tiene una utilidad limitada, sobre todo con registros de longitud fija, puesto que los nombres de los registros en COBOL siguen a la instrucción FD (son la primera entrada al nivel 01 de la descripción del registro). Los nombres de registros que aparezcan en el nivel 01 deben coincidir con los que aparezcan en la cláusula DATA RECORDS. Esta cláusula es opcional en todos los casos.

#### EJEMPLO 5.9

```
FD FICHERO-TIEMPO-SEMANAL
BLOCK CONTAINS 436 TO 1636 CHARACTERS
RECORD CONTAINS 50 TO 200 CHARACTERS
LABEL RECORDS ARE STANDARD
DATA RECORDS ARE REGISTRO-TIEMPO-PERDIDO
                           REGISTRO-TIEMPO-ABOGADOS
```

```
01 REGISTRO-TIEMPO-PERDIDO
   (descripción de un registro lógico de 50 bytes)
01 REGISTRO-TIEMPO-ABOGADOS
   (descripción de un registro lógico de 200 bytes)
```

Se asume en el COBOL de IBM OS/VS un factor de bloqueo para FICHERO-TIEMPO-SEMANAL de 8.

Los ficheros en tarjetas (en ningún caso pueden estar bloqueados) tienen a menudo más de un tipo de registro porque todos los campos que se necesitan no caben en una tarjeta. Cuando sucede esto, la información se divide entre varias tarjetas y cada tarjeta se codifica para indicar los campos que contiene. El código de tarjeta se suele ubicar en el primer o en el último byte del registro (tarjeta).

**EJEMPLO 5.10** Se desea colocar etiquetas de correo en tarjetas de 80 columnas. Como los datos requieren más de 80 caracteres, el fichero contendrá tres tipos de registro. Las descripciones del fichero (FD) y de los registros (nivel 01) podrían ser:

```

FD FICHERO-ETIQUETAS-CORREO
RECORD CONTAINS 80 CHARACTERS
LABEL RECORDS ARE OMITTED
DATA RECORDS ARE REGISTRO-TIPO1-NOMBRE
                           REGISTRO-TIPO2-CIUDAD
                           REGISTRO-TIPO3-NEGOCIO

01 REGISTRO-TIPO1-NOMBRE
 05 CODIGO-1          PIC X
 05 ID-1              PIC X(9).
 05 NOMBRE-1          PIC X(30).
 05 CALLE-1           PIC X(30).
 05 FILLER            PIC X(10).

01 REGISTRO-TIPO2-CIUDAD
 05 CODIGO-2          PIC X.
 05 ID-2              PIC X(9).
 05 TELEFONO-2        PIC X(8).
 05 CIUDAD-2          PIC X(30).
 05 ESTADO-2          PIC X(2).
 05 CODIGO-POSTAL-2   PIC X(9).
 05 FILLER            PIC X(21).

01 REGISTRO-TIPO3-NEGOCIO
 05 CODIGO-3          PIC X.
 05 ID-3              PIC X(9).
 05 NEGOCIO-3         PIC X(70).

```

Obsérvese que la cláusula BLOCK CONTAINS... se omite para los ficheros en tarjetas. RECORD CONTAINS... se utiliza para que el compilador genere un mensaje de atención en el caso de que la descripción de un registro a nivel 01 no sume 80 caracteres (comprobación útil, puesto que se intenta no tener registros de longitud variable, sino tres tipos de registros todos con la misma longitud). La cláusula LABEL RECORDS... se omite para el fichero de tarjetas. DATA RECORDS... sirve como documentación del programa, facilitando la comprensión de que hay tres tipos de registros en el fichero. ¿Podría enumerar los campos de cada tipo de registro lógico?

## 5.7 FD LINAGE...

La cláusula LINAGE sólo se utiliza para los ficheros de impresora y siempre es opcional. Cuando se utiliza, define una *página lógica* que se compone de un *margen superior*, un *margen inferior* y un *cuerpo de página* en el que tiene lugar la escritura. Un área inferior del cuerpo de página se puede destinar para las *notas al pie*. LINAGE... es una *alternativa* al uso de los canales de control del carro (Sección 4.4); los dos métodos no se pueden utilizar al mismo tiempo para el mismo fichero. La cláusula LINAGE... no se puede utilizar con todos los compiladores. Véase el Problema 8.30 para conocer técnicas de control del paginado sin LINAGE.

## EJEMPLO 5.11

```

FD INFORME-VENTAS-POR-VENDEDORES
RECORD CONTAINS 132 CHARACTERS
LABEL RECORDS ARE OMITTED
LINAGE IS 54 LINES
      WITH FOOTING AT 44
      LINES AT TOP 6
      LINES AT BOTTOM 6

```

El fichero de impresión no tiene bloques ni etiqueta. La cláusula LINAGE define una página lógica como se indica en la Figura 5-4. Toda la impresión se efectuará en el cuerpo de página (sombreado).

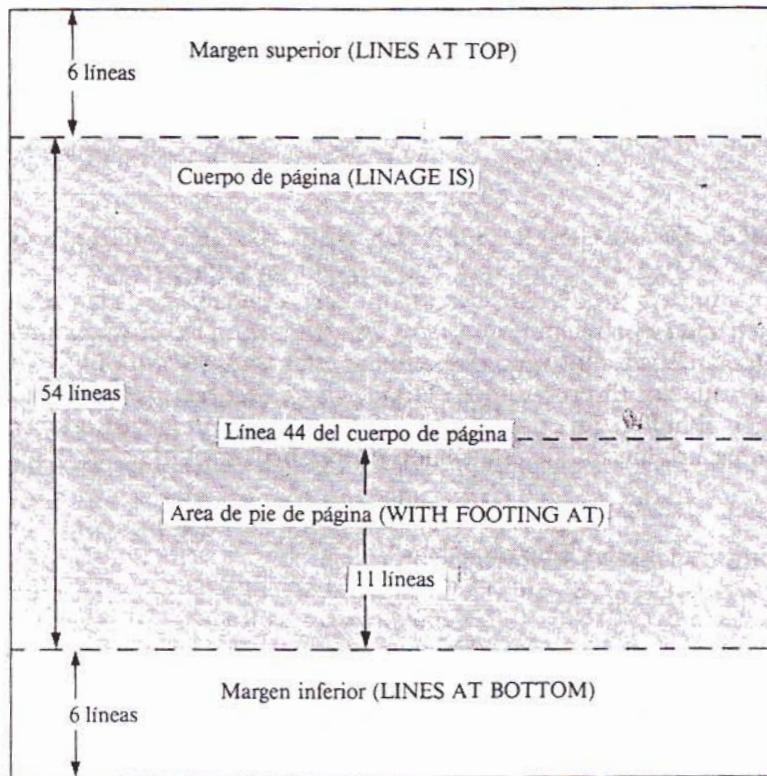


Fig. 5-4

Los márgenes superior e inferior de una página lógica siempre se dejan en blanco, por tanto los márgenes sirven sólo para separar un cuerpo de página de otro. Tanto "LINES AT TOP..." como "LINES AT BOTTOM..." son opcionales; si se omite cualquiera de ellos, el margen correspondiente contendrá cero líneas.

El área de notas al pie dentro del cuerpo de página se puede utilizar para: (1) imprimir los totales de página de ciertos campos, (2) imprimir un mensaje cuando el informe continúa en la página siguiente o (3) asegurarse que hay suficiente espacio al final de la página para completar cualquier cosa sin necesidad de acudir a otra página. El área de notas al pie es opcional y si se omite "WITH FOOTING AT" no existirá esta área al final del cuerpo de página.

El cuerpo de página es donde tiene lugar la escritura. Véase el Capítulo 6 para conocer las consideraciones que hay que tener en cuenta en la división de procedimientos con las instrucciones WRITE relativas a ficheros con LINAGE...

#### EJEMPLO 5.12

- FD INFORME-PROD-BAJO-MINIMO  
RECORD CONTAINS 132 CHARACTERS  
LABEL RECORDS ARE OMITTED  
LINAGE IS 66 LINES

Cada cuerpo de página tiene 66 líneas y ocupa toda la página. No existe área de notas al pie, pero la detección de un área de notas al pie será apuntada por la instrucción WRITE cuando se alcance la parte inferior de la página.

- FD INFORME-VENTAS-POR-CLIENTES  
RECORD CONTAINS 132 CHARACTERS  
LABEL RECORDS OMITTED  
LINAGE 50 LINES  
LINES AT TOP TAMANO-MARGEN-SUPERIOR  
LINES AT BOTTOM 5

Todas o alguna de las cláusulas LINAGE pueden especificar un valor que ha sido almacenado antes de abrir el fichero (prepararle para el procesamiento, véase el Ejemplo 2.5). En este caso, dicho entero está contenido en la posición TAMANYO-MARGEN-SUPERIOR cuando se abre el fichero INFORME-VENTAS-POR-CLIENTES e indica la anchura del margen superior. De este modo el programa puede *calcular* la anchura de las áreas asociadas con LINAGE.

## 5.8 DESCRIPCION DE REGISTROS EN LA SECCION DE FICHEROS

Cuando un programa objeto COBOL procesa un fichero, una parte de la memoria de la computadora debe estar preparada para recibir un *registro físico* (desde un fichero de entrada) o para construir un registro físico (con destino a un fichero de salida); estas áreas se conocen con el nombre de *buffer* de un fichero determinado. El propósito de la *descripción del registro* es detallar el *registro lógico* tal como aparece en el buffer. En otras palabras, la descripción del registro especifica que parte del buffer contiene el registro lógico que está siendo procesado.

**EJEMPLO 5.13** El segmento de programa del Ejemplo 5.9 continúa así:

```

FD  FICHERO-TIEMPO-SEMANAL
    BLOCK CONTAINS 436 TO 1636 CHARACTERS
    RECORD CONTAINS 50 TO 200 CHARACTERS
    LABEL RECORDS ARE STANDARD
    DATA RECORDS ARE REGISTRO-TIEMPO-PERDIDO
                                REGISTRO-TIEMPO-ABOGADOS

01  REGISTRO-TIEMPO-PERDIDO.
    (descripción de un registro lógico de 50 bytes)
01  REGISTRO-TIEMPO-ABOGADOS.
    (descripción de un registro lógico de 200 bytes)
...
PROCEDURE DIVISION.
...
OPEN INPUT FICHERO-TIEMPO-SEMANAL
...
READ FICHERO-TIEMPO-SEMANAL RECORD
...

```

Las instrucciones de la división de procedimientos ilustran el modo de preparar un fichero para la entrada ("OPEN INPUT FICHERO-TIEMPO-SEMANAL") y cómo se introduce un registro del fichero ("READ FICHERO-TIEMPO-SEMANAL RECORD"). La instrucción READ realiza las siguientes funciones:

- (1) Si el buffer estuviera vacío, capturaría un registro físico y pondría de este modo un registro lógico a disposición del programa.
- (2) Si el buffer ya contiene un registro físico del cual ya se han leído todos los registros lógicos, se repite la operación (1).
- (3) Si el buffer ya contiene un registro físico del cual no se han leído todos los registros lógicos, pone a disposición del programa el *siguiente* registro lógico (no procesado).

En cualquier caso, después de cada instrucción READ hay un *registro lógico* a disposición del programa. *Depende del programador reconocer el tipo de registro lógico de que se trate*. El procedimiento general consiste en reservar un byte de cada registro para el *código de registro*. De este modo el primer byte de los registros de tiempo perdido podría contener una "C", mientras que el primer byte de los registros de tiempo de abogados podría contener una "L". Un registro tipo "C" recibirá un procesamiento tipo "C" por parte del programa y un registro tipo "L" recibirá un procesamiento tipo "L".

### Números de nivel

Como parte esencial de la descripción del registro, se asigna a los campos unos *números de nivel* de 01 a 49. El nivel 01 se reserva para el nombre del registro. Si el número de nivel aumenta al pasar de un campo al siguiente, se dice que el segundo es un *subcampo* del primero.

**EJEMPLO 5.14**

```

FD FICHERO-DE-PAGOS
BLOCK CONTAINS 100 RECORDS
RECORD CONTAINS 33 CHARACTERS
LABEL RECORDS ARE STANDARD

01 REGISTRO-DE-PAGOS.
  05 NUMERO-DE-CLIENTE      PIC X(5).
  05 ZONA-DEL-CLIENTE      PIC X(2).
  05 FECHA-DEUDA.
    10 MES-DEUDA           PIC X(2).
    10 DIA-DEUDA           PIC X(2).
    10 AÑO-DEUDA           PIC X(2).
  05 FECHA-RECIBO-PAGO
    10 MES-RECIBO          PIC X(2).
    10 DIA-RECIBO          PIC X(2).
    10 AÑO-RECIBO          PIC X(2).
  05 SALDO-ANTERIOR       PIC S9(5)V99.
  05 IMPORTE-PAGO         PIC S9(5)99.

```

El nivel 01 se empieza a escribir en el margen A e indica el comienzo de la descripción del registro. Los otros números de nivel, en este caso 05 y 10, se escriben en el margen B. Obsérvese el uso de la jerarquización de las líneas en la división de datos. Aunque el compilador no lo precisa, es muy importante para poner de manifiesto la estructura del registro.

La estructura definida por los números de nivel aparece en el diagrama de la Figura 5-5.

Los campos con subcampos se llaman *elementos compuestos*; los campos que no tienen subcampos se llaman *elementos simples*.

**EJEMPLO 5-15** En el Ejemplo 5.14:

elementos compuestos	REGISTRO-DE-PAGOS FECHA-DEUDA FECHA-RECIBO-PAGO
elementos simples	NUMERO-DE-CLIENTE ZONA-CLIENTE MES-DEUDA DIA-DEUDA AÑO-DEUDA MES-RECIBO DIA-RECIBO AÑO-RECIBO SALDO-ANTERIOR IMPORTE-PAGO

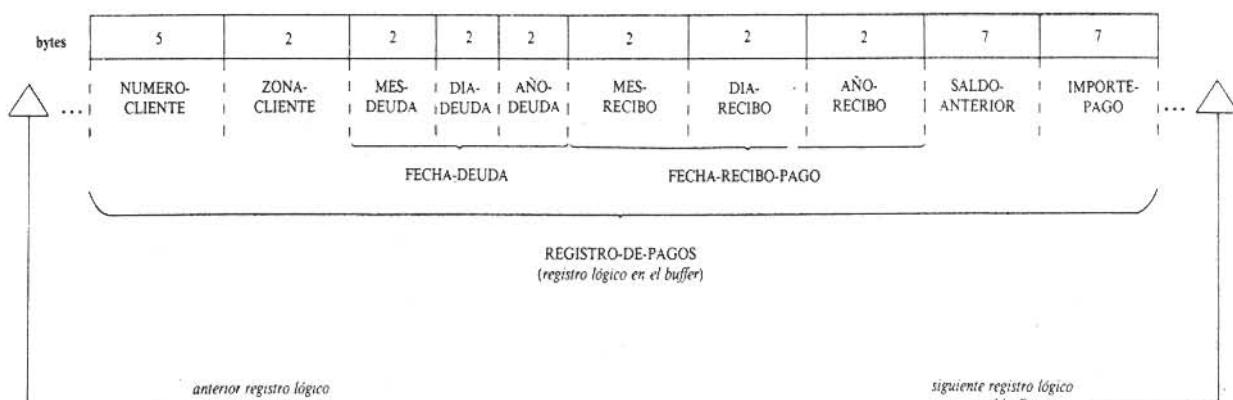


Fig. 5-5

Obsérvese que los elementos simples tienen una cláusula "PIC" que describe su longitud y tipo de datos. Los elementos compuestos no tienen tal descripción, porque quedan completamente definidos por sus componentes elementales. *Los nombres definidos por el programador para elementos simples y compuestos siempre se refieren al registro lógico en activo contenido en el buffer.*

Se recomienda como norma que una abreviatura del nombre del fichero o del registro sirva como prefijo a cada elemento (campo) del registro. De este modo se conocerá a qué registro o fichero pertenece.

#### EJEMPLO 5.16

```
FD FICHERO-ETIQUETAS-CORREO
  RECORD CONTAINS 76
    LABEL RECORDS OMITTED
      DATA RECORDS ETIQ-CORREO-CALLE-REG
          ETIQ-CORREO-ESTADO-REG
```

01 ETIQ-CORREO-CALLE-REG.

05	ETIQ-CORREO-CALLE-CODIGO	PIC X.
05	ETIQ-CORREO-CALLE-ID	PIC X(5).
05	ETIQ-CORREO-CALLE-NOMBRE	PIC X(30).
05	ETIQ-CORREO-CALLE-CALLE	PIC X(30).
05	FILLER	PIC X(10).

01 ETIQ-CORREO-ESTADO-REG.

05	ETIQ-CORREO-ESTADO-CODIGO	PIC X.
05	ETIQ-CORREO-ESTADO-ID	PIC X(5).
05	ETIQ-CORREO-ESTADO-TELEF	PIC X(8).
05	ETIQ-CORREO-ESTADO-CIUDAD	PIC X(30).
05	ETIQ-CORREO-ESTADO-ESTADO	PIC X(2).
05	ETIQ-CORREO-ESTADO-CD-POSTAL	PIC X(9).
05	FILLER	PIC X(21).

Hay dos niveles de prefijos en este ejemplo:

- (1) Cada nombre de registro y de campo comienza con "ETIQ-CORREO", que es una abreviatura del nombre del "FICHERO-ETIQUETAS-CORREO". Como consecuencia, cualquier dato que comience con "ETIQ-CORREO" se asocia con el "FICHERO-ETIQUETAS-CORREO". De este modo, se puede leer la división de procedimientos sin referencias continuas a la división de datos para saber a qué fichero o registro pertenece un campo.
- (2) El nombre de cada campo del registro ETIQ-CORREO-CALLE-REG contiene el prefijo secundario "CALLE" (por ejemplo, ETIQ-CORREO-CALLE-CODIGO se asocia con el FICHERO-ETIQUETAS-CORREO y en particular con el registro ETIQ-CORREO-CALLE-REG de dicho fichero). De igual modo, cada campo del registro ETIQ-CORREO-ESTADO-REG tiene el prefijo secundario "ESTADO".

**EJEMPLO 5.17** En el Ejemplo 5.16 hay dos niveles 01 con las descripciones correspondientes a los registros lógicos en el buffer; estas dos descripciones se representan en la Figura 5-6. (Cuando existe más de un nivel 01 en una FD, las descripciones de los registros lógicos redefinen implicitamente una a la otra.)

Obsérvese que las descripciones de *ambos* registros coinciden en la interpretación de los primeros 6 bytes del registro lógico. Como tanto ETIQ-CORREO-CALLE-CODIGO como ETIQ-CORREO-ESTADO-CODIGO se refieren a la misma posición de memoria (primer byte del registro lógico), estos dos nombres pueden considerarse sinónimos en el resto del programa. De igual manera, ETIQ-CORREO-CALLE-ID y ETIQ-CORREO-ESTADO-ID ambos se refieren a los bytes comprendidos entre el 2 y el 6 del registro lógico y, por tanto, el intercambio entre estos nombres en la división de procedimientos no afecta a la ejecución del programa. El primer byte del registro lógico (al que se puede referir mediante ETIQ-CORREO-CALLE-CODIGO o mediante el sinónimo ETIQ-CORREO-ESTADO-CODIGO) se puede usar para contener el código de registro que le identifica como un registro calle o un registro estado. (Véanse el Ejemplo 5.13 y el Problema 5.80.)

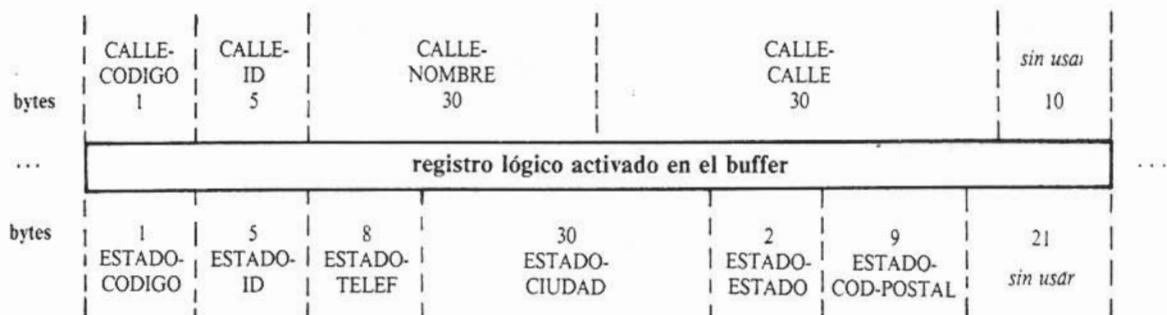


Fig. 5-6

### Buffers múltiples

Para un fichero secuencial, podrá ser ventajoso disponer de dos (o más) buffers: el contenido de uno estará sometido al procesamiento de la CPU al mismo tiempo que el otro se está cargando con el *siguiente* bloque desde el dispositivo de entrada/salida. En el COBOL de IBM OS/VS, el número de buffer por defecto para un fichero secuencial es dos; para un fichero con acceso al azar el número de buffers por defecto es uno. El COBOL ANS tiene una cláusula adicional en la instrucción SELECT que permite al programador especificar el número (entero) de buffers para un fichero:

```
SELECT ... ASSIGN TO ... [RESERVE entero [AREA ] ] ...  
[AREAS ] ...
```

A menos que conozca perfectamente la computadora y su estructura de entrada/salida, es preferible omitir la cláusula RESERVE y aceptar el número de buffer por defecto que designe el compilador.

### 5.9 SINTAXIS DE LA DESCRIPCION DE DATOS

El propósito de la descripción de los datos en la sección de ficheros es detallar por completo un campo del fichero. La Figura 5-7 indica la sintaxis general de la descripción de datos.

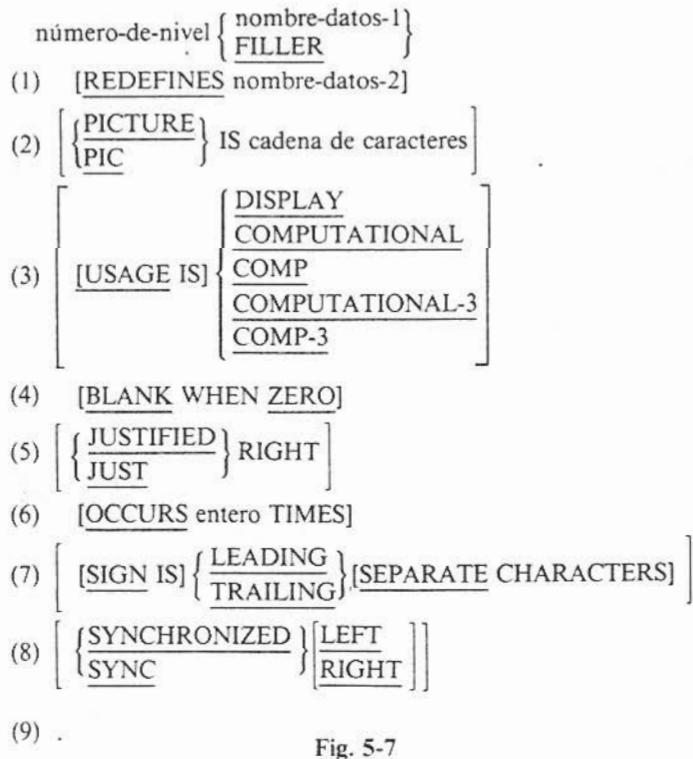


Fig. 5-7

El número de nivel siempre estará comprendido entre 01 y 49, 01 se reserva para los nombres de registro. "Nombre-datos-1" es el nombre definido por el programador para el campo tal y como se usará en la división de procedimientos cuando se le procese. Si un campo presente no va a ser procesado por el programa, se puede usar la palabra reservada "FILLER" en lugar del nombre del campo. "FILLER" se puede usar tanto como se desee.

#### EJEMPLO 5.18

```

FD FICHERO-DE-PAGOS
LABEL RECORDS ARE STANDARD.
01 REGISTRO-DE-PAGOS.
  05 NOMBRE-DE-CLIENTE          PIC X(20).
  05 IMPORTE-DEL-PAGO          PIC S9(5)V99.

FD INFORME-COBROS-SEMANALES
LABEL RECORDS ARE OMITTED.
01 REGISTRO-COBROS-SEMANALES.
  05 NOMBRE-DE-CLIENTE          PIC X(20).
  05 FILLER                     PIC X(5).
  05 IMPORTE-DEL-PAGO          PIC ZZ,ZZZ.99-.
  05 FILLER                     PIC X(65).

```

PROCEDURE DIVISION.

```

READ FICHERO-DE-PAGOS RECORD
MOVE NOMBRE-DE-CLIENTE OF REGISTRO-DE-PAGOS
  TO NOMBRE-DE-CLIENTE OF REGISTRO-COBROS-SEMANALES
MOVE IMPORTE-DEL-PAGO OF REGISTRO-DE-PAGOS
  TO IMPORTE-DEL-PAGO OF REGISTRO-COBROS-SEMANALES

```

Las reglas del lenguaje COBOL permiten duplicar los nombres "NOMBRE-DE-CLIENTE" e "IMPORTE-DEL-PAGO" que en el ejemplo aparecen en dos registros diferentes. Cuando se utiliza un nombre duplicado en la división de procedimientos debe *adjetivarse*; por ejemplo, debe estar subordinado a un único nombre de grupo (como "IMPORTE-PAGO OF REGISTRO-PAGO" y "NOMBRE-CLIENTE OF REGISTRO-RECIBOS-SEMANALES").

Se recomienda utilizar nombres de datos únicos con prefijos que sean abreviaturas de los nombres de registro o fichero (Ejemplo 5.16) para evitar tener que adjetivar en la división de procedimientos.

Si se utiliza la cláusula REDEFINES..., ésta debe aparecer a continuación del "número de nivel"; las otras cláusulas deben aparecer en orden. En las Secciones 5.10-5.17 se comentan las cláusulas en orden decreciente de su uso. (Excepto en lo relativo a REDEFINES..., este es el orden que se indica en la Fig. 5-7.)

#### 5.10 CLAUSULA PICTURE (Fig. 5-7, línea 2)

Esta cláusula describe el formato de los datos elementales; no se puede dar para un elemento compuesto. La *cadena de caracteres* puede contener los siguientes:

A B P S V X Z 0 9 / , . + - CR DB \* \$

La cadena de caracteres especifica (1) el número de posiciones de un carácter en el elemento simple; (2) el tipo de datos (numéricos, alfanuméricos, alfábéticos) que ocuparán las posiciones; y (3) si el elemento se editaría optimizando su aspecto al imprimirla en papel o en la pantalla.

La cláusula PICTURE es complicada porque ofrece muchas variaciones. Se comentará cada carácter de PICTURE individualmente. Recuerde: *cada elemento simple debe tener una cláusula PICTURE*.

A

Cada "A" en la "cadena de caracteres" representa una posición de carácter (byte) que puede contener sólo una letra o un espacio.

**EJEMPLO 5.19**

01 NOMBRE-EMPLEADO.	
05 APELLIDO-EMPLEADO	PIC A(15).
05 NOMBRE-EMPLEADO	PIC A(10).
05 INICIAL-SEG-NOMBRE	PIC A.

**Comentarios:** (1) El elemento compuesto NOMBRE-EMPLEADO no tiene cláusula PICTURE. (2) Todos los elementos simples tienen cláusulas PICTURE descriptivas. (3) El uso de un *factor de repetición* simplifica la escritura de PICTURE:

"A(10)" es una abreviatura de "AAAAAAA" (diez Aes)  
 "A(4)" es una abreviatura de "AAA" (cuatro Aes)

(4) APELLIDO-EMPLEADO ocupa 15 caracteres (bytes); NOMBRE-EMPLEADO ocupa 10 bytes; INICIAL-SEG-NOMBRE ocupa 1 byte. (5) Los tres elementos contendrán sólo letras del alfabeto o espacios. (6) Se utiliza "PIC" en vez de "PICTURE IS".

**B (Carácter de edición)**

Cada "B" de una cadena PICTURE representa un byte en el que se insertará un espacio en blanco cuando los datos se carguen en el campo. "B" es un ejemplo de *carácter de edición* que hace que los datos se formateen al cargarse en un campo con la cláusula PICTURE.

**EJEMPLO 5.20**

- 01 SALUDOS PIC AABA(5).

**Comentarios:** (1) SALUDOS es un campo de 8 bytes: 2 letras o espacios ("AA"), seguidos por un espacio ("B") y después 5 letras o espacios ["A(5)"]. (2) MOVE "HITHERE" TO SALUDOS producirá como resultado el siguiente contenido de SALUDOS:

"HI THERE"

La "B" hace que se inserte un espacio en blanco entre las dos primeras letras y las cinco últimas letras.

- 01 INICIALES-EMPLEADO.  
 05 INICIALES-COMPRIMIDAS PIC A(3).  
 05 INICIALES-SEPARADAS PIC ABABAB.

**Comentarios:** (1) No se puede repetir grupos de caracteres de PICTURE excepto escribiéndolos. ["AB(3)"] produciría "ABBB" y *no* "ABABAB". (2) Si INICIALES-COMPRIMIDAS contiene "DAF", entonces

MOVE INICIALES-COMPRIMIDAS TO  
 INICIALES-SEPARADAS

produciría "D A F" en INICIALES-SEPARADAS.

**X**

Este carácter es usado con mucha frecuencia y representa un byte que puede contener *cualquier* carácter del conjunto de caracteres de la computadora (en los sistemas IBM-370 caracteres EBCDIC). A menudo se usa X en la cláusula PICTURE que acompaña a un registro a nivel 01 que no se divide en campos.

**EJEMPLO 5.21**

05 ENTRADA-NUM-SEG-SOCIAL	PIC X(9).
.....	.....
05 IMPRESION-NUM-SEG-SOCIAL	PIC XXXBXXBXXXX.
.....	.....

ENTRADA-NUM-SEG-SOCIAL podría contener un valor como "111223333", correspondiente al número de la seguridad social de un trabajador, sin necesidad de espacios ni puntos. "PIC X(9)" prepara espacio para almacenar 9 caracteres cualesquiera. Cuando se vaya a imprimir el número de la seguridad social será preciso editarla para hacerlo más legible. Esto puede llevarse a cabo insertando espacios en blanco por medio del carácter "B", produciendo, por ejemplo, "111 22 3333".

En general, cuando se utilizan datos numéricos como identificativos y no para cálculos (por ejemplo, número de la seguridad social, números de teléfono, etc.), es preferible usar "PIC X".

9

Cada "9" en una cadena de PICTURE representa la posición de un dígito decimal (0 a 9). A diferencia de "A", "B" y "X", "9" no siempre requiere un byte (o posición de un carácter) de memoria. El tamaño real de un elemento numérico, que depende del código de datos, se tratará en la Sección 5.18. Aunque no es válido programar cálculos en COBOL con "números" en formato "PIC X", "PIC 9", sin embargo, está específicamente diseñado para realizar cálculos. "PIC 9" no produce ninguna edición de elementos numéricos; en particular, no inserta puntos ni elimina los *ceros a la izquierda*.

#### EJEMPLO 5.22

##### 01 REGISTRO-INFORMACION-EMPLEADO.

05 EMPLEADO-INFO-SEG-SOC	PIC X(9).
05 EMPLEADO-INFO-FAMILIARES	PIC 99.
05 EMPLEADO-INFO-DIAS-ENF	PIC 9(3).
05 EMPLEADO-INFO-DIAS-VACA	PIC 9(3).

##### 01 REGISTRO-CUENTA-DE-AHORRO.

05 CUENT-AHORRO-NUM-ING	PIC 9(5).
05 CUENT-AHORRO-REINTEGROS	PIC 9(5).
05 CUENT-AHORRO-NUMERO	PIC X(7).

##### 01 REGISTRO-MAESTRO-INVENTARIO.

05 MAESTRO-INV-EXISTENCIAS	PIC 9(7).
----------------------------	-----------

Los campos EMPLEADO-INFO-SEG-SOC y CUENT-AHORRO-NUMERO se usan como identificativos y no para cálculos, por eso se utiliza "PIC X". Todos los otros campos pueden tomar parte en cálculos y por eso se usa "PIC 9". Obsérvese que si se traslada 52 al campo MAESTRO-INV-EXISTENCIAS, se almacenará como "0000052".

S

El carácter "S" indica la presencia de un *signo de operación*. Un signo no ocupa espacio *adicional* para su almacenamiento, porque se incorpora internamente en la representación de los datos (véanse las Secciones 1.11 y 5.11). Los sistemas de codificación "complemento binario a dos" (COMP) y "decimal empaquetado" (COMP-3) integran el *signo en su representación*; por ello, en estos casos, omitir la "S" equivale a considerar exclusivamente el *valor absoluto*. La representación mediante DISPLAY (EBCDIC y ASCII) reserva automáticamente espacio para un signo como parte de la representación del dígito más a la derecha, pero el hecho de que se utilice o no este espacio depende exclusivamente de que se especifique una "S" en la cadena de PICTURE.

#### EJEMPLO 5.23

05 SEGUNDOS-ANTES-LANZAMIENTO PIC S9(5).

**Comentarios:**

- (1) Si se usa "S", debe ser el primer carácter de la cadena de PICTURE.
- (2) Cuando se imprimen números con signo en formato DISPLAY, el *dígito más a la derecha se imprime como una letra o un blanco*. Así, en EBCDIC (Tabla 5-1), si SEGUNDOS-ANTES-LANZAMIENTO contuviera 17 se imprimiría 0001G y si contuviera 12 imprimiría 0001K.
- (3) Los números con signo deben trasladarse a *campos de edición* e imprimir su versión editada para evitar dificultades de interpretación.

Tabla 5-1

Dígito más a la derecha	Impresión si lleva signo +	Impresión si lleva signo -
0	blanco	blanco
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

**EJEMPLO 5.24**

05 SALDO-TOTAL-DEUDA PIC 9(5).

Sin la "S" en la cláusula PICTURE de un dato numérico, la computadora siempre genera un signo + dando como resultado su valor absoluto. Con este problema, no sólo se requieren ciertas instrucciones en lenguaje máquina cada vez que se manipula el campo, sino que también puede ser peligroso. Por ejemplo, suponga que SALDO-TOTAL-DEUDA contiene 00015 para indicar que un determinado cliente debe 15 dólares. Si el cliente paga 20 dólares y se resta el valor 20 del contenido del campo SALDO-TOTAL-DEUDA, ocurriría lo siguiente: 00015 menos 00020 da -00005 que como no hay "S" se almacena como +00005. Todo indica que el cliente sigue debiendo 5 dólares cuando en realidad es acreedor de 5 dólares.

*A menos que se deseé expresamente el valor absoluto, es preferible añadir signo a los datos numéricos en la cláusula PICTURE.*

05 SALDO-TOTAL-DEUDA PIC S9(5).

V

El carácter "V" se usa para indicar la presencia de un punto decimal en un dato numérico. Se utilizó en conjunción con el carácter "9" y puede aparecer sólo una vez en la cadena PICTURE. Si se omite el carácter "V", se asume que el punto decimal sigue al dígito más a la derecha del número. Puesto que el punto decimal no forma parte de la representación interna de un número, no ocupa espacio en la memoria. Los puntos decimales se utilizan para alinear datos numéricos a efectos de cálculos y de edición.

**EJEMPLO 5.25** Suponiendo que los campos05 HORAS-TRABAJ-ESTA-SEMANA PIC S99V9.  
05 TOTAL-HORAS-AÑO PIC S9(4).

contienen +42,5 y +0603., respectivamente (una fracción de hora es importante en el tiempo semanal, pero no en el tiempo anual). Si la computadora tuviera que sumar las horas de la semana a las horas del año, debería indicarse dónde se supone que están los puntos decimales:

<i>Incorrecto</i>	<i>Correcto</i>
horas-semana = 425	horas-semana = 42,5
horas-año = 0603	horas-año = 0603,
SUMA <u>1028</u>	SUMA <u>0645,5</u>

Si el resultado correcto se tiene que almacenar en TOTAL-HORAS-AÑO, o bien la parte fraccionada (0.5) tiene que truncarse (despreciarse) o el número debe redondearse. Si se trunca 0645,5 a PIC S9(4), el resultado es +0645. El redondeo suma 1 si la parte fraccionaria es 0.5 o mayor. En este caso, si 0645,5 se redondea a PIC S9(4), el resultado es +0646.

**Z (Carácter de edición)**

El carácter "Z" se usa en lugar de "9" para representar la posición de un dígito decimal que se reemplaza por un espacio en blanco en el caso de que sea un cero a la izquierda. Los campos con

caracteres de edición (incluido "Z") no se pueden utilizar en cálculos, aunque si pueden alimentarse con el resultado de un cálculo.

#### EJEMPLO 5.26

- 05 PRECIO PIC ZZZZ9.

Si PRECIO contiene el valor doscientos tres y es parte de un registro de salida a la impresora, se imprimiría *bb203* (*b*=espacio en blanco). Los dos primeros caracteres "Z" de la cadena de PICTURE representan ceros a la izquierda y por tanto se llenan con espacios en blanco. El cuarto carácter "Z" también corresponde a un cero, pero no es un cero a la izquierda (no sólo le preceden ceros), por eso se imprime. (Si PRECIO contuviera cero, se imprimiría *bbbb0*, puesto que PIC "9" siempre representa un dígito sea cero o no).

- 05 PRECIO PIC ZZZZ.

Si PRECIO contuviera cero, en caso de impresión el resultado sería *bbbb*.

#### , (Punto, carácter de edición)

El carácter “.” (punto) representa la posición de un punto decimal *real* en un campo (en contraposición con “V”, que representa un supuesto punto decimal). El carácter “.” ocupa un byte de memoria en el campo y aparece cuando se imprime el campo en papel o en la pantalla. A “.” se le considera un carácter de edición; por tanto, los campos que contengan “.” en la cláusula PICTURE *no* pueden ser usados para cálculos. Más aún, “.” no pueden ser el último carácter de la cadena de PICTURE.

#### EJEMPLO 5.27

01 LINEA-FACTURA.		
05 NUM-ARTICULO-FACTURA	PIC X(5).	
05 FILLER	PIC X(8).	
05 DESC-ARTICULO-FACTURA	PIC X(20).	
05 FILLER	PIC X(5).	
05 CANTIDAD-ARTICULO-FACTURA	PIC ZZ9.	
05 FILLER	PIC X(3).	
05 PRECIO-UNIDAD-FACTURA	PIC ZZZ.99.	
05 FILLER	PIC X(10).	
05 PRECIO-TOTAL-FACTURA	PIC ZZZZZZ.99.	

Comentarios:

- (1) El elemento compuesto (nivel 01) no tiene cláusula PICTURE.
- (2) NUM-ARTICULO-FACTURA es un campo numérico que sirve como identificativo y por eso tiene “PIC X(5)”.
- (3) FILLER es una palabra reservada que se utiliza en lugar de un nombre de campo que no interviene en la división de procedimientos. Aquí se utiliza como nombre de un campo en blanco que sirve de separación entre campos reales que van a ser impresos en papel o en la pantalla CRT.
- (4) CANTIDAD-ARTICULO-FACTURA es un campo editado para su impresión. Es un campo de 3 dígitos (dos caracteres "Z" y un "9"), los dos primeros se convertirán en espacios en blanco si contienen ceros a la izquierda.
- (5) Los campos PRECIO-UNIDAD-FACTURA y PRECIO-TOTAL-FACTURA están editados para su impresión. Se anulan los ceros a la izquierda y se incluye un punto decimal. Observe el uso de "9" en lugar de "Z" después de “.” en la cláusula PIC, así se imprime siempre el punto decimal. Los caracteres de puntuación rodeados de "Z" en una cláusula PIC se convierten en espacios en blanco si ambas "Z" representan ceros a la izquierda.

#### , (Coma, carácter de edición)

El carácter “,” representa un byte de memoria en el que se situará el código del carácter coma. Al igual que “.”, una coma entre dos “Z” se convertirá en espacios en blanco si ambas “Z” representan ceros a la izquierda. La coma también se convierte en espacio en blanco *cuando sólo hay espacios en blanco a su izquierda*. Los campos numéricos con “,” en la cadena de PIC no pueden ser utilizados en cálculos (aunque pueden alimentarse del resultado de un cómputo).

**EJEMPLO 5.28**

<i>PICTURE</i>	<i>Valor</i>	<i>Impreso como:</i>
ZZ,ZZZ.99	00010.05	bbb10.05
ZZ,ZZZ.99	00123.45	bbb123.45
ZZ,ZZZ.99	01000.00	b1,000.00
ZZ,ZZZ.99	98765.43	98,765.43
ZZ,ZZZ.ZZ	00000.00	bbbbbbbbb
ZZ,ZZZ.ZZ	00000.03	bbbbbbb3
ZZ,ZZZ.99	00000.03	(falseado)
		bbbbbb0.3
		(versión correcta del anterior)

**\$ (Carácter de edición)**

El carácter \$ se puede usar para insertar un *signo de dólar fijo* inmediatamente a la izquierda de la primera posición correspondiente a un dígito.

**EJEMPLO 5.29**

<i>PICTURE</i>	<i>Valor</i>	<i>Impreso como:</i>
\$Z,ZZZ.99	0072.15	\$bbb72.15
\$Z,ZZZ.99	9876.54	\$9,876.54

El carácter "S" también puede significar un *signo flotante de dólar* que se imprime a la izquierda del primer carácter significativo (es decir, el primer dígito o el punto decimal). Para ello se necesita más de un "\$" en la cláusula PICTURE: el carácter "\$" más a la izquierda representa un lugar para el signo dólar, los restantes caracteres "\$" representan posiciones de dígitos en el campo.

**EJEMPLO 5.30**

	<i>PICTURE</i>	<i>Valor</i>	<i>Impreso como:</i>
(a)	\$\$,\$\$\$\$.99	0001,23	bbbb\$1.23
(b)	\$\$,\$\$\$\$.99	0845,79	bb\$845.79
(c)	\$\$,\$\$\$\$.99	9876,54	\$9,876.54
(d)	\$\$,\$\$\$\$.99	12345,67	\$2,345,67
(e)	\$\$,\$\$\$\$.99	-0000,05	bbbb\$,.05

**Comentarios:** (a)-(c) El signo dólar "flota" sobre la cifra más significativa del valor editado. (d) Hay sólo 6 posiciones de dígito en la cláusula PICTURE (los cuatro "\$" y los dos "9"). Por eso el séptimo dígito se trunca *por la izquierda*. (e) No está previsto el signo en la cláusula PICTURE, por ello se imprime el valor como si fuera positivo. Aquí el primer carácter significativo es el punto decimal y el signo "\$" flota sobre él.

Recuerde que el signo dólar flotante requiere un carácter "\$" más en la cadena de PICTURE que el número de posiciones de dígitos a editar. Los campos que utilicen "\$" en su cláusula PIC no pueden usarse en cálculos.

**CR, DB, -, + (Caracteres de control de edición del signo)**

Estos caracteres se usan principalmente en la edición de valores negativos.

**EJEMPLO 5.31** Los caracteres "CR" (o "DB") se pueden usar en el extremo derecho de la cadena de PICTURE de un elemento numérico para indicar un "abono" (crédito) o "débito" negativo.

	<i>PICTURE</i>	<i>Valor</i>	<i>Impreso como:</i>
(a)	\$Z,ZZZ.99CR	0010,56	\$bbb10.56bb
(b)	\$Z,ZZZ.99CR	-1435,72	\$1,435.72CR
(c)	\$\$,\$\$\$\$.99DB	-3476,52	\$3,476.52 DB

**Comentarios:** (a)-(b) Si el valor es negativo, se imprime "CR" (o "DB"); en otro caso se sustituyen por blancos. (c) Aquí el carácter "B" introduce un espacio en blanco entre el último dígito impreso y "DB" (o "CR").

Los otros dos caracteres de control de edición del signo, “+” y “-”, funcionan idénticamente excepto en un sentido: el carácter “+” siempre imprime un signo para el número (con independencia de que sea positivo o negativo), mientras que “-” imprime un signo menos sólo cuando el valor es negativo. Estos caracteres de PICTURE se pueden usar *al principio* (precediendo a la primera posición de un dígito) o *al final* (a continuación de la última posición de un dígito). También se usan en modos *flotante* y *fijo* al igual que “\$”.

#### EJEMPLO 5.32

	<i>PICTURE</i>	<i>Valor</i>	<i>Impreso como:</i>
(a)	\$\$,\$\$.99-	-2345.67	\$2,345.67-
(b)	\$\$,\$\$.99-	0023.78	bbb\$23.78b
(c)	ZZZ.9+	-763.7	763.7-
(d)	-ZZ.9	23.7	b23.7
(e)	-ZZ.9	00.7	-bb.7
(f)	+ZZ9	-005	-bb5
(g)	++,+++.9	0003.7	bbbb+3.7
(h)	--,-9	0000	bbbb0
(i)	--,-9	-0100	bb-100
(j)	++,++9	0000	bbb+0

**Comentarios:** (a) El signo menos por detrás se usa frecuentemente en los informes de negocios para hacer más visibles las cantidades negativas. (b) El signo menos por detrás se convierte en espacio en blanco cuando el valor es positivo. (c) El signo + por detrás imprime siempre el signo del valor. (d) El signo menos fijo por delante se convierte en espacio en blanco para valores positivos. (e) El signo menos fijo por delante siempre se imprime en la primera posición. (f) El signo más flotante se sitúa inmediatamente antes del primer carácter significativo. (g) El signo más flotante se sitúa antes del primer carácter significativo. (h) El signo menos flotante se imprime sólo si el valor es negativo; acabando PIC con “9” obliga a imprimir “0”. (i) El signo menos flotante se imprime cuando el valor es negativo. (j) Cero se considera positivo.

En los informes de negocios, el control del signo se suele realizar con signo “-” por detrás o con “CR” o “DB”. Como con los otros caracteres de edición, los campos con control del signo no se pueden usar en los cálculos.

#### \* (Asterisco, carácter de edición)

Como *protección en los cheques* se llenan las posiciones en blanco por delante con asteriscos, de forma que el importe no se pueda incrementar: \$\*\*\*23.54. Para la protección de cheques se utiliza el carácter “\*” en la cláusula PICTURE.

#### EJEMPLO 5.33

	<i>PICTURE</i>	<i>Valor</i>	<i>Impreso como:</i>
(a)	\$*,***,.99	12345.67	\$2,345,67
(b)	\$*,***,***.99	0001234.56	\$****1,234.56
(c)	\$*,***.**	0000.00	\$***** **

**Comentarios:** (a) Cuatro caracteres “\*” y dos “9” sólo pueden contener 6 dígitos; el truncamiento se produce por la izquierda. (b) Los ceros y comas por la izquierda se sustituyen por “\*”; los dígitos y comas significativos se mantienen. (c) El punto decimal siempre se imprime al utilizar la protección de cheques, incluso para un valor nulo.

#### 0 (Cero, carácter de edición)

El carácter “0” se usa para reservar una posición en un resultado editado, en la cual se insertará un “0”.

**EJEMPLO 5.34**

Un uso típico de "0" es para imprimir céntimos cuando el valor se almacena entero en la memoria de la computadora.

<i>PICTURE</i>	<i>Valor</i>	<i>Impreso como:</i>
\$ZZ,ZZZ.00	01205	\$b1,205.00
\$ZZ,ZZZ.00	00002	\$bbbbbb2.00
\$\$,\$\$\$\$.00	00002	bbbbbb\$2.00

/ (Barra, carácter de edición)

El carácter "/" se usa para reservar un byte en el resultado editado que contendrá siempre un "/". Este carácter es sumamente útil al editar números que representan fechas (por ejemplo, 231083 se editaría como 23/10/83 con PIC 99/99/99).

**P**

Una cadena de caracteres "P" se utiliza para representar la posición de un punto decimal *asumido* cuando el lugar cae fuera del rango de dígitos almacenados en la memoria. "P" no es un carácter de edición y se usa para definir campos numéricos que van a usarse en cálculos. "P" no ocupa espacio adicional de almacenamiento para el campo, simplemente indica la posición de un punto decimal asumido. "P" puede aparecer sólo en los extremos derecho o izquierdo de un campo numérico. Los únicos caracteres que pueden preceder a "P" por la izquierda son "S" y "V"; el único carácter que puede seguir a "P" por la derecha es "V".

**EJEMPLO 5.35**

	<i>PICTURE</i>	<i>Valor en memoria</i>	<i>Valor usado en los cálculos</i>
(a)	PP99	12	.0012
(b)	99PP	12	1200
(c)	PP99PP	—	—
(d)	SPPP999	735	+ .000735
(e)	SVPPP999	735	+ .000735
(f)	SPP999	1234	+ .00234
(g)	S9(3)P(4)	538	+5380000
(h)	SP99	03	+ .003

**Comentarios:** (a), (b), (g) El punto decimal asumido está tantos lugares a la izquierda del primer dígito (a la derecha del último dígito) como caracteres "P" haya a la izquierda (derecha) del campo. (c) *Incorrecto:* "P" no puede aparecer en ambos extremos. (d) "S" puede preceder a "P" por la izquierda. (e) "V" es redundante en este caso porque el punto decimal asumido ya está fijado inmediatamente a la izquierda del carácter "P" situado más a la izquierda. (f) Si el valor en memoria tiene más dígitos que caracteres "9" hay en la cadena de PICTURE, el valor se trunca por la izquierda y el punto decimal en el valor truncado aparece donde corresponde de acuerdo con los caracteres "P". (h) A los efectos del funcionamiento del carácter "P" no tiene importancia que el dígito más a la izquierda sea un cero.

**5.11 CLAUSULA USAGE (Figura 5-7, línea 3)**

En esta cláusula se especifica la representación interna para un elemento de datos; con frecuencia se omite dado que el valor DISPLAY que toma por defecto es apropiado en la mayor parte de los casos.

**USAGE IS DISPLAY**

Con DISPLAY un dato se representa en el sistema de códigos de caracteres propio de la computadora (EBCDIC o ASCII), en el que cada carácter alfanumérico queda reducido a un código binario que ocupa un byte de memoria. DISPLAY es obligatorio cuando el elemento de datos es: (1) alfanumérico (PIC X); (2) alfabético (PIC A); (3) cualquier campo en un fichero de tarjetas (de entrada o de salida); (4) cualquier campo de un fichero con destino a la impresora; (5) cualquier

Puesto que los tres datos están en formato DISPLAY, el compilador se da cuenta de que tiene que convertirlos a formato COMP (o COMP-3) para efectuar cálculos. El resultado se volverá a convertir a formato DISPLAY. Una traducción posible al lenguaje máquina de la instrucción SUBTRACT sería:

- (1) Convierte PRECIO-ORIGINAL a formato COMP y coloca el resultado provisionalmente en el lugar de memoria T1.
- (2) Convierte DESCUENTO a formato COMP y coloca el resultado provisionalmente en el lugar de memoria T2.
- (3) Resta el contenido de la posición de memoria T2 del contenido de la posición de memoria T1; sitúa el resultado en la posición de memoria T3.
- (4) Convierte el contenido de la posición de memoria T3 a formato DISPLAY y coloca el resultado en PRECIO-NETO.

Si los tres datos estuvieran en formato COMP, la traducción al lenguaje máquina hubiera sido más económica:

- (1) Resta el contenido de DESCUENTO del contenido de PRECIO-ORIGINAL; coloca el resultado en PRECIO-NETO.

### USAGE IS COMPUTATIONAL-3 (abreviado COMP-3)

COMP-3 es una extensión del COBOL ANS disponible en algunos pero no en todos los sistemas. COMP-3 es similar a COMP en el sentido de que la CPU puede realizar operaciones aritméticas con ambos tipos de números; igualmente, ambos formatos ocupan menos memoria que la que ocuparía el mismo dato numérico en formato DISPLAY. Si se dispone de COMP-3, probablemente es más adecuado que COMP para los datos numéricos en los cálculos que se realizan en el mundo de los negocios (consulte el manual de su sistema). *Excepción:* No utilice COMP-3 para subíndices (véase el Capítulo 10).

**EJEMPLO 5.39** En el Ejemplo 5.38; sustituir "PIC S9(3)V99." por "PIC S9(3)V99 COMP-3." produciría el programa objeto más eficiente en una computadora del tipo IBM-370. El uso del formato COMP para los tres datos ocuparía el segundo lugar en términos de eficiencia; el formato DISPLAY genera (como en el Ejemplo 5.38) la traducción menos eficiente.

### 5.12 CLAUSULA BLANK (Figura 5-7, línea 4)

La frase BLANK WHEN ZERO hace que un elemento se rellene con espacios en blanco cuando vaya a tomar el valor cero. La cláusula sólo se puede utilizar con elementos numéricos que no requieran protección de cheque. Un dato con la cláusula BLANK WHEN ZERO se considera automáticamente editado, por lo que su USAGE debe ser DISPLAY.

### EJEMPLO 5.40

01 LINEA-CUENTAS-DE-CLIENTES.		
05 NOMBRE-DE-CLIENTE	PIC X(30).	
05 FILLER	PIC X(5).	
05 TIPO-DE-CLIENTE	PIC XBX.	
05 FILLER	PIC X(4).	
05 DIAS-DESDE-VENCIMIENTO	PIC ZZ9	BLANK WHEN ZERO.
05 FILLER	PIC X(7).	
05 IMPORTE-DEBIDO	PIC \$\$\$\$\$.99	BLANK WHEN ZERO.

Si DIAS-DESDE-VENCIMIENTO tomara el valor cero, se imprimiría como *bbb* (en lugar de *bb0*). Si IMPORTE-DEBIDO tomara el valor cero, se imprimiría como *bbbbbbbb* (en lugar de *bbbb\$0.00*). El resultado es que en el informe sólo se imprimen valores no nulos, concentrando la atención del usuario en aquellos clientes que no han efectuado sus pagos cuando les correspondía.

**EJEMPLO 5.41** Un informe producido por el Ejemplo 5.40, sin la cláusula BLANK WHEN ZERO, podría tener el siguiente aspecto:

NOMBRE	TIPO DE PAGO	DIAS PASADOS	IMPORTE
FRANK	A 7	0	\$00
ACE	A 7	0	\$00
CARTER	B 3	5	\$25.32
SMITH	A 7	0	\$00
PERELMAN	B 5	63	\$587.00
CONVERSE	A 7	0	\$00
GETZ	B 3	0	\$00

Obsérvese cómo se requiere cierta atención para no perderse en el laberinto de ceros.

### 5.13 CLAUSULA JUSTIFIED (Figura 5-7, línea 5)

La cláusula JUSTIFIED sólo puede usarse con datos alfanuméricos (PIC X) o alfabeticos (PIC A). Por defecto los datos alfanuméricos y alfabeticos se *justifican por la izquierda* del campo; al incluir la cláusula se *justifican por la derecha*. Esta cláusula se utiliza raramente.

#### EJEMPLO 5.42

```
05 CAMPO-A    PIC X(5).
05 CAMPO-B    PIC X(5)  JUSTIFIED RIGHT.
```

Supongamos que CAMPO-A y CAMPO-B toman el valor "CAT". Entonces:

<i>Campo</i>	<i>Impreso como:</i>
CAMPO-A	CATbb
CAMPO-B	bbCAT

#### EJEMPLO 5.43

```
05 CAMPO-A    PIC X(3).
05 CAMPO-B    PIC X(3)  JUSTIFIED RIGHT.
```

Supongamos que CAMPO-A y CAMPO-B toman el valor "SONRIE". Entonces:

<i>Campo</i>	<i>Impreso como:</i>
(a) CAMPO-A	SON
(b) CAMPO-B	RIE

**Comentarios:** (a) Si el valor es demasiado largo para el campo, los elementos PIC X y PIC A se truncan por el mismo lado que se rellenan si fueran cortos. Como la justificación por defecto se produce a la izquierda, el truncamiento se produce por la derecha y el campo se rellena con los tres caracteres más a la izquierda. (b) La cláusula JUSTIFIED obliga a la justificación por la derecha. En este caso el campo PIC X(3) se rellena con los 3 caracteres más a la derecha del valor.

### 5.14 CLAUSULA OCCURS (Figura 5-7, línea 6)

La cláusula OCCURS permite al programador crear una estructura conocida como *tabla* o *matriz* (array); la manipulación de tablas en COBOL se comenzará en el Capítulo 10.

#### EJEMPLO 5.44

```
01 REGISTRO-DESCRIPCION-ARTICULO.
  05 NUMERO-ARTICULO          PIC X(7).
  05 ALMACEN-ARTICULO         PIC X(2).
  05 EXISTENCIAS-ARTICULO     PIC S9(5)      COMP-3.
  05 PROVEEDORES-ARTICULO    PIC X(20).    OCCURS 10 TIMES.
    10 NOMBRE-PROVEEDOR-ARTICULO  PIC X(5).
    10 ID-PROVEEDOR-ARTICULO    PIC S9(5)V99  COMP-3.
    10 COSTE-PROVEEDOR-ARTICULO PIC S9(5)V99  COMP-3.
  05 PRECIO-ARTICULO          PIC S9(5)V99  COMP-3.
```

El elemento compuesto PROVEEDORES-ARTICULO se repite 10 veces, constituyendo una *tabla*. En la división de procedimientos, el programador tendrá que utilizar *subíndices* o *índices* para indicar la línea de la tabla que tiene que procesarse:

MOVE 'CAMANYO Y CIA' TO NOMBRE-PROVEEDOR-ARTICULO (QUE-PROVEEDOR)

QUE-PROVEEDOR es un dato numérico comprendido entre 1 y 10, que indica a cuál de las 10 líneas de la tabla se refiere. Estos subíndices deben tener formato (USAGE) COMP:

05 QUE-PROVEEDOR PIC S9(2) COMP.

### 5.15 CLAUSULA SIGN (Figura 5-7, línea 7)

La cláusula SIGN sólo puede usarse con datos numéricos que tengan una "S" en la cláusula PICTURE y estén en formato DISPLAY. Existen las siguientes opciones:

- SIGN IS TRAILING

El signo forma parte de la representación interna del dígito más a la derecha y no precisa espacio adicional en la memoria. Este es el valor por defecto en el COBOL del sistema IBM OS/VS.

- SIGN IS LEADING

El signo forma parte de la representación interna del dígito más a la izquierda. No precisa espacio adicional en la memoria.

- SIGN IS TRAILING SEPARATE CHARACTER

A la representación interna del número se le añade por la derecha el código EBCDIC (o ASCII) correspondiente a "+" o "-". En este supuesto, el carácter "S" de la cláusula PICTURE *representa* un byte extra de almacenamiento. Si el signo no estuviera presente en un dato de este tipo, el programa fallaría.

- SIGN IS LEADING SEPARATE CHARACTER

A la representación interna del número se le añade por la izquierda el código EBCDIC (o ASCII) correspondiente a "+" o "-". Aquí también el carácter "S" de la cláusula PICTURE *representa* un byte extra de almacenamiento. Si el signo no estuviera presente en un dato de este tipo, el programa fallaría.

#### EJEMPLO 5.45 Dado

ELEMENTO-DE-DATOS PIC S9(3)

con un valor +123, la Figura 5-8 muestra la representación interna de ELEMENTO-DE-DATOS para las diversas posibilidades de la cláusula SIGN. Cada cuadrado representa un byte de memoria.

SIGN IS TRAILING:	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>+</td></tr></table>	1	2	3	+	(3 bytes)
1	2	3	+			
SIGN IS LEADING:	<table border="1"><tr><td>+</td><td>1</td><td>2</td><td>3</td></tr></table>	+	1	2	3	(3 bytes)
+	1	2	3			
SIGN IS TRAILING SEPARATE CHARACTER:	<table border="1"><tr><td>1</td><td>2</td><td>3</td><td>+</td></tr></table>	1	2	3	+	(4 bytes)
1	2	3	+			
SIGN IS LEADING SEPARATE CHARACTER:	<table border="1"><tr><td>+</td><td>1</td><td>2</td><td>3</td></tr></table>	+	1	2	3	(4 bytes)
+	1	2	3			

Fig. 5-8

Las opciones "SEPARATE CHARACTER" y "LEADING" producirán un programa objeto menos eficiente para las computadoras del tipo IBM 370. En general es preferible la omisión de la cláusula "SIGN..." y limitarse a la representación por defecto que realice un sistema concreto.

### 5.16 CLAUSULA SYNCHRONIZED (Figura 5-7, línea 8)

En algunas computadoras (como las IBM 370) los datos numéricos en formato COMP se pueden manipular de forma más eficiente si se *alinean* las posiciones de memoria asignadas en unos *límites* apropiados. Por ejemplo, un elemento COMP de 2 bytes se puede procesar más fácilmente si la dirección del primero de los 2 bytes es divisible por 2; un campo en formato COMP de 4 bytes se puede procesar más eficientemente si empieza en un byte cuya dirección es divisible por 4.

La cláusula SYNCHRONIZED (abreviada SYNC) hace que el compilador asigne una posición de memoria a un elemento en formato COMP alineada con el límite que optimiza la eficiencia. Esto puede implicar la necesidad de insertar *bytes de holgura* entre el campo anterior en el registro y el campo que se está alineando; dicha inserción se realiza automáticamente por el compilador.

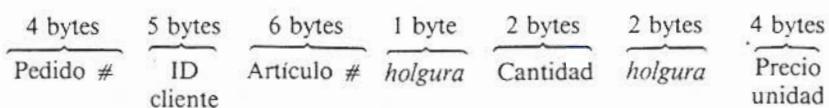
**EJEMPLO 5.46** Dada la siguiente descripción de registro

*Longitud en bytes*

01 REGISTRO-PEDIDO.

4	05 NUMERO-PEDIDO	PIC X(4).
5	05 ID-CLIENTE-PEDIDO	PIC X(5).
6	05 NUM-ARTICULO-PEDIDO	PIC X(6).
2	05 CANTIDAD-PEDIDO	PIC S9(4)      COMP SYNC.
4	05 PRECIO-UNIDAD-PEDIDO	PIC S9(5)V99    COMP SYNC.

la distribución en memoria de REGISTRO-PEDIDO será la siguiente (el byte más a la izquierda tiene dirección cero):



Dado que los bytes de holgura ocupan espacio y pueden conducir a errores en la determinación de la longitud del registro, se recomienda el uso de COMP SYNC sólo para subíndices en el sistema IBM 370.

**EJEMPLO 5.47** En el Ejemplo 5.44, QUE-PROVEEDOR estaría mejor definido así:

05 QUE-PROVEEDOR    PIC S9(2)    COMP SYNC.

Es preciso decir que en el COBOL del sistema IBM OS/VS se ignora por el compilador el adjetivo "LEFT" o "RIGHT" en la cláusula SYNCHRONIZED.

### 5.17 CLAUSULA REDEFINES (Figura 5-7, línea 1)

La cláusula REDEFINES se utiliza para dar una descripción alternativa de un elemento simple o compuesto. Cuando se utiliza REDEFINES... debe ser la primera cláusula que siga al número de nivel y nombre del dato:

número-de-nivel    nombre-del-dato-1    REDEFINES nombre-de-dato-2

El "nombre-de-dato-1" es un nombre alternativo y el "nombre-de-dato-2" es el nombre anterior. El número de nivel del nombre-de-dato-1 debe ser el mismo que el correspondiente a nombre-de-datos-2

**EJEMPLO 5.48**

```
FD FICHERO-MAESTRO-DE-CLIENTES
RECORD CONTAINS 105 CHARACTERS
LABEL RECORDS ARE STANDARD
```

## 01 REGISTRO-MAESTRO-CLIENTES.

05	ID-CLIENTE	PIC X(6).
05	NOMBRE-CLIENTE	PIC X(30).
05	DIRECCION-CLIENTE	PIC X(30).
05	DATOS-CLIENTE-HABITUAL.	
10	TIPO-FACTURA-CLIENTE	PIC XX.
10	LIMITE-CREDITO-CLIENTE	PIC S9(5)V99 COMP-3.
10	LIMITE-PESO-CLIENTE	PIC S9(3)V9 COMP-3.
05	DATOS-CLIENTE-PREFERENTE	
	REDEFINES DATOS-CLIENTE-HABITUAL.	
10	TIPO-DE-DESCUENTO	PIC SV99 COMP-3.
10	CODIGO-POBLACION	PIC X(3).
10	MODO-DE-ENVIO	PIC X(3).
	10 TIPO-FACTURA.	PIC X.
05	DIRECCION-CLIENTE.	
10	CIUDAD	PIC X(19).
10	ESTADO	PIC X(2).
	10 CODIGO-POSTAL	PIC X(9).

Si el fichero está almacenado en disco, se pueden utilizar los formatos COMP-3 o COMP para los datos numéricos que se utilicen en cálculos. Así no sólo se ahorra espacio del disco, sino que el programa objeto será más eficiente. La cláusula REDEFINES hace que DATOS-CLIENTE-HABITUAL y DATOS-CLIENTE-PREFERENTE ocupen exactamente las mismas posiciones de memoria (como esto forma parte de la FD, estas posiciones son parte del registro lógico activado en el buffer). Así hay un área de 9 bytes del registro lógico en activo que puede tener dos interpretaciones diferentes (Fig. 5-9). Depende del programador trabajar con los nombres de datos que casan con el contenido real de la memoria en un instante determinado durante la ejecución del programa. Por ejemplo, los clientes preferentes podrían tener todos identificativos que acabasen en cero. Después de leer un REGISTRO-MAESTRO-CLIENTES, el programa podría comprobar si el último carácter del campo ID-CLIENTE es un cero. Si no fuera cero, se procesaría DATOS-CLIENTE-HABITUAL, en caso contrario se procesaría DATOS-CLIENTE-PREFERENTE.

## DATOS-CLIENTE-PREFERENTE

Tipo de descuento	Código de población	Modo de envío	Tipo de factura
Pic SV99 COMP-3 (2 bytes)	Pic X(3) (3 bytes)	Pic X(3) (3 bytes)	Pic X (1 byte)
...	parte del registro lógico activo en el buffer	...	

Tipo de factura	Límite del crédito	Límite de peso
PIC XX (2 bytes)	PIC S9(5)V99 COMP-3 (4 bytes)	PIC S9(3)V9 COMP-3 (3 bytes)

## DATOS-CLIENTE-HABITUAL

Fig. 5-9

Obsérvese que hay total libertad para usar REDEFINE con un área que contenga cláusulas PICTURE, USAGE, etc. La única restricción es que *las longitudes del elemento redefinido y del elemento redefinidor sean iguales*. Obsérvese también que REDEFINES debe tenerse en cuenta para calcular la longitud de un registro: la longitud del REGISTRO-MAESTRO-CLIENTES es 105 y no 114 bytes.

**EJEMPLO 5.49** En el programa del Ejemplo 5.16, la instrucción

01 ETIQ-CORREO-ESTADO-REG  
REDEFINES ETIQ-CORREO-CALLE-REG.

causará un error sintáctico (y sería innecesario puesto que los dos elementos del nivel 01 se *redefinen implicitamente* uno a otro). Los elementos del nivel 01 en la sección de ficheros *no* pueden ser REDEFINED. Los elementos de nivel 01 pueden, sin embargo, ser REDEFINED en el WORKING-STORAGE.

**EJEMPLO 5.50** A menudo es útil al comprobar campos de entrada (*validación de entradas*) poder referirse a un campo numérico como si fuera PIC X o PIC 9. La cláusula REDEFINES... permite hacerlo de ambos modos:

```
.....  
05 CAMPO-ENTRADA-NUMERICO          PIC S9(3)V99.  
05 CAMPO-ENTRADA-NUMERICO-X  
    REDEFINES CAMPO-ENTRADANUMERICO PIC X(5).
```

## 5.18 DETERMINACION DE LA LONGITUD DE UN ELEMENTO DE DATOS

La cantidad de memoria requerida para almacenar un dato elemental se determina teniendo en cuenta las cláusulas PICTURE y USAGE. La longitud de un elemento compuesto es la suma de las longitudes de los elementos simples que lo conforman, por lo que es suficiente saber determinar la longitud de un dato elemental.

**DISPLAY USAGE.** Cada carácter de la cadena PICTURE (a excepción de "P", "S" y "V") requiere un byte de almacenamiento; "P", "S" y "V" no utilizan espacio.

**COMP USAGE.** Como los datos COMP no se pueden editar, los únicos caracteres permitidos son "S", "V", "9" y "P". Como sucedía con los datos en formato DISPLAY, "S", "V" y "P" no consumen espacio de memoria. Desgraciadamente, el número de "9" en la cadena PICTURE de un dato COMP no determina por sí mismo la longitud. Para los sistemas del tipo IBM 370:

<i>Número de "9" en PIC</i>	<i>Longitud del elemento en bytes</i>
1 a 4	2
5 a 9	4
10 a 18	8

**COMP-3 USAGE.** Los únicos caracteres permitidos son "S", "V", "9" y "P". Al igual que con COMP, la longitud de un dato en formato COMP-3 depende del número de dígitos del dato (número de "9" en la cadena de PICTURE). Si hubiera  $k$  caracteres "9" ( $1 \leq k \leq 18$ ) en la cadena PICTURE de un dato COMP-3, el número de bytes ocupados en un sistema IBM 370 sería

$$[k/2] + 1$$

donde  $[k/2]$  es el mayor entero menor o igual a  $k/2$  (las fracciones se truncan).

## EJEMPLO 5.51

```
01 REGISTRO-INVENTARIO-DISCO.  
05 NUM-ARTICULO-INVENTARIO      PIC X(8).  
05 DESCRIPCION-INVENTARIO      PIC X(30).  
05 EXISTENCIAS-INVENTARIO      PIC S9(5)      COMP.  
05 PEND-RECIBIR-INVENTARIO      PIC S9(4)      COMP.  
05 DESCUENTO-INVENTARIO        PIC S9(3)V99   COMP.  
05 PRECIO-VENTA-INVENTARIO     PIC S9(5)V99   COMP.  
05 FECHA-PEDIDO-INVENTARIO.  
    10 MES-PEDIDO-INVENTARIO     PIC XX.  
    10 DIA-PEDIDO-INVENTARIO     PIC XX.  
    10 AÑO-PEDIDO-INVENTARIO     PIC XX.  
05 PORC-PERDIDA-INVENTARIO    PIC SVP9      COMP.
```

En un sistema del tipo IBM 370, se tendría:

<i>Nombre del dato</i>	<i>Longitud en bytes</i>	<i>Comentario</i>
REGISTRO-INVENTARIO-DISCO	60	la longitud del registro es la suma de las longitudes de los campos
NUM-ARTICULO-INVENTARIO	8	un byte por cada PIC X
DESCRIPCION-INVENTARIO	30	un byte por cada PIC X
EXISTENCIAS-INVENTARIO	4	5 a 9 dígitos COMP: 4 bytes
PEND-RECIBIR-INVENTARIO	2	1 a 4 dígitos COMP: 2 bytes
DESCUENTO-INVENTARIO	4	5 a 9 dígitos COMP: 4 bytes
PRECIO-VENTA-INVENTARIO	4	5 a 9 dígitos COMP: 4 bytes
FECHA-PEDIDO-INVENTARIO	6	el campo longitud es la suma de los subcampos longitud.
MES-PEDIDO-INVENTARIO	2	un byte por cada PIC X
DIA-PEDIDO-INVENTARIO	2	un byte por cada PIC X
AÑO-PEDIDO-INVENTARIO	2	un byte por cada PIC X
PORC-PERDIDA-INVENARIO	2	1 a 4 dígitos COMP: 2 bytes (no sólo un 9 en PIC)

Con COMP-3 en lugar de COMP, tendrían lugar los siguientes cambios en las longitudes de los campos:

<i>Nombre del dato</i>	<i>Longitud en bytes</i>	<i>Comentario</i>
REGISTRO-INVENTARIO-DISCO	58	se ahorran 2 bytes utilizando COMP-3
EXISTENCIAS-INVENTARIO	[5/2] + 1 = 3	ahorra un byte con relación a COMP; hay cinco "9" en PICTURE
PEND-RECIBIR-INVENTARIO	[4/2] + 1 = 3	toma un byte adicional COMP; hay cuatro "9" en PICTURE
DESCUENTO-INVENTARIO	[5/2] + 1 = 3	ahorra un byte con relación a COMP
PRECIO-VENTA-INVENTARIO	[7/2] + 1 = 4	igual que COMP
PORC-PERDIDA-INVENTARIO	[1/2] + 1 = 4	ahorra un byte con relación a COMP

## 5.19 SECCION DE ALMACENAMIENTO DE TRABAJO (WORKING-STORAGE SECTION)

La sección de almacenamiento de trabajo de la división de datos permite que el programador defina áreas de almacenamiento de datos en memoria principal que requiere el programa y que no son parte de ningún registro (véase el Problema 2.21). La descripción de los datos en la WORKING-STORAGE SECTION es bastante similar a la de la sección de ficheros, con las siguientes diferencias:

- (1) Como los campos del WORKING-STORAGE no forman parte de los registros lógicos, no hay entradas como las de FD en la sección de almacenamiento de trabajo.
- (2) Hay un nuevo número de nivel (77) para su uso al definir campos del almacenamiento de trabajo que no son parte de un registro.
- (3) También existe una nueva cláusula de descripción de datos ("VALUE..."), que sirve para dar, a un elemento del almacenamiento temporal, el valor que debe tomar al principio de la ejecución del programa.

### Nivel 77

Mientras que todos los campos de la sección de ficheros (FILE SECTION) son registros lógicos o parte de registros lógicos, muchos campos de la sección de almacenamiento de trabajo son *datos independientes* sin relación con otros campos del WORKING-STORAGE. A los datos independientes se les puede dar a cada uno el nivel 77 y colocarlos al principio de la sección de almacenamiento de trabajo. Sin embargo, se recomienda que siempre que sea posible, los datos elementales se agrupen formando registros en el WORKING-STORAGE con entradas descriptivas al nivel 01 (al igual que en la sección de ficheros). De este modo se facilita la depuración y mantenimiento del programa.

**EJEMPLO 5.52** Compare las dos versiones siguientes de la sección de almacenamiento de trabajo:

•	77 NUM-CHEQUES-IMPRESOS	PIC S9(5)	COMP-3.
	77 TRAB-HORAS-EXTRAS	PIC S9(5)	COMP-3.
	77 INDICE-TABLA-IMPUESTOS	PIC S9(4)	COMP SYNC.
	77 TODOS-REG-PROCESADOS	PIC X.	
	77 INDICE-TABLA-DEDUCCIONES	PIC S9(4)	COMP SYNC.
	77 PERTENENCIA-SINDICATO	PIC X.	
	01 LINEA-INFORME-NOMINA.		
	05 ID-EMPLEADO-NOMINA	PIC X(5).	
	05 FILLER	PIC X(3)	VALUE SPACES.
	05 NOMBRE-EMPLEADO-NOMINA	PIC X(20).	
	05 FILLER	PIC X(4)	VALUE SPACES.
	05 SALARIO-BRUTO-NOMINA	PIC \$ \$\$\$.99.	
•	01 CONTADORES-DE-TRABAJO.		
	05 NUM-CHEQUES-IMPRESOS	PIC S9(5)	COMP-3.
	05 TRAB-HORAS-EXTRAS	PIC S9(5)	COMP-3.
	01 INTERRUPTORES-PROGRAMA.		
	05 TODOS-REG-PROCESADOS	PIC X.	
	05 PERTENENCIA-SINDICATO	PIC X.	
	01 INDICES-TABLAS.		
	05 INDICE-TABLA-IMPUESTOS	PIC S9(4)	COMP SYNC.
	05 INDICE-TABLA-DEDUCCIONES	PIC S9(4)	COMP SYNC.
	01 LINEA-INFORME-NOMINA.		
	05 ID-EMPLEADO-NOMINA	PIC X(5).	
	05 FILLER	PIC X(3)	VALUE SPACES.
	05 NOMBRE-EMPLEADO-NOMINA	PIC X(20).	
	05 FILLER	PIC X(4)	VALUE SPACES.
	05 SALARIO-BRUTO-NOMINA	PIC \$ \$\$\$.99.	

En la segunda versión, los seis elementos de nivel 77 han sido agrupados en tres clases: contadores, interruptores y subíndices. Cada grupo de datos se ha tratado como un conjunto de campos en nivel 01. El nombre de dato a nivel 01 es descriptivo de la clase de datos. La segunda versión está mucho mejor organizada que la primera.

**EJEMPLO 5.53** Aunque la cláusula PICTURE es inválida para un elemento compuesto, USAGE... y SYNCHRONIZED... sí se pueden utilizar con esos datos, en cuyo caso se aplican a todos los elementos del grupo. Recomendamos que esta práctica se utilice en muy raras ocasiones, porque dificultará a otros programadores durante el mantenimiento del programa el que puedan reconocer el USAGE para estos datos. Un lugar relativamente "apropiado" para el uso de USAGE con elementos compuestos es con aquellos que de otro modo estarían en el nivel 77 del WORKING-STORAGE (por ejemplo, en el Ejemplo 5.52, "01 CONTADORES-DE-TRABAJO COMP-3").

#### Cláusula VALUE

Cuando se utiliza la cláusula

[ VALUE IS {literal  
constante-figurativa} ]

es para indicar el valor de un elemento de la sección de almacenamiento de trabajo (WORKING-STORAGE) al principio de la ejecución del programa. Si no se utiliza la cláusula VALUE, es impredecible el contenido inicial de estos datos; los programadores en estos casos dicen que contienen *basura* o que están *indefinidos*. Muchas veces, el contenido inicial de un dato no tiene importancia porque la lógica del programa inyecta el valor preciso (borrando la basura) antes de hacer uso de su contenido.

Hay dos tipos de literales y seis tipos de constantes figurativas:

**Literales no numéricos.** Un literal no numérico es una cadena de caracteres entre comillas (Sección 2.3). Si se desea representar unas comillas en un literal no numérico, se utilizan dos dobles comillas seguidas y sólo una se considera parte del literal.

#### EJEMPLO 5.54

05 NOMBRE-CLIENTE PIC X(15) VALUE ""KLIK"" CLEAN".

especifica el valor "KLIK" CLEAN (cada *pareja* de dobles comillas dentro del literal representa unas dobles comillas en el valor).

Algunos compiladores utilizan la comilla simple (o el apóstrofo) en lugar de las dobles comillas. La norma ANS 74 especifica el uso de las dobles comillas, pero el programador tiene que conformarse con el sistema del que disponga:

05 NOMBRE-CLIENTE PIC X(15) VALUE 'PERELMAN' 'S PETS'.

**Literales numéricos.** Se componen de dígitos entre 0 y 9, signo más (+), signo (-) y el punto decimal. El máximo valor de un literal numérico depende de la computadora; en el COBOL del sistema IBM OS/VS es de  $10^{18}-1$  (18 dígitos). Si no se especifica signo en un literal numérico, se asume que es positivo. Si un literal numérico no tiene punto decimal, se supone que es *entero*.

**Constantes figurativas.** Una constante figurativa es una palabra reservada que representa un valor constante y pueden usarse en cualquier parte de un programa en lugar del literal correspondiente. La Tabla 5-2 muestra las constantes figurativas del COBOL de IMB OS/VS.

Tabla 5-2

Palabra reservada	Tipo del literal al que reemplaza	Valor
ZERO ZEROS (or ZEROES)	numérico	0 (repetido un adecuado número de veces; el compilador se ajusta para cero en los formatos DISPLAY o COMP)
SPACE SPACES	no numérico	un espacio en blanco (apropiadamente repetido)
HIGH-VALUE HIGH-VALUES	no numérico	el carácter mayor en la secuencia de orden de caracteres (repetido apropiadamente)
LOW-VALUE LOW-VALUES	no numérico	el carácter menor en la secuencia de orden de caracteres (repetido apropiadamente)
QUOTE QUOTES	no numérico	las comillas (simples o dobles) que se usen (apropiadamente repetido)
ALL literal	no numérico	"literal" puede ser cualquier cadena de caracteres entre comillas (cadenas repetidas apropiadamente)

**EJEMPLO 5.55** Observe el uso de constantes figurativas y de literales en lo siguiente:

```
WORKING-STORAGE SECTION.
01 TODO-EL-WORKING-STORAGE.
    05 FILLER          PIC X(30)
                        VALUE "WORKING-STORAGE
                        EMPIEZA AQUI".
    05 BANDERA-ULTIMO-REGISTRO   PIC X(3)
                        VALUE HIGH-VALUES.
    05 AREA-SUBRAYADO        PIC X(100)  VALUE ALL "-".
    05 AREA-LINEA-EN-BLANCO   PIC X(100)  VALUE SPACES.
    05 CONTADOR-REGISTROS-INVALIDOS  PIC S9(4)   COMP-3
                        VALUE ZERO.
```

#### PROCEDURE DIVISION.

```
MOVE SPACES TO AREA-SALIDA-MENSAJES
MOVE "NUM-ARTICULO-INVALIDO" TO MENSAJE-SALIDA
MOVE 200.95 TO COTIZ-DIARIA
MOVE ALL "*" TO AREA-MENS-PAGADO
MOVE LOW-VALUES TO ID-MAESTRO-EMPLEADOS
```

Observe especialmente el uso de un literal no numérico para marcar el principio del WORKING-STORAGE. De este modo se facilita la tarea de encontrar el WORKING-STORAGE en cualquier circunstancia (véase la Sección 9.2)

#### Comparación entre VALUE... y MOVE...

Como se ha visto, la cláusula VALUE... se puede utilizar para inicializar un elemento del almacenamiento de trabajo:

```
05 CONTADOR-REGISTROS-INVALIDOS  PIC S9(4)  COMP-3  VALUE ZERO.
```

También con el mismo resultado se puede utilizar una instrucción MOVE en la división de procedimientos para inicializar el elemento:

```
MOVE ZERO TO CONTADOR-REGISTROS-INVALIDOS
```

Con respecto al uso de estas dos posibilidades se dan los siguientes consejos:

- (1) Si el elemento que se cita inicializando permanece inalterado durante la ejecución del programa, debe usarse la cláusula VALUE.
- (2) Si el valor del elemento cambia durante la ejecución del programa, es mejor usar para inicializar la instrucción MOVE.
- (3) Si el elemento no pertenece al WORKING-STORAGE (a excepción de los elementos de nivel 88 que se ven en la Sección 7.8), *no* puede usarse la cláusula VALUE.
- (4) Si el valor inicial no tiene importancia para el correcto funcionamiento del programa, es preferible no inicializarla.

## 5.20 NORMAS DE CODIFICACION

Recopilamos aquí una serie de normas recomendadas para la codificación de las divisiones de identificación, del entorno y de datos.

1. Los párrafos de la división de identificación se incluyen en las normas ANS porque la información que contienen es importante. No se salte las entradas de la IDENTIFICATION DIVISION.
2. Hay mucha flexibilidad para la escritura de la división de identificación: ayude a que se pueda leer fácilmente.
3. Utilice comentarios al final de la división de identificación para explicar lo que hace el programa.

4. Utilice puntos estructurados cuando una instrucción tenga varias cláusulas (por ejemplo, SPECIAL-NAMES, FD).
5. Incremente los números de nivel en más de una unidad para permitir posteriormente inserciones. (En este libro se suelen numerar 01, 05, 10, 15, 20,..., etc.).
6. Desplace las líneas para que se aprecien con claridad los números de nivel.
7. Cada nombre de dato debe tener un corto prefijo (o sufijo) que le asocie con el registro del que forma parte.
8. El resto de los nombres de datos deben ser tan descriptivos como sea posible (recuerde que dispone de hasta 30 caracteres).
9. Coloque las instrucciones SELECT en algún orden (por ejemplo, primero ficheros de entrada, después ficheros de entrada/salida y por último ficheros de salida) y utilice el mismo orden con las instrucciones FD.
10. No utilice elementos de nivel 77 en el almacenamiento de trabajo; en su lugar agrupe los elementos formando categorías bajo nombres descriptivos al nivel 01.
11. Utilice alguna norma para la codificación de elementos del almacenamiento de trabajo, tales como: interruptores de programa, banderas, contadores, constantes figurativas y literales, acumuladores, subíndices, tablas y copias de trabajo de los registros lógicos para los ficheros. Estos últimos deben ordenarse como en FD; en los ficheros de impresión deben definirse primero las copias de trabajo de las líneas de cabecera, a continuación las líneas del cuerpo y después los totales y las notas al pie.
12. Siempre que sea posible, alinee las cláusulas PIC, USAGE, SYNC y VALUE en columnas.
13. Si es posible, indique la función de cada elemento de datos en su nombre; por ejemplo, todos los interruptores de programa deberían comenzar con "INTER", todas las banderas con "BANDERA", todas las áreas de total con "TOTAL", todos los contadores con "CONTADOR" o con "NUMERO".
14. Algunos compiladores disponen de extensiones del COBOL ANS que permiten al programador separar los elementos en el listado del programa fuente para que sea más fácil su lectura. Por ejemplo, en el COBOL del sistema IMB OS/VS, la instrucción "EJECT" escrita en cualquier parte del margen B de una línea hace que la siguiente línea del listado comience al principio de una nueva página. "SKIP1", "SKIP2", "SKIP3", escrito en cualquier parte del margen B de una línea, obligan al compilador a saltar el número de líneas indicado. Estas posibilidades pueden usarse para que cada división y sección empiece en una página nueva y que los párrafos estén separados uno de otro (con el nombre del párrafo seguido de alguna línea en blanco). Las instrucciones SKIP1, SKIP2 y SKIP3 se procesan por el compilador con mayor rapidez que una línea en blanco en el propio programa.

### Preguntas de repaso

- 5.1 ¿Cuál es el fin de la división de datos?, ¿y el de la sección de ficheros?, ¿y el de la sección de almacenamiento de trabajo?
- 5.2 ¿Qué tipo de datos se encuentran en el almacenamiento de trabajo?
- 5.3 ¿Cuál es el propósito de una entrada a nivel 01 en la división de datos?
- 5.4 Explique la relación entre las instrucciones FD y SELECT.
- 5.5 ¿Qué información se especifica en la instrucción FD?
- 5.6 Explique cómo ahorra espacio en disco o cinta el bloqueo.
- 5.7 Explicar las diferencias entre los registros físicos y lógicos.

- 5.8 Defina *factor de bloqueo*.
- 5.9 ¿Qué son los espacios interregistros e interbloques?
- 5.10 Explique de qué manera el bloqueo aumenta la velocidad de procesamiento de un fichero secuencial en cinta o disco.
- 5.11 Explique el efecto del bloqueo en el procesamiento al azar de un fichero.
- 5.12 Explique cómo se manipulan los registros de longitud variable en el sistema IBM OS/VS:
- 5.13 Explique el modo de calcular el argumento de la cláusula BLOCK CONTAINS... para registros de longitud variable.
- 5.14 ¿Por qué es opcional la cláusula RECORD CONTAINS? ¿Cuándo debe utilizarse?
- 5.15 Explique el modo de calcular el argumento de la cláusula RECORD CONTAINS... para registros de longitud variable.
- 5.16 ¿Cuál es el significado especial de "BLOCK CONTAINS 0 RECORDS" en el COBOL del sistema IBM OS/VS?
- 5.17 Explique el uso de etiquetas con ficheros en cinta o disco.
- 5.18 ¿Qué es el directorio o VTOC?
- 5.19 ¿Cuál es la única entrada obligatoria en la instrucción FD?
- 5.20 ¿Qué ficheros se supone que tienen etiquetas y cuáles no?
- 5.21 ¿Por qué es tan infrecuente el uso de DATA RECORDS...? ¿Cuándo podría ser útil?
- 5.22 Dé un ejemplo de fichero con más de un tipo de registro lógico. Explique cómo se puede utilizar el *código de registro* para distinguir entre los diversos tipos.
- 5.23 Defina los siguientes términos asociados con la cláusula LINAGE: (a) página lógica, (b) cuerpo de página, (c) área de notas al pie, (d) margen superior, (e) margen inferior.
- 5.24 ¿Para qué tipo de fichero debe usarse la cláusula LINAGE?
- 5.25 ¿Qué es un *área de entrada/salida* o *buffer*?
- 5.26 Explique la relación entre las entradas al nivel 01 de la instrucción FD y los buffers de ficheros.
- 5.27 ¿Cuáles son los números de nivel válidos?
- 5.28 ¿Qué tiene de especial el nivel 01?
- 5.29 Explique cómo se utilizan los números de nivel para mostrar la estructura del registro.
- 5.30 ¿Qué importancia tiene el desplazamiento de líneas en la división de datos?
- 5.31 Defina: (a) elemento compuesto, (b) elemento simple.
- 5.32 Dé sugerencias para la invención de nombres de elementos.
- 5.33 En un fichero que contiene más de un tipo de registro lógico, ¿por qué ambos tipos suelen coincidir en unos bytes particulares?
- 5.34 Explique la forma en que los buffers múltiples suelen aumentar la velocidad de procesamiento.
- 5.35 ¿Para qué se utiliza "FILLER"?

- 5.36 ¿Qué se entiende por *edición*?
- 5.37 ¿Para qué sirve la cláusula PICTURE?
- 5.38 ¿Cómo se puede repetir fácilmente un carácter en una cadena de PICTURE?
- 5.39 Explique el uso de los caracteres "A" y "B" en la cláusula PICTURE.
- 5.40 ¿Por qué se usa tan frecuentemente el carácter "X" en la cláusula PICTURE?
- 5.41 Indique qué carácter debe usarse para representar "números" en la cláusula PICTURE.
- 5.42 Explique el uso de "S", "9" y "V" para representar números. Distinga entre esto y PIC "X".
- 5.43 Defina: (a) truncamiento, (b) redondeo.
- 5.44 ¿Por qué "S" y "V" no requieren espacio de almacenamiento adicional?
- 5.45 ¿Cuándo debe usarse "S" y por qué? ¿Cuándo ha de omitirse?
- 5.46 ¿Cuál será el resultado si se imprime un dato numérico con PIC "S", "9" y "V" en un sistema IBM?
- 5.47 Explique la edición a que dan lugar los caracteres: "Z", punto, coma, "\$", "CR", "DB", "-", "+".
- 5.48 Explique en qué consiste la protección de cheques.
- 5.49 Explique la diferencia entre edición fija y flotante.
- 5.50 ¿De qué forma se suelen imprimir los números negativos en los informes del mundo de los negocios?
- 5.51 Explique el uso con PIC de "0", "/" y "P".
- 5.52 Diga los formatos USAGE disponibles en el COBOL de IBM OS/VS e indique cuándo deben usarse cada uno de ellos.
- 5.53 ¿Cuándo un elemento debe tener formato DISPLAY?
- 5.54 ¿Por qué son COMP y COMP-3 más eficientes para hacer cálculos?
- 5.55 ¿Qué ventajas ofrece el uso de COMP y COMP-3 para campos numéricos en registros en cinta o disco?
- 5.56 Explique cómo determinar la longitud de un elemento cuyo formato es (a) DISPLAY, (b) COMP, (c) COMP-3.
- 5.57 Explique el uso de la cláusula BLANK WHEN ZERO.
- 5.58 Explique la función desempeñada por JUSTIFIED.
- 5.59 ¿Qué hace la cláusula OCCURS?
- 5.60 Comente las diversas opciones de SIGN IS...
- 5.61 Explique lo que hace la cláusula SYNCHRONIZED...
- 5.62 ¿Qué son los bytes de holgura? Relacionelos con el uso conveniente o no de SYNC.
- 5.63 Explique el uso de REDEFINES... Dé algún ejemplo concreto.
- 5.64 ¿Qué debe utilizarse en lugar del nivel 77 del almacenamiento de trabajo?
- 5.65 ¿Qué es la *basura*?

- 5.66 Diga las reglas para la formación en COBOL de literales y constantes figurativas.
- 5.67 Explique la función de la cláusula VALUE.
- 5.68 ¿Cuándo debe inicializarse un campo con la cláusula VALUE?, ¿y con una instrucción MOVE de la división de procedimientos?

### Problemas resueltos

- 5.69 ¿Qué está equivocado en la siguiente entrada del WORKING-STORAGE?

```
05 MAXIMO-VENTAS-TOTALES    PIC S9(5)99  VALUE HIGH-VALUE.
```

HIGH-VALUES y LOW-VALUES son constantes figurativas que pueden usarse sólo con PIC X, nunca en cálculos.

- 5.70 Dadas las instrucciones

```
SELECT FICHERO-PEDIDOS-CLIENTES    ASSIGN TO PEDIDOS.
SELECT INFORME-DE-PEDIDOS          ASSIGN TO INFORME.
SELECT FICHERO-MAESTRO-CLIENTES   ASSIGN TO MAESCLIE.
```

¿qué nombres FD deberán usarse en la sección de ficheros?

FILE SECTION.

```
FD  FICHERO-PEDIDOS-CLIENTES
```

```
.....
```

```
FD  INFORME-DE-PEDIDOS
```

```
.....
```

```
FD  FICHERO-MAESTRO-CLIENTES
```

```
.....
```

- 5.71 Suponga que el FICHERO-PEDIDOS-CLIENTES del Problema 5.70 contiene 65 bytes por registro con un factor de bloqueo 40. Almacenado en disco. Dé la descripción FD completa.

```
FD  FICHERO-PEDIDOS-CLIENTES
BLOCK CONTAINS 40 RECORDS
RECORD CONTAINS 65 CHARACTERS
LABEL RECORDS ARE STANDARD
```

Puesto que el fichero está en disco, se necesita usar las etiquetas estándar. En los sistemas IBM, hubiera sido mejor utilizar "BLOCK CONTAINS 0 RECORDS" para que el factor de bloqueo pudiera ser tomado de la etiqueta del fichero ya existente.

- 5.72 Si el fichero del Problema 5.71 contuviera 2.000 registros lógicos, cuántas operaciones de entrada/salida se necesitarían durante un procesamiento secuencial del fichero?

Con 40 registros lógicos por bloque, 2.000 registros lógicos requieren  $2.000/40 = 50$  bloques. Como cada entrada del disco transfiere un bloque completo, se necesitan sólo 50 operaciones.

En el caso de que el fichero no estuviera bloqueado, habría un registro lógico por cada registro físico y se hubieran necesitado 2.000 operaciones de entrada/salida.

- 5.73 ¿Por qué el número de "operaciones de entrada/salida del disco" es un indicativo importante de la velocidad con que se puede procesar el fichero?

A diferencia del acceso a datos contenidos en la memoria principal, que puede hacerse en nanosegundos ( $10^{-9}$  seg.) o menos, el acceso a datos contenidos en disco *algunas* veces implica el desplazamiento del brazo (a menos que se disponga de una cabeza por cada pista o la cabeza móvil esté precisamente situada sobre la pista que se necesita) y *siempre* implica el tiempo de rotación necesario para que la información se sitúe debajo de la cabeza. Estas operaciones son mecánicas y por tanto lentas en comparación con las operaciones de la computadora: para un disco modelo IBM 3380 el tiempo medio para el movimiento del brazo es 16 ms y el tiempo medio de rotación es 8,3 ms. Si el procesamiento del fichero contiene un número grande de accesos físicos al disco, estos tiempos en milisegundos cobran una importancia extraordinaria.

- 5.74 ¿Cómo afecta el bloqueo a la utilización del espacio en cinta o disco?

El bloqueo ahorra espacio en disco o cinta eliminando algunos de los espacios interregistros. Por ejemplo, en el fichero del Problema 5.72 se requieren 4 pistas en un disco modelo 3380 con un factor de bloqueo 40. Si el factor de bloqueo fuera 1, se necesitarían 25 pistas.

- 5.75 Suponga que el INFORME-DE-PEDIDOS del Problema 5.70 debe imprimirse en papel de 132 caracteres por línea. Dé la descripción completa de este fichero (FD).

```
FD INFORME-PEDIDOS
  RECORD CONTAINS 132 CHARACTERS
  LABEL RECORDS ARE OMITTED
  LINAGE IS 60
    WITH FOOTING AT 55
    LINES AT TOP 3
    LINES AT BOTTOM 3
```

Se omite la cláusula BLOCK CONTAINS porque los ficheros de impresión no deben estar bloqueados. No hay etiquetas en los ficheros de impresión (LABEL RECORDS ARE OMITTED). La cláusula LINAGE facilita a las instrucciones "WRITE" de la división de procedimientos el control de la posición de las líneas en la página. Con este fichero no se pueden utilizar los canales de control del carro porque se utiliza la cláusula LINAGE.

- 5.76 Suponga que el FICHERO-MAESTRO-CLIENTES del Problema 5.70 tiene dos tipos de registros lógicos: El REGISTRO-CLIENTE-NORMAL con 200 bytes y el REGISTRO-CLIENTE-PREFERENTE con 350 bytes. Detalle completamente la instrucción FD para un factor de bloqueo 40. Utilice BLOCK CONTAINS... RECORDS.

```
FD FICHERO-MAESTRO-CLIENTES
  BLOCK CONTAINS 40 RECORDS
  RECORD CONTAINS 200 TO 350 CHARACTERS
  LABEL RECORDS ARE STANDARD
  DATA RECORDS ARE
    REGISTRO-CLIENTE-NORMAL
    REGISTRO-CLIENTE-PREFERENTE

  01 REGISTRO-CLIENTE-NORMAL
  01 REGISTRO-CLIENTE-PREFERENTE
```

- 5.77 Resuelva el Problema 5.76 utilizando BLOCK CONTAINS... CHARACTERS.

```
FD FICHERO-MAESTRO-CLIENTES
  BLOCK CONTAINS 8164 TO 14164 CHARACTERS
  RECORD CONTAINS 200 TO 350 CHARACTERS
  LABEL RECORDS ARE STANDARD
  DATA RECORDS ARE
    REGISTRO-CLIENTE-NORMAL
    REGISTRO-CLIENTE-PREFERENTE
```

La cláusula RECORD CONTAINS... no incluye los bytes ocupados por las palabras de descripción de registro y bloque; pero, cuando se utiliza la opción CHARACTERS, BLOCK CONTAINS... debe

contener estas áreas de 4 bytes. El número de bytes precisos para contener un bloque mínimo es 8.164: cuarenta registros lógicos de 200 bytes más cuarenta palabras de descripción de registro de 4 bytes más una palabra de descripción de bloque de 4 bytes. De forma similar se precisan 14.164 bytes para almacenar un bloque de la máxima longitud.

En la versión de COBOL del sistema IBM OS/VS sería preferible especificar "BLOCK CONTAINS 0 CHARACTERS" para que el tamaño del bloque se pueda conocer a partir de la etiqueta del fichero (que ya existe y dispone de etiquetas estándar).

- 5.78** Si no existe un fichero y pretende crearse mediante un programa, ¿se puede utilizar "BLOCK CONTAINS 0...?"

Sí, pero en este caso la instrucción del lenguaje de control de trabajos que define el fichero debe especificar el tamaño del bloque o de lo contrario tendrá lugar un error.

- 5.79** Clasifique los dispositivos de entrada/salida que se han mencionado hasta el momento en términos de si soportan o no etiquetas de fichero.

<i>No soportan etiquetas</i>	<i>Soportan etiquetas estándar</i>
1. Lectores de tarjetas	1. Disco de cabeza móvil
2. Perforadores de tarjetas	2. Discos de cabeza fija y tambores
3. Impresoras	3. Cinta magnética (carrete) 4. Cinta magnética (casete) 5. Disco flexible

Las etiquetas son precisas en todos los ficheros en disco. En los ficheros en cinta suele estar generalizado el uso de etiquetas por la seguridad que proporcionan.

- 5.80** Escriba las instrucciones de la división de procedimientos que impriman una etiqueta de correo manipulando códigos de registros para el fichero de los Ejemplos 5.16 y 5.17.

```

READ FICHERO-ETIQUETAS-CORREO RECORD
  IF ETIQ-CORREO-CALLE-CODIGO IS EQUAL TO "1"
    MOVE ETIQ-CORREO-CALLE-NOMBRE      TO NOMBRE-ETIQUETA
    MOVE ETIQ-CORREO-CALLE-CALLE     TO CALLE-ETIQUETA
  ELSE
    IF ETIQ-CORREO-CALLE-CODIGO IS EQUAL TO "2"
      MOVE ETIQ-CORREO-ESTADO-CIUDAD   TO CIUDAD-ETIQUETA
      MOVE ETIQ-CORREO-ESTADO-ESTADO   TO ESTADO-ETIQUETA
      MOVE ETIQ-CORREO-ESTADO-CODPOSTAL TO CODPOSTAL-ETIQUETA
    ELSE
      DISPLAY "ERROR--CODIGO REGISTRO INVALIDO: IGNORADO"
    ENDIF
  ENDIF

```

La instrucción READ lee un registro lógico del FICHERO-ETIQUETAS-CORREO. Recuerde que como se vio en el Ejemplo 5.17 el código de registro ocupa el primer byte del registro lógico e identifica su tipo. Suponga que los registros tipo calle tienen "1" en el primer byte y los registros tipo estado tienen un "2". Como el primer byte de cada registro es del tipo PIC X, no importa cuál de los dos nombres utilice el programador para referirse a él (ETIQ-CORREO-CALLE-CODIGO y ETIQ-CORREO-ESTADO-CODIGO, ambos se refieren al primer byte del registro lógico y lo interpretan como "PIC X").

Después de la instrucción READ, el programa utiliza la instrucción IF para comprobar si el valor del primer byte del registro lógico es "1". En caso afirmativo, se mueven el nombre y la calle al área de la etiqueta; en caso negativo, se comprueba si el primer byte del registro lógico es "2". Si este primer byte no es ni "1" ni "2", el programa imprime un mensaje de error e ignora el registro lógico correspondiente. Observe cómo los nombres utilizados en las sentencias MOVE dependen totalmente del código de registro.

- 5.81** Tanto PIC A como PIC X se pueden usar con datos alfabéticos. ¿Cuál es mejor?

Muchos programadores en COBOL utilizan PIC X en lugar de PIC A para datos alfabéticos. No hay inconvenientes y sí algunas ventajas: algunas instrucciones de la división de procedimientos funcionan con PIC X y no con PIC A (véanse los Capítulos 6 y 7).

- 5.82** Clasifique los elementos de los que a continuación se da su cláusula PICTURE en una de estas cuatro categorías: (i) alfabético, (ii) alfanumérico, (iii) numérico y (iv) numérico editado.

(a) AABAAA	(e) 99/99/99	(i) A(10)
(b) XXBXXX	(f) \$ZZ,ZZZ.99	(j) 9(3)V99
(c) S9(3)V99	(g) S99PP	(k) \$,\$,\$,\$.99BCR
(d) S9(3).99	(h) X(7)	(l) Z999

(a) i; (b) ii; (c) iii; (d) iv (el signo operacional “S” y el carácter “.” son incompatibles); (e) iv; (f) iv; (g) iii; (h) ii; (i) i; (j) iii; (k) iv; (l) iv.

- 5.83** Dar las cláusulas PICTURE y USAGE (si no es DISPLAY) apropiadas para los siguientes elementos de datos (en el caso de más de una posibilidad, poner la mejor en primer lugar): (a) un número de la seguridad social (sin guiones); (b) el importe de un cheque inferior a 1.000 dólares; (c) un número de la seguridad social con guiones; (d) un apellido (máximo 20 caracteres); (e) número de horas extras (redondeado a la decena)—fichero en cinta; (f) número de artículos en un almacén (hasta 99.999)—fichero en disco; (g) saldo de una hipoteca (hasta 500.000,00)—fichero en tarjetas; (h) saldo de una hipoteca (hasta 500.000,00)—fichero en disco; (i) saldo de una hipoteca (hasta 500.000,00)—informe impreso; (j) número de horas extras (redondeado al más cercano múltiplo de 0,1)—fichero en tarjetas; (k) número de horas extras (redondeado al más cercano múltiplo de 0,1)—informe impreso; (l) número de artículos en un almacén (hasta 99.999)—fichero en tarjetas; (m) número de artículos en un almacén (hasta 99.999)—informe impreso; (n) flete por libra (redondeado al más cercano múltiplo de 0,1)—fichero en tarjetas; (o) flete por libra (redondeado al más cercano múltiplo de 0,1)—informe impreso; (q) una fecha en el formato aa mm dd (sin guiones)—fichero en tarjetas; (r) una fecha en el formato aa mm dd (sin guiones)—fichero en disco; (s) una fecha en formato aa mm dd, editada para su impresión; (t) un código de registro de un dígito—fichero en tarjetas; (u) un código de registro de un dígito—fichero en disco; (v) un código de registro de un dígito—informe impreso; (w) instrucciones de envío (hasta 70 caracteres)—ficheros en tarjetas disco impresora; (x) número de artículo (7 dígitos)—fichero en tarjetas; (y) un número de artículo (7 dígitos)—fichero en disco; (z) el salario que le gustaría ganar (expresado en el COBOL de IBM OS/VS)—informe impreso.

(a) PIC X(9) o PIC 9(9) (b) PIC \$.\*.\*.\*.99 (c) PIC X(11) (d) PIC X(20) o PIC A(20)  
 (e) PIC S99V9 o PIC 99V9 con COMP-3 o COMP (f) PIC 59(5) o PIC 9(5) con COMP-3 o COMP  
 (g) PIC S9(6)V99 o PIC 9(6)V99 (h) PIC S9(6)V99 o PIC 9(6)V99 con COMP-3 o COMP  
 (i) PIC \$\$\$\$,\$\$\$\$.99- (j) PIC S99V9 o PIC 99V9 (k) PIC ZZ.9 (l) PIC S9(5) o PIC 9(5)  
 (m) PIC ZZ,ZZ9- o PIC ZZ,ZZ9 (n) PIC S9(3)V999 o PIC (3)V999 (o) PIC S9(3)V999 o PIC  
 9(3)V999 con COMP-3 o COMP (p) PIC ZZZ.999 (q) PIC X(6) o PIC 9(6) (r) PIC X(6) o PIC  
 9(6) (s) PIC XX/XX/XX o PIC 99/99/99 (t) PIC X o PIC 9 (u) PIC X o PIC 9 (v) PIC X o  
 PIC 9 (w) PIC X(70) (x) PIC X(7) o PIC 9(7) (y) PIC X(7) o PIC 9(7)  
 (z) PIC \$\$,\$\$\$,\$\$\$,\$\$\$,.99

- 5.84** Calcular la longitud en byte de todos los datos definidos en el Problema 5.83. (Asuma que está trabajando con COBOL para un sistema IBM OS/VS a los efectos de las longitudes de los datos en formatos COMP y COMP-3.)

- (a) 9 (con DISPLAY cada carácter de PIC, excepto “S” y “V”, requiere 1 byte).  
 (b) 9 (con DISPLAY cada carácter de PIC, excepto “S” y “V”, requiere 1 byte).  
 (c) 11  
 (d) 20  
 (e) COMP-3: 2 (PIC tiene 3 dígitos, luego  $[3/2] + 1 = 2$  bytes)  
 COMP : 2 (se pueden manipular de 1 a 4 dígitos)  
 (f) COMP-3: 3 (PIC tiene 5 dígitos, luego  $[5/2] + 1 = 3$  bytes)  
 COMP : 4 (se pueden manipular de 5 a 9 dígitos)  
 (g) 8 (con DISPLAY, cada carácter PIC, a excepción de “S” y “V”, requiere 1 byte)

- (h) COMP-3: 5 (PIC tiene 8 dígitos, luego  $[8/2] + 1 = 5$  bytes)  
COMP : 4 (se pueden manipular de 5 a 9 dígitos)
- (i) 12
- (j) 3 ("S" y "V" no ocupan espacio)
- (k) 4 (el punto "." representa un punto decimal real y ocupa 1 byte)
- (l) 5
- (m) 7 con el signo menos por delante, 6 sin el signo menos
- (n) 6 ("S" y "V" no ocupan espacio)
- (o) COMP-3: 4 (PIC tiene 6 dígitos, luego  $[6/2] + 1 = 4$  bytes)  
COMP : 4 (se pueden manipular de 5 a 9 dígitos)
- (p) 7 (r) 6 (t) 1 (v) 1 (x) 7
- (q) 6 (s) 8 (u) 1 (w) 70 (y) 7
- (z) 17 (cada carácter PIC en un dato editado requiere 1 byte)

**5.85** Dada la instrucción MOVE ZEROS TO ELEMENTO-DATOS-SIMPLE, mostrar el resultado al definirse ELEMENTO-DATOS-SIMPLE como:

- |                         |                                      |                  |
|-------------------------|--------------------------------------|------------------|
| (a) PIC S9(3)V99 COMP-3 | (b) PIC S9(3)V99                     | (c) PIC ZZ.ZZ    |
| (d) PIC \$**,**.*       | (e) PIC \$**,**.*<br>BLANK WHEN ZERO | (f) PIC Z,ZZZ.99 |
| (g) PIC Z,ZZZ.Z9        | (h) PIC Z,ZZZ.99<br>BLANK WHEN ZERO  | (i) PIC X(4)     |
| (j) PIC A(4)            | (k) PIC XX/XX/XX                     | (l) PIC XXBXXBXX |

- (a) +000,00 (formato COMP-3)
- (b) +000,00 (formato DISPLAY)
- (c) bbbb
- (d) \$\*\*\*\*\*.\* ("." siempre se imprime con protección de cheque)
- (e) Inválido: PIC A sólo puede admitir letras y espacios.
- (f) bbbbb.00
- (g) bbbbbb0
- (h) bbbbbb0
- (i) 0000 (ceros EBCDIC)
- (j) Inválido: PIC A sólo puede admitir letras y espacios
- (k) 00/00/00
- (l) 00b00b00

**5.86** Dada la instrucción MOVE ZEROS TO ELEMENTO-DATOS-SIMPLE, mostrar el resultado al definirse ELEMENTO-DATOS-SIMPLE como:

- (a) PIC X(3) (b) PIC A(4) (c) PIC 9(3) (d) PIC S9(4)V99
- (e) PIC XXX/XX

- (a) bbb (b) bbbb (c) inválido (d) inválido (e) bbb/bb

(Recuerde que "PIC 9" sólo puede contener un dígito.)

**5.87** Dada la instrucción MOVE "ABCDE" TO ELEMENTO-DATOS-SIMPLE, mostrar el resultado al definirse ELEMENTO-DATOS-SIMPLE como:

- (a) PIC X(3) (b) PIC X(3) JUSTIFIED RIGHT (c) PIC XX/XX
- (d) PIC X(8) (e) PIC X(8) JUSTIFIED RIGHT
- (a) ABC (truncamiento por la derecha)
- (b) CDE (truncamiento por la izquierda, como consecuencia de la cláusula JUSTIFIED RIGHT)
- (c) AB/CD (truncamiento por la derecha; se inserta "/")
- (d) ABCDEbbb (espacios por la derecha)
- (e) bbbABCDE (la cláusula JUSTIFIED RIGHT mueve el valor a la parte derecha del campo; los espacios aparecen por la izquierda)

**5.88** ¿Por qué es importante el uso de SYNC con elementos en formato COMP?

La importancia de la cláusula SYNCHRONIZING en relación con elementos en formato COMP depende de la arquitectura de la computadora de que se trate. En los sistemas IBM-370, el ahorro de tiempo es insignificante. A causa de la inserción de los bytes de holgura (Sección 5.16), SYNC no debe usarse con elementos COMP que pertenezcan a registros lógicos.

**5.89** Dar el resultado de la instrucción MOVE ZEROS TO ELEM-COMP-A si dicho elemento se define como:

```
01 ELEM-COMP-A
 05 A          PIC X(2).
 05 B          PIC S9(3)V99.
 05 C          PIC S9(5)V99  COMP-3.
 05 D          PIC S9(4)  COMP.
```

Al manipular elementos compuestos en la división de procedimientos, siempre se trata como *un* (largo) campo PIC X. Como la longitud de ELEM-COMP-A es 13 bytes (2+5+4+2), la instrucción MOVE procesa ELEM-COMP-A como si estuviera definido con PIC X(13). Por tanto, se trasladan 13 ceros DISPLAY al interior del campo. Estos ceros encajan correctamente en los campos A y B puesto que están en formato DISPLAY (por defecto). Sin embargo, C requiere ceros COMP-3 y D requiere ceros COMP en lugar de ceros DISPLAY. El resultado es que los campos C y D no contienen datos válidos; de hecho la manipulación de los campos C y D producirá resultados erróneos y muy probablemente el programa fallará.

**5.90** Dado

```
01 ELEMENTO-SIMPLE.
 05 A          PIC 9(4)V99.
 05 B REDEFINES A.
 10 C          PIC 9(2)V9.
 10 D          PIC 9(3).
```

¿cuál es el contenido de C y D después de que la instrucción MOVE 12.34 TO A se ejecute?

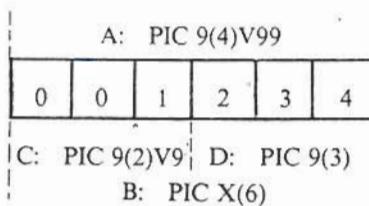


Fig. 5-10

Como consecuencia de la instrucción MOVE, A contiene 0012.34. Sin embargo, como resultado de REDEFINES los campos C y D se refieren a las mismas posiciones de memoria que el campo A. La situación se esquematiza en la Figura 5-10. Se ve que C contiene la mitad izquierda de A, es decir (con el actual contenido de A), 00.1; D contiene 234 (mitad derecha de A). B, que es un elemento compuesto, contendría 001234 y sería tratado como si estuviera definido como PIC X(6) DISPLAY.

**5.91** Dado

```
01 ELEMENTO-SIMPLE.
 05 A          PIC X(8)      VALUE "ABCDEFGH".
 05 B REDEFINES A.
 10 C          PIC X(3).
 10 D          PIC X(2).
 10 E          PIC X(3).
```

¿cuál es el contenido de C, D, E y B?

Como B está redefinido con A, se está en la situación de la Figura 5-11.

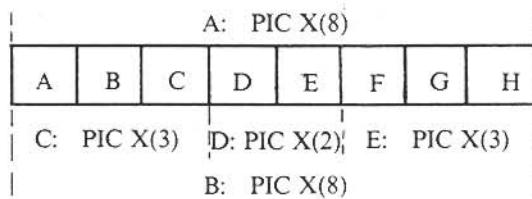


Fig. 5-11

Al ser B un elemento compuesto, se considera PIC X(8). Tenga en cuenta que para usar REDEFINES, ambos elementos deben tener la misma longitud. C contiene "ABC", D contiene "DE" y E contiene "FGH". B contiene "ABCDEFGH" al referirse a los mismos lugares de memoria que el elemento A.

- 5.92 Defina un cuerpo que vaya a ser parte de una línea impresa. El campo debe contener bien (1) el valor (máximo 50.000,00) que tienen las existencias de un artículo o (2) el mensaje "CONSUMIDO" si las existencias se han acabado.

```
05 VALOR-INVENTARIO      PIC ZZ,ZZZ.99.
05 AREA-MENS-SIN-EXISTENCIAS  REDEFINES VALOR-INVENTARIO
                                PIC X(9).
```

Este problema ilustra la necesidad y el medio de obtener que un mismo cuerpo de un registro disponga a la vez de PICTURE numérico y alfanumérico. La instrucción "MOVE "CONSUMIDO" TO VALOR-INVENTARIO" produciría un error sintáctico, puesto que a ese campo sólo se pueden transferir datos numéricos. La instrucción "MOVE "CONSUMIDO" TO AREA-MENS-SIN-EXISTENCIAS" es correcta y realiza el mismo cometido que la anterior.

- 5.93 ¿Qué está equivocado en lo siguiente?

```
01 ELEMENTO-SIMPLE.
  05 A                  PIC X(5).
  05 B                  REDEFINES A.
    10 C                PIC S9(5)      COMP-3.
    10 D                PIC S9(4)      COMP.
```

Nada está equivocado. La longitud de B es  $3 + 2 = 5$  bytes, que es igual que la longitud de A. El programador puede cambiar libremente las cláusulas PICTURE y USAGE de los subcampos cuando se utiliza REDEFINES. Las únicas condiciones son: (i) la longitud debe ser la misma, (ii) no puede usarse REDEFINES para elementos de nivel 01 en la sección de ficheros, (iii) los elementos redefinido y redefinidor deben tener el mismo nivel.

- 5.94 ¿Qué está equivocado en lo siguiente?

```
01 ELEMENTO-SIMPLE      PIC X(5).
01 OTRO-ELEMENTO       REDEFINES ELEMENTO-SIMPLE.
  05 A                  PIC X(4).
  05 B                  PIC X(3).
  05 C                  PIC X(2) REDEFINES B.
```

- (1) REDEFINES se utiliza con elementos de nivel 01: correcto en el WORKING-STORAGE, pero incorrecto en la sección de ficheros.
- (2) La longitud de ELEMENTO-SIMPLE no es igual a la de OTRO-ELEMENTO.
- (3) La longitud de C no es igual que la de B.
- (4) Para el elemento C, REDEFINES debe ser la primera cláusula de la descripción.
- (5) El nivel del elemento redefinido C no es el mismo que el del elemento B.

5.95 ¿Qué está equivocado en lo siguiente?

```
WORKING-STORAGE SECTION.  
01 CONTADORES-DE-PROGRAMA.  
    05 NUMERO-DE-FACT-IMPAGADAS    PIC S9(5)    COMP-3.
```

PROCEDURE DIVISION.

ADD 1 TO NUMERO-DE-FACT-IMPAGADAS

La instrucción ADD coge el valor de NUMERO-DE-FACT-IMPAGADAS, le suma 1 y almacena el resultado en NUMERO-DE-FACT-IMPAGADAS. Pero ¿cuál es el contenido de NUMERO-DE-FACT-IMPAGADAS cuando comienza la ejecución del programa? Como el programador no ha inicializado el dato, su contenido inicial es *indefinido o basura*. Este es un error muy común en los programadores principiantes; las consecuencias son una cuenta incorrecta o un fallo del programa.

5.96 ¿Qué puede hacerse para eliminar el error del Problema 5.95?

El elemento debe inicializarse a cero. Como su valor cambia durante la ejecución del programa, es preferible inicializar mediante una instrucción de la división de procedimientos:

```
PROCEDURE DIVISION.  
MOVE ZERO TO NUMERO-DE-FACT-IMPAGADAS  
.....  
ADD 1 TO NUMERO-DE-FACT-IMPAGADAS
```

Como el NUMERO-DE-FACT-IMPAGADAS se define en el almacenamiento de trabajo, se podría utilizar como método alternativo con la cláusula VALUE el siguiente:

05 NUMERO-DE-FACT-IMPAGADAS PIC S9(5) COMP-3 VALUE ZERO.

que hará que NUMERO-DE-FACT-IMPAGADAS contenga cero (en lugar de basura) al comenzar la ejecución del programa. (Las normas de codificación de algunas compañías reservan el uso de la cláusula VALUE para aquellos elementos cuyo valor no se altera durante la ejecución del programa.)

5.97 Muchos programadores prefieren no escribir descripciones detalladas de los registros lógicos en la FILE SECTION. En su lugar, definen un área en el almacenamiento de trabajo de la misma longitud que un registro lógico. Cuando se da entrada a un registro lógico del fichero (mediante la instrucción READ), el programa mueve una copia del registro lógico al almacenamiento de trabajo. Toda la manipulación de los datos se realiza utilizando la copia del WORKING-STORAGE. De forma similar, los registros de salida se preparan en un área del almacenamiento de trabajo de longitud y descripción apropiadas, después se traslada al área del registro lógico del fichero (nivel 01 en FD) antes de ejecutar la instrucción WRITE.

Modifique la siguiente sección de ficheros para que el procesamiento pueda tener lugar en el almacenamiento temporal.

FILE SECTION.

```
FD FICHERO-ENTRADA-TIEMPOS  
BLOCK CONTAINS 0 CHARACTERS  
RECORD CONTAINS 21 CHARACTERS  
LABEL RECORDS ARE STANDARD
```

01 REGISTRO-TIEMPO.	
05 ID-EMPLEADO-TIEMPO	PIC X(5).
05 HORAS-NORMALES-TIEMPO	PIC 99V9.
05 HORAS-EXTRAS-TIEMPO	PIC 99V9.
05 GASTOS-TIEMPO	PIC S9(4)V99.
05 DESCANSO-TIEMPO	PIC 99.
05 VACACIONES-TIEMPO	PIC 99.

## FILE SECTION.

FD FICHERO-ENTRADA-TIEMPOS  
 BLOCK CONTAINS 0 CHARACTERS  
 RECORD CONTAINS 21 CHARACTERS  
 LABEL RECORDS ARE STANDARD

01 REGISTRO-TIEMPO PIC X(21).

## WORKING-STORAGE SECTION.

01	WORKING-TIME-RECORD-AREA.	
05	ID-EMPLEADO-TRABAJO	PIC X(5).
05	HORAS-NORMALES-TRABAJO	PIC 99V9.
05	HORAS-EXTRAS-TRABAJO	PIC 99V9.
05	GASTOS-TRABAJO	PIC S9(4)V99.
05	DESCANSO-TRABAJO	PIC 99.
05	VACACIONES-TRABAJO	PIC 99.

Observe que la descripción del registro en la sección de fichero se ha convertido en trivial —sólo PIC X(21), donde 21 es la longitud del registro. La descripción detallada de cada campo del registro se ha llevado al almacenamiento temporal. Observe que los campos HORAS-NORMALES, HORAS-EXTRAS, DESCANSO y VACACIONES no tienen signos. Esto no afecta a la eficiencia del programa objeto puesto que *los elementos nunca recibirán el resultado de un cálculo ni pueden ser negativos* (sin PIC "S" es imposible). Un programador conservador usaría PIC "S" en estos elementos por si las moscas.

- 5.98 El procedimiento de definición en el WORKING-STORAGE que se vio en el Problema 5.97 casi siempre se utiliza para los ficheros de salida por impresora porque sus registros tienen mucha información constante como el título de la página, las cabeceras de las columnas, etc. Se puede utilizar la cláusula VALUE para inicializar tales datos. Revise la descripción que sigue y modifiquela para que los registros se puedan definir en el almacenamiento de trabajo. Crear también un registro para la impresión de cabeceras de columnas de esta información.

## FILE SECTION.

FD FICHERO-NOMINAS  
 RECORD CONTAINS 132 CHARACTERS  
 LABEL RECORDS ARE OMITTED

01	REGISTRO-LINEA-NOMINA.	
05	REGISTRO-ID-EMPLEADO	PIC X(5).
05	FILLER	PIC X(10).
05	REGISTRO-HORAS-NORMALES	PIC ZZ.9.
05	FILLER	PIC X(10).
05	REGISTRO-HORAS-EXTRAS	PIC ZZ.9.
05	FILLER	PIC X(99).

## FILE SECTION.

FD FICHERO-NOMINAS  
 RECORD CONTAINS 132 CHARACTERS  
 LABEL RECORDS ARE OMITTED

01 REGISTRO-LINEA-NOMINA PIC X(132).

## WORKING-STORAGE SECTION.

01	REGISTRO-LINEA-TRABAJO.	
05	ID-EMPLEADO-TRABAJO	PIC X(5).
05	FILLER	PIC X(10) VALUE SPACES.
05	HORAS-NORMALES-TRABAJO	PIC ZZ.9.
05	FILLER	PIC X(10) VALUE SPACES.
05	HORAS-EXTRAS-TRABAJO	PIC ZZ.9.
05	FILLER	PIC X(99) VALUE SPACES.
01	LINEA-CABECERA.	
05	FILLER	PIC X(15) VALUE "ID-EMPLEADO".
05	FILLER	PIC X(14) VALUE "HORAS-NORMALES".
05	FILLER	PIC X(103) VALUE "HORAS-EXTRAS".

Observe que ahora el REGISTRO-LINEA-NOMINA se define simplemente como PIC X(132) y la descripción real se deja para REGISTRO-LINEA-TRABAJO. Las áreas "FILLER" para separar los datos impresos se inicializan con espacios mediante la cláusula VALUE (que no podría usarse cuando la descripción se daba en la sección de ficheros). La LINEA-CABECERA sólo tiene campos "FILLER" que se llenan adecuadamente con la cláusula VALUE. Compruebe usted mismo que cada cabecera de columna comienza sobre el primer carácter de información en la misma. (Observe, sin embargo, que no están centradas.)

### Ejercicios de programación

- 5.99-5.108 Modifique los Problemas 2.29-2.38 para reflejar su mejor conocimiento de la división de datos. Cambie todos los importes en dólares a dólares con céntimos, modificando las cláusulas PIC que sean necesarias. Edite todas las salidas por impresora y haga que los cálculos se realicen del modo más eficiente.

# Capítulo 6

## **División de procedimientos (PROCEDURE DIVISION)**

### **6.1 INTRODUCCION: NORMAS DE CODIFICACION**

Las divisiones de identificación, entorno y datos sirven como una especie de prólogo de la división de procedimientos, en la cual el programador especifica las instrucciones que tiene que llevar a cabo la computadora. En este capítulo se asumirá que estas instrucciones se refieren a la entrada, procesamiento y salida de *ficheros organizados secuencialmente (acceso secuencial)*.

Estructuralmente, la división de procedimientos se compone de secciones y/o párrafos. La cabecera de división ("PROCEDURE DIVISION") se escribe en el margen A; las cabeceras de secciones y párrafos también empiezan en el margen A. Los párrafos y secciones se componen de *frases* que se separan mediante puntos. Cada frase se compone de una o varias *instrucciones* de la división de procedimientos y que deben empezar con un *verbo COBOL*.

#### **Normas de codificación de la división de procedimientos**

1. Escriba sólo una instrucción por línea.
2. Las cabeceras de párrafo y sección deben ocupar ellas solas una línea.
3. Cada párrafo o sección debe tener un objetivo concreto; el nombre debe describir dicho objetivo.
4. Ordene las cabeceras de párrafo o sección mediante prefijos o sufijos añadidos a los nombres.
5. Desplace las líneas (comenzando en la columna 12) para hacer visible la continuación de una instrucción en otra línea y para clarificar las relaciones entre las instrucciones.
6. Use líneas en blanco, líneas de comentarios en blanco (mejor que las anteriores si se dispone de ellas), EJECT, SKIP1, SKIP2 y SKIP3, para dejar espacio suficiente entre los párrafos y secciones y para separar las cabeceras.
7. Use el punto sólo cuando sea absolutamente necesario y utilice preferiblemente el punto estructurado.
9. Evite el uso de SECTION a menos que de verdad se requiera.
10. Procure que todo sea tan simple y directo como sea posible.

### **6.2 ENTRADA/SALIDA: INSTRUCCION OPEN**

Todo fichero tiene que ser abierto (OPENed) antes de poder usar cualquier otra instrucción de la división de procedimientos con él. (i) OPEN... genera el espacio en memoria necesario para los buffers. (ii) OPEN... hace que se procesen las etiquetas del fichero si existen (es decir, si se incluyó en la FD la cláusula LABEL RECORDS ARE STANDARD):

**Fichero de entrada o de entrada/salida.** Las etiquetas existentes se comprueban para asegurarse de que el fichero es accesible y está correctamente identificado.

**Fichero de salida.** Se crean las etiquetas para el fichero.

(iii) OPEN... posiciona el fichero para el acceso a sus registros lógicos —por lo general al principio (primer registro lógico) del fichero.

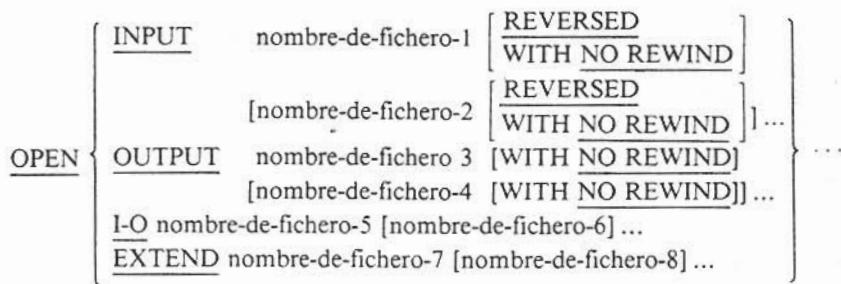


Fig. 6-1

La sintaxis general de la instrucción OPEN se muestra en la Figura 6-1. Como allí se indica, debe especificarse por lo menos uno de los cuatro modos OPEN; en cada modo pueden abrirse tantos ficheros como se desee.

**Modo INPUT.** El fichero se abre *sólo* para entrada; el fichero debe existir y ser accesible o de lo contrario el programa fallará. "READ" es el único verbo que puede utilizarse y con él se produce la entrada de un registro lógico.

**Modo OUTPUT.** El fichero se abre *sólo* para salida; el único verbo que puede aplicarse es "WRITE". Si el fichero ya existiera, la versión antigua se borraría totalmente y la nueva versión tiene que crearse mediante el uso de instrucciones WRITE para la escritura de registros. Si el fichero no existiera, se crearía uno que debería llenarse mediante instrucciones WRITE.

**Modo I-O.** El fichero se abre para entrada y para salida; el fichero ya debe existir y ser accesible o de lo contrario el programa fallará. Se pueden utilizar los verbos "READ", "WRITE" y "REWRITe"; en un fichero soportado por un dispositivo de acceso directo, el verbo REWRITe se puede utilizar para grabar una versión modificada de un registro lógico en su posición original en el fichero (actualización en su lugar).

**Modo EXTEND.** El fichero se abre *sólo para salida* y se posiciona al final (*después* del último registro). De esta forma, cuando se utiliza la instrucción WRITE para la escritura de registros lógicos, se añaden al final físico del fichero, respetando los registros ya existentes.

Las opciones "REVERSED" y "WITH NO REWIND" se utilizan poco y se aplican sólo a ficheros en cinta magnética.

**Opción REVERSED.** Esta opción sólo puede utilizarse con ficheros en cinta abiertos en modo INPUT. Indica que el fichero se posicionará al final y se leerá de atrás para delante (no todas las unidades de cinta permiten esto).

**Opción WITH NO REWIND.** Esta opción sólo puede utilizarse con ficheros en cinta abiertos en modo INPUT o en modo OUTPUT. Indica que *el fichero no será posicionado durante el proceso de apertura*. De esta forma el fichero quedará posicionado donde quiera que estuviera al ejecutar la instrucción OPEN.

**EJEMPLO 6.1** Todos los ficheros utilizados en la siguiente instrucción OPEN se deben corresponder con los que aparezcan en las instrucciones SELECT y FD:

OPEN	INPUT	FICHERO-TIEMPO-EDITADO
		VIEJO-MAESTRO-EMPLEADOS
OUTPUT		INFORME-NOMINAS
		NUEVO-MAESTRO-EMPLEADOS
EXTEND		FICHERO-SUMARIO-TIEMPO

Observe el cumplimiento de las normas de codificación para la escritura de instrucciones OPEN: todos los ficheros comienzan a escribirse en la misma columna; todos los modos OPEN se escriben en la misma columna con adecuada separación.

Alternativamente cada fichero podría tener su propia instrucción OPEN. En general, la apertura de varios ficheros con una sola instrucción OPEN consumirá más memoria de la computadora durante la apertura, pero el proceso será más rápido.

**EJEMPLO 6.2** Lo siguiente ilustra un error muy común entre programadores principiantes:

```

IDENTIFICATION DIVISION.
PROGRAM-ID. ERRONEO.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FICHERO-TIEMPO-TARJETAS
        ASSIGN TO WKLYTIME

    SELECT INFORME-NOMINAS
        ASSIGN TO TIMEEDIT

```

```

DATA DIVISION.
FILE SECTION.
FD FICHERO-TIEMPO-TARJETAS
    LABEL RECORDS ARE OMITTED

01 REGISTRO-TIEMPO-TARJETAS.
    05 ID-TIEMPO-TARJETAS      PIC X(6).
    .....

FD INFORME-NOMINAS
    LABEL RECORDS ARE OMITTED
    RECORD CONTAINS 132 CHARACTERS

01 LINEA-EDICION.
    05 ID-EDICION            PIC X(6).
    05 FILLER                PIC X(5).
    .....

```

```

PROCEDURE DIVISION.
    MOVE ALL "*" TO LINEA-EDICION
    WRITE LINEA-EDICION
        AFTER ADVANCING PAGE
    OPEN   OUTPUT   INFORME-NOMINAS
    OPEN   INPUT    FICHERO-TIEMPO-TARJETAS
    .....

```

Observe de cerca la división de procedimientos. LINEA-EDICION, el área del registro lógico de INFORME-NOMINAS, *no debe ser manipulada* hasta que el fichero no haya sido abierto. Tal y como están las cosas, la instrucción "MOVE ALL \*\*" TO LINEA-EDICION" hará abortar al programa.

**EJEMPLO 6.3** Suponga una cláusula FILE STATUS (Sección 4.5) en el programa (corregido) del Ejemplo 6.2.

```

FILE-CONTROL.
    SELECT FICHERO-TIEMPO-TARJETAS
        ASSIGN TO WKLYTIME
        FILE STATUS IS TIEMPO-TARJETAS-ESTADO
    .....

```

```

WORKING-STORAGE SECTION.
01 TIEMPO-TARJETAS-ESTADO      PIC XX.
    .....

```

```

PROCEDURE DIVISION.
    .....
    OPEN INPUT FICHERO-TIEMPO-TARJETAS
        IF TIEMPO-TARJETAS-ESTADO NOT EQUAL "00"
    .....

```

TIEMPO-TARJETAS-ESTADO se define como un área de dos bytes (PIC XX). Después de ejecutar la instrucción OPEN, esta área debe contener el literal "00" si la apertura se ha realizado con éxito. Si se hubiera definido TIEMPO-TARJETAS-ESTADO con PIC 99, tendría que usarse un literal *numérico* (o una constante figurativa equivalente) para la comprobación:

```
IF TIEMPO-TARJETAS-ESTADO NOT EQUAL 0 ...
o
IF TIEMPO-TARJETAS-ESTADO NOT EQUAL ZERO ...
```

### 6.3 ENTRADA/SALIDA: INSTRUCCION CLOSE

La instrucción CLOSE finaliza el procesamiento de un fichero mediante (i) recuperación del espacio de memoria ocupado por los buffers, (ii) realización del procesamiento de las etiquetas en los ficheros que dispongan de etiquetas estándar y (iii) *deshaciendo* todo lo que hizo la instrucción OPEN. Todo fichero que se abra mediante OPEN debe cerrarse mediante CLOSE antes de que termine el programa; olvidarse de cerrar un fichero puede conducir a serios errores en algunos sistemas. La sintaxis de la instrucción CLOSE se muestra en la Figura 6-2.

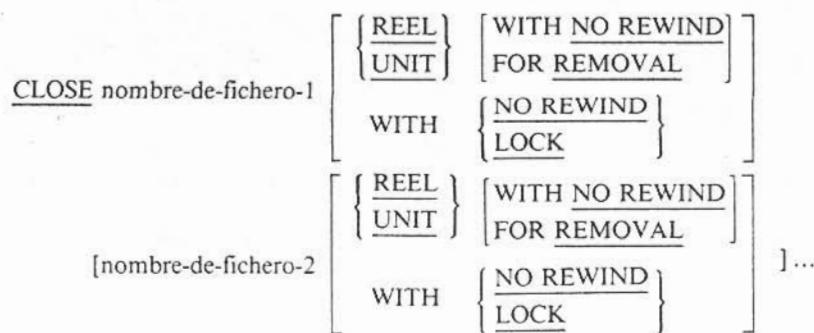


Fig. 6-2

#### Versión principal de CLOSE...

La versión más frecuente de la instrucción es simplemente el verbo seguido de uno o más nombres de fichero. Los ficheros se cierran individual (utilizando menos memoria) o colectivamente (empleando menos tiempo) y en cualquier orden (independientemente del orden en el que se abrieron). Cualquiera que sea la versión que se utilice de CLOSE, si se ha especificado la cláusula FILE STATUS... para el fichero que se está cerrando, la instrucción hará que se coloque un código de estado en el elemento de la cláusula FILE STATUS para indicar el éxito o fracaso de la operación de cierre.

#### EJEMPLO 6.4

PROCEDURE DIVISION.

```

OPEN OUTPUT FICHERO-MAESTRO-INVENTARIO
.....  

..... (use WRITE para añadir registro a FICHERO-MAESTRO-INVENTARIO)
.....  

CLOSE FICHERO-MAESTRO-INVENTARIO
OPEN INPUT FICHERO-MAESTRO-INVENTARIO
.....  

..... (use READ para dar entrada a registros lógicos de FICHERO-MAESTRO-INVENTARIO)
.....  

CLOSE FICHERO-MAESTRO-INVENTARIO
OPEN I-O FICHERO-MAESTRO-INVENTARIO
.....  

..... (use READ y REWRITE para actualizar registros lógicos de FICHERO-MAESTRO-INVENTARIO)
.....  

CLOSE FICHERO-MAESTRO-INVENTARIO

```

Después de cerrar un fichero puede volver a abrirse para continuar con su procesamiento. Aquí se abre un fichero para salida, se añaden registros y después se cierra. A continuación se abre para entrada, se leen los registros lógicos (por ejemplo, para imprimir un informe) y se vuelve a cerrar. Por último, se abre en modo I-O, se actualizan algunos registros y se cierra definitivamente.

#### EJEMPLO 6.5

PROCEDURE DIVISION.

```
OPEN I-O FICHERO-MAESTRO-EMPLEADOS
.....  
CLOSE FICHERO-MAESTRO-EMPLEADOS WITH LOCK
```

Cuando se cierra un fichero con la cláusula CLOSE WITH LOCK no puede volver a abrirse en la misma ejecución del programa. Por supuesto, el fichero puede abrirse por otros programas o incluso por el mismo programa al ejecutarlo otra vez.

#### Otras versiones de CLOSE...

Estas versiones se utilizan sólo con ficheros en cinta. Un carrete de cinta que almacena más de un fichero recibe el nombre de *carrete multifichero* (o *cinta o volumen*). Lleva una *etiqueta de volumen* que contiene el nombre del carrete (*nombre del volumen*) y otras informaciones relativas a toda la cinta. Cada fichero tiene dos *etiquetas de fichero*: *etiqueta de principio* y *etiqueta de fin* de fichero. La etiqueta de fin de fichero toma especial importancia cuando se lee de atrás para delante (OPEN INPUT REVERSED).

De modo análogo se entiende por *fichero multivolumen* uno que esté almacenado en más de un carrete de cinta (o disco).

#### EJEMPLO 6.6

CLOSE FICHERO-MAESTRO-EMPLEADOS WITH NO REWIND

Por lo general, cuando se cierra un fichero en cinta, ésta se rebobina hasta el principio del fichero. En este caso: (i) FICHERO-MAESTRO-EMPLEADOS es un fichero multivolumen y se desea dejar la cinta en dicha posición para la apertura del *siguiente* fichero en cinta; o (ii) FICHERO-MAESTRO-EMPLEADOS va a abrirse para INPUT REVERSED y, por tanto, se desea dejarlo posicionado al final.

#### EJEMPLO 6.7

PROCEDURE DIVISION.

```
OPEN OUTPUT FICHERO-COSTE-HISTORICO
.....  
WRITE REGISTRO-COSTE-HISTORICO
.....  
CLOSE FICHERO-COSTE-HISTORICO UNIT
.....  
WRITE REGISTRO-COSTE-HISTORICO
.....  
CLOSE FICHERO-COSTE-HISTORICO
```

El programa anterior crea un FICHERO-COSTE-HISTORICO multivolumen, que se abre en modo OUTPUT. La primera instrucción WRITE se utiliza para escribir registros en el fichero. La instrucción CLOSE... UNIT (o su equivalente CLOSE... REEL) no cierra realmente el fichero, sino que obliga a que la *siguiente* instrucción WRITE escriba el registro en el carrete o disco *siguiente*. (Esto se conoce como un *cambio de volumen*.) Observe que cuando se hayan escrito todos los registros lógicos el fichero se cierra normalmente.

En un fichero multivolumen abierto en modo INPUT, una instrucción READ después de CLOSE... UNIT leerá un registro del carrete o disco siguiente.

Si en el programa anterior se sustituyera la primera instrucción CLOSE por

CLOSE FICHERO-COSTE-HISTORICO UNIT FOR REMOVAL

el sistema operativo no sólo efectuará el cambio de volumen, sino que también "sabrá" que la unidad que se cierra no va a volver a utilizarse. De este modo el dispositivo de entrada/salida se libera para otros trabajos. La "unidad" cerrada *podría* volver a utilizarse si el fichero fuera cerrado y después reabierto.

#### Otras instrucciones de entrada/salida

Además de las instrucciones OPEN... y CLOSE..., se dispone de la instrucción REWRITE, que se describe en el Capítulo 11.

#### 6.4 ENTRADA: INSTRUCCION READ

La instrucción READ sólo puede utilizarse con los ficheros que hayan sido abiertos en modo INPUT o en modo I-O. Su formato general es:

READ nombre-de-fichero RECORD [INTO nombre-de-datos]  
[AT END instrucción-imperativa-1 [instrucción-imperativa-2] ....]

El nombre del fichero debe ser el mismo que se utilice en las instrucciones SELECT y FD relativas al mismo. Asumiendo, como siempre, que la organización es secuencial, la instrucción READ coloca el siguiente registro lógico en el área correspondiente (nivel 01 de la descripción en la instrucción FD). Si se hubiera definido un área para FILE STATUS, la instrucción READ modificará su contenido para reflejar el resultado de la ejecución.

**EJEMPLO 6.8** La sintaxis de READ... requiere el uso del nombre de un *fichero* y no del nombre de un registro. Para la instrucción WRITE... lo correcto es exactamente lo contrario (véase la Sección 6.5). Para evitar equivocaciones recuerde el eslogan de COBOL: "Lee el fichero, escribe el registro".

Al ejecutar READ cuando no hay más registros disponibles tiene lugar la condición *fin de fichero*. Las instrucciones de la cláusula AT END sólo se ejecuta cuando tiene lugar ésta condición; en otro caso, la computadora ignora esta cláusula. Cuando se alcanza el fin de fichero debe cerrarse y reabrirse antes de realizar cualquier otro procesamiento.

#### EJEMPLO 6.9

.....  
ENVIRONMENT DIVISION.

SELECT FICHERO-VENTAS ASSIGN TO VENTAS.

DATA DIVISION.  
FILE SECTION.

FD FICHERO-VENTAS  
BLOCK CONTAINS 0 RECORDS  
RECORD CONTAINS 39 CHARACTERS  
LABEL RECORDS ARE STANDARD

01	REGISTRO-VENTAS	
05	VENTAS-NOMBRE	PIC X(15).
05	VENTAS-MES-1	PIC S9(4)V99.
05	VENTAS-MES-2	PIC S9(4)V99.
05	VENTAS-MES-3	PIC S9(4)V99.
05	VENTAS-CUOTA	PIC S9(4)V99.

WORKING-STORAGE SECTION.

01	WORK-AREAS.	
05	INTER-FIN-FICHERO-VENTAS	PIC X.
05	TOTAL-TRIMESTRAL	PIC S9(5)V99 COMP-3.

PROCEDURE DIVISION.

```

OPEN INPUT FICHERO-VENTAS
MOVE "F" TO INTER-FIN-FICHERO-VENTAS
.
.
.
READ FICHERO-VENTAS
AT END
MOVE "T" TO INTER-FIN-FICHERO-VENTAS

IF INTER-FIN-FICHERO-VENTAS EQUAL "F"
ADD VENTAS-MES-1
VENTAS-MES-2
VENTAS-MES-3
GIVING TOTAL-TRIMESTRAL
.
.
```

Después de abrir el FICHERO-VENTAS en modo INPUT se inicializa con "F" (de falso) el INTER-FIN-FICHERO-VENTAS. Después el programa lee un registro lógico del FICHERO-VENTAS. Si no hubiera más registros lógicos, se ejecutaría la cláusula AT END colocando el valor "T" en INTER-FIN-FICHERO-VENTAS. Si la operación de lectura tiene éxito, se ignorará la cláusula AT END... y el interruptor seguirá valiendo "F". Observe que el programa comprueba si el FICHERO-VENTAS no está al final antes de efectuar la suma mediante la instrucción ADD... También merece especial atención el uso del punto estructurado para marcar el final de la cláusula AT END que es necesario, puesto que la computadora asumirá que la cláusula continúa hasta que no se encuentra un punto.

**EJEMPLO 6.10** Considere el segmento de programa

```

READ FICHERO-VENTAS RECORD
AT END
MOVE "T" TO INTER-FIN-FICHERO-VENTAS
MOVE CUOTA-VENTAS TO INFORME-CUOTA-VENTAS
MOVE "NO" TO BANDERA-ERROR-ENTRADA-VENTAS
WRITE...
.
```

En primer lugar, se aprecia que el programador se ha olvidado de incluir un punto al final de la cláusula AT END, haciendo, de este modo, inútiles las instrucciones MOVE... y WRITE... que siguen hasta que no tenga lugar la condición fin de fichero. En segundo término, quizás menos fácil de ver es que ha olvidado que si la instrucción READ... alcanza el fin de fichero, el área del registro lógico REGISTRO-VENTAS, contendrá basura; por tanto, sólo basura se trasladará a INFORME-CUOTA-VENTAS, causando una salida incorrecta y probablemente otros errores. *No procese un registro lógico después de que se ejecute AT END...* (a menos que se haya cerrado y reabierto el fichero).

#### Opción READ... INTO

La opción INTO... de la instrucción READ hace que el registro lógico al que se está dando entrada se copie desde el área del registro lógico en otro elemento de datos. Así,

READ FICHERO-VENTAS INTO COPIA-TRABAJO

y

```

READ FICHERO-VENTAS
MOVE REGISTRO-VENTAS TO COPIA-TRABAJO
.
```

son exactamente equivalentes. La opción READ... INTO se utiliza muchas veces para crear copias en el almacenamiento de trabajo de los registros lógicos (véanse los Problemas 5.97 y 5.98). En el COBOL ANS de 1974 esta opción sólo se puede utilizar con registros de longitud fija (véase el Apéndice C para el COBOL de 1980).

#### 6.5 SALIDA: INSTRUCCION WRITE

La instrucción WRITE sólo puede utilizarse con ficheros que hayan sido abiertos en modo OUTPUT, en modo I-O o en modo EXTEND. La ejecución de la instrucción libera el contenido del área del registro lógico (nivel 01) para salida y al mismo tiempo *cambia la referencia del nivel 01*

*del nombre del registro a un área del buffer sin usar, pero asociada con el fichero.* (Repase los conceptos “buffer múltiple” de la Sección 5.8 y “bloqueo” de la Sección 5.3; el nuevo segmento del buffer, que en este momento contiene basura, puede utilizarse para construir el *siguiente* registro lógico y darle salida.) Como consecuencia, el registro lógico que acaba de escribirse *no puede volver a ser accesible para el programa.*

#### La instrucción WRITE... para ficheros que no sean de impresión

En este caso la sintaxis es

WRITE nombre-registro-lógico [FROM nombre-datos]

La opción FROM se utiliza frecuentemente por los programadores que prefieren hacer todo el procesamiento de registros utilizando copias en el almacenamiento de trabajo. (Véase READ... INTO..., Sección 6.4.)

```
WRITE REGISTRO-FACTURA-A-PAGAR
      FROM WS-FACTURA-A-PAGAR
```

tiene la misma traducción al lenguaje máquina que

```
MOVE WS-FACTURA-A-PAGAR
      TO REGISTRO-FACTURA-A-PAGAR
      WRITE REGISTRO-FACTURA-A-PAGAR
```

El “nombre-registro-lógico” debe ser el nombre del nivel 01 de la instrucción FD correspondiente al fichero. Cuando se utiliza la opción FROM, el “nombre-datos” suele estar definido en el WORKING-STORAGE. Si se hubiera definido un área de FILE STATUS, la instrucción WRITE... situaría un código de 2 bytes en dicha área para reflejar el resultado de la ejecución.

#### EJEMPLO 6.11

```
FD FICHERO-FACTURAS-PENDIENTES
  BLOCK CONTAINS 0 RECORDS
  RECORD CONTAINS 32 CHARACTERS
  LABEL RECORDS ARE STANDARD

  01 REGISTRO-FACTURAS-PENDIENTES      PIC X(32).

FD FICHERO-FACTURAS-PAGADAS
  BLOCK CONTAINS 0 RECORDS
  RECORD CONTAINS 16 CHARACTERS
  LABEL RECORDS ARE STANDARD

  01 REGISTRO-FACTURAS-PAGADAS      PIC X(16).

WORKING-STORAGE SECTION.

  01 WS-REGISTRO-FACT-PENDIENTES.
    05 WS-FACT-PEND-CODIGO          PIC X.
    05 WS-FACT-PEND-VENDEDOR        PIC X(7).
    05 WS-FACT-PEND-NUMERO         PIC X(4).
    05 WS-FACT-PEND-FECHA          PIC 9(6).
    05 WS-FACT-PEND-VENCIMIENTO    PIC 9(6).
    05 WS-FACT-PEND-IMPORTE        PIC S9(5)V99  COMP-3.
    05 WS-FACT-PEND-DESCUENTO      PIC S9(5)V99  COMP-3.

  01 WS-REGISTRO-FACT-PAGADAS.
    05 WS-FACT-PAG-CODIGO          PIC X.
    05 WS-FACT-PAG-VENDEDOR        PIC X(7).
    05 WS-FACT-PAG-NUMERO         PIC X(4).
    05 WS-FACT-PAG-IMPORTE        PIC S9(5)V99  COMP-3.
```

## PROCEDURE DIVISION.

```

OPEN INPUT FICHERO-FACTURAS-PENDIENTES
      OUTPUT FICHERO-FACTURAS-PAGADAS
.....
READ FICHERO-FACTURAS-PENDIENTES
  INTO WS-REGISTRO-FACT-PENDIENTES
  AT END
    MOVE "T" TO WS-FIN-FICHERO

IF WS-FIN-FICHERO NOT EQUAL "T"
  MOVE WS-FACT-PEND-CODIGO           TO WS-FACT-PAG-CODIGO
  MOVE WS-FACT-PEND-VENDEDOR        TO WS-FACT-PAG-VENDEDOR
  MOVE WS-FACT-PEND-NUMERO         TO WS-FACT-PEND-NUMERO
  SUBTRACT WS-FACT-PEND-DESCUENTO
    FROM WS-FACT-PEND-IMPORTE
    GIVING WS-FACT-PAG-IMPORTE

  WRITE REGISTRO-FACT-PAGADAS FROM WS-REGISTRO-FACT-PAGADAS
.....
CLOSE FICHERO-FACTURAS-PENDIENTES
  FICHERO-FACTURAS-PAGADAS
.....

```

Este programa ilustra el procesamiento de ficheros utilizando copias de los registros lógicos en el almacenamiento de trabajo. Los registros REGISTRO-FACTURAS-PENDIENTES y REGISTRO-FACTURAS-PAGADAS se describen simplemente con PIC X(32) y PIC X(16), respectivamente, porque no van a procesarse en el área del registro lógico en el buffer. La instrucción READ FICHERO-FACTURAS-PENDIENTES INTO WS-REGISTRO-FACT-PENDIENTES se utiliza para dar entrada a un registro lógico y crear una copia en el almacenamiento de trabajo. Las instrucciones MOVE y SUBTRACT se utilizan para construir un registro lógico en WS-REGISTRO-FACT-PAGADAS. Después, la instrucción WRITE REGISTRO-FACTURAS-PAGADAS FROM WS-REGISTRO-FACT-PAGADAS mueve un registro del almacenamiento de trabajo al área del registro lógico en el buffer y lo libera para darle salida.

Después de la instrucción WRITE, el REGISTRO-FACTURAS-PAGADAS no vuelve a contener los datos a los que se ha dado salida; ahora se refiere al siguiente segmento vacío disponible en el buffer, que puede utilizarse para construir el *siguiente* registro lógico. Sin embargo, el WS-REGISTRO-FACT-PAGADAS *todavía contiene* una copia del registro lógico que acaba de escribirse; esto es otra de las ventajas de efectuar el procesamiento de los registros en el almacenamiento de trabajo:

Al igual que "READ... INTO...", "WRITE... FROM..." nunca debe usarse con registros de longitud variable (véase el Apéndice C para las consideraciones en el COBOL de 1980). El método del procesamiento de los registros en el almacenamiento temporal requiere parejas de instrucciones MOVE y WRITE:

```

MOVE WS-REGISTRO-FACT-PAGADAS TO REGISTRO-FACTURAS-PAGADAS
WRITE REGISTRO-FACTURAS-PAGADAS

```

**La instrucción WRITE... para ficheros de impresión**

En este caso la sintaxis aparece reflejada en la Figura 6-3. Las diversas opciones tienen todas que ver con el control de formateado de la salida.

WRITE nombre-registro [FROM identificador]

$\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\}$	ADVANCING	$\left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{entero} \\ \text{nombre-nemotécnico} \end{array} \right\} \\ \text{PAGE} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{LINE} \\ \text{LINES} \end{array} \right\}$
---	-----------	---	---

[AT {END-OF-PAGE} EOP] instrucción-imperativa-1 [instrucción-imperativa-2] ...]

Fig. 6-3

**EJEMPLO 6.12**

```

FD  FICHERO-TIEMPO-TARJETAS
BLOCK CONTAINS 0 RECORDS
RECORD CONTAINS 33 CHARACTERS
LABEL RECORDS ARE STANDARD

01  REGISTRO-TIEMPO-TARJETAS          PIC X(33).

FD  INFORME-TIEMPO-TARJETAS
RECORD CONTAINS 132 CHARACTERS
LABEL RECORDS ARE OMITTED

01  LINEA-TIEMPO-TARJETAS            PIC X(132).

WORKING-STORAGE SECTION.

01  WS-TIEMPO-TARJETAS-REG.
05  WS-TT-ID                      PIC X(6).
05  WS-TT-NOMBRE                  PIC X(20).
05  WS-TT-DEPARTAMENTO           PIC X(4).
05  WS-TT-HORAS                  PIC S99V9.

01  WS-TIEMPO-INFORME-LINEA
05  WS-TI-ID                      PIC X(6).
05  FILLER                         PIC X(4)    VALUE SPACES.
05  WS-TI-NOMBRE                  PIC X(20).
05  FILLER                         PIC X(10)   VALUE SPACES.
05  WS-TI-DEPARTAMENTO           PIC X(4).
05  FILLER                         PIC X(6)    VALUE SPACES.
05  WS-TI-HORAS                  PIC ZZ.9-.
05  FILLER                         PIC X(77)   VALUE SPACES.

01  NUMERO-PARA-SALTO             PIC S9(3*) COMP-3.

PROCEDURE DIVISION.

OPEN   INPUT    FICHERO-TIEMPO-TARJETAS
      OUTPUT   INFORME-TIEMPO-TARJETAS

READ FICHERO-TIEMPO-TARJETAS
INTO WS-TIEMPO-TARJETAS-REG
AT END .....

MOVE WS-TT-ID                      TO WS-TI-ID
MOVE WS-TT-NOMBRE                  TO WS-TI-NOMBRE
MOVE WS-TT-DEPARTAMENTO           TO WS-TI-DEPARTAMENTO
MOVE WS-TT-HORAS                  TO WS-TI-HORAS

WRITE LINEA-TIEMPO-TARJETAS FROM WS-TIEMPO-INFORME-LINEA
BEFORE ADVANCING 3 LINES

```

La instrucción "WRITE... BEFORE ADVANCING 3 LINES" hace que el contenido de WS-TIEMPO-INFORME-LINEA (a través de LINEA-TIEMPO-TARJETAS) se imprima en donde esté posicionada la impresora. A continuación se avanza tres líneas el papel. Por tanto, al ejecutar WRITE... una y otra vez, el resultado será que habrá dos líneas en blanco entre cada dos líneas impresas.

En este ejemplo se utiliza la técnica del procesamiento en el WORKING-STORAGE y se rellena de espacios las áreas FILLER (que separan los campos del informe) con la cláusula VALUE. Si se construyera LINEA-TIEMPO-TARJETAS en la sección de ficheros (donde la cláusula "VALUE..." es ilegal), se tendrían que usar instrucciones MOVE para llenar de espacios el área del registro lógico antes de mover la información que se vaya a imprimir.

Si se omitiera la cláusula "BEFORE ADVANCING 3 LINES" en este programa, se activaría el valor por defecto que es "AFTER ADVANCING 1 LINE" y no habría líneas en blanco entre las líneas impresas.

**EJEMPLO 6.13** Para ilustrar el uso de la opción

WRITE... BEFORE/AFTER ADVANCING identificador-2 LINES

suponga que en el Ejemplo 6.12 se define un elemento NUMERO-PARA-SALTO con un PICTURE numérico sin punto decimal. Entonces:

```
.....  
MOVE 3 TO NUMERO-PARA-SALTO  
.....  
WRITE LINEA-TIEMPO-TARJETAS  
    WS-TIEMPO-INFORME-LINEA  
    AFTER ADVANCING NUMERO-PARA-SALTO LINES
```

El contenido de NUMERO-PARA-SALTO puede ser un entero arbitrario inicializado desde el programa o resultado de un cálculo efectuado por el mismo. WRITE... AFTER ADVANCING... hace que el papel avance el número de líneas indicado y después se realice la impresión.

**EJEMPLO 6.14** La opción BEFORE/AFTER ADVANCING PAGE desplaza el papel hasta la parte superior de la siguiente página. Si se incluye la cláusula LINAGE en la instrucción FD correspondiente al fichero, el papel se posiciona en la parte superior del cuerpo de la página siguiente (véase la Sección 5.7). En otro caso, el papel se posiciona sobre el canal 1 del mecanismo de control de carro de la impresora (que generalmente se corresponde con la línea 1 de la siguiente página; véase la Sección 4.4).

**EJEMPLO 6.15** La cláusula "BEFORE/AFTER ADVANCING nombre-nemotécnico" obliga al papel a posicionarse en el canal de control del carro que se determina en el párrafo SPECIAL-NAMES de la división del entorno y configuración (repase la Sección 4.4).

```
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
SPECIAL-NAMES.  
    C01 IS PARTE-SUPERIOR-PÁGINA  
    C02 IS AREA-CABECERAS  
    C05 IS AREA-TOTALES
```

```
.....  
PROCEDURE DIVISION.  
    WRITE LINEA-INFORME  
        FROM WS-LINEA-NUM-PAGINA-FECHA  
        AFTER ADVANCING PARTE-SUPERIOR-PAGINA  
    WRITE LINEA-INFORME  
        FROM WS-LINEA-CABECERAS  
        AFTER ADVANCING AREA-CABECERAS  
    WRITE LINEA-INFORME  
        FROM WS-LINEA-TOTALES  
        AFTER ADVANCING AREA-TOTALES
```

Aquí, "AFTER ADVANCING PARTE-SUPERIOR-PAGINA" avanza el papel al canal 1 del control del carro, después imprime el contenido de WS-LINEA-NUM-PAGINA-FECHA, etc. *Antes de ejecutar el programa, el operador deberá posicionar la impresora de forma que los canales de control del carro utilizados por el programa coincidan con las partes apropiadas del papel continuo.* Si no se hace esto, el espaciado no se lleva a cabo correctamente.

Los canales de control del carro definidos en el párrafo SPECIAL-NAMES y la cláusula LINAGE de la instrucción FD son *mutuamente excluyentes*.

En la Figura 6-3, "AT END-OF-PAGE" y "AT EOP" son equivalentes. Se pueden utilizar sólo cuando se incluya la cláusula LINAGE en la instrucción FD; en tal caso, el compilador COBOL hará que el programa objeto mantenga una variable LINAGE-COUNTER que contará todas las líneas impresas o saltadas de una página lógica. Cuando se comienza una nueva página lógica, el LINAGE-COUNTER vuelve a tomar el valor 1 automáticamente.

Las instrucciones especificadas en AT END-OF-PAGE se ejecutan sólo cuando la ejecución de WRITE... BEFORE/AFTER ADVANCING... haga que el valor de LINAGE-COUNTER exceda (1) el valor especificado en WITH FOOTING AT... (pero no el tamaño del cuerpo de página prescrito en LINAGE IS...) o (2) el tamaño del cuerpo de página.

**Caso 1.** La instrucción WRITE se ejecuta normalmente y las instrucciones de la cláusula AT END-OF-PAGE se ejecutan a continuación, después continúa el orden normal de ejecución de instrucciones.

**Caso 2.** Si se especifica BEFORE ADVANCING..., la línea se imprime y después se avanza el papel al principio del cuerpo de la página siguiente; si se especifica AFTER ADVANCING..., se avanza el papel al principio del cuerpo de página siguiente y después se imprime la línea. Después se ejecutan las instrucciones de la cláusula AT END-OF-PAGE.

**EJEMPLO 6.16** El caso 2 se conoce como *desbordamiento automático* de página. Suponga que para el FICHERO-INFORME

```
LINAGE IS 60
  WITH FOOTING AT 53
  LINES AT TOP 3
  LINES AT BOTTOM 3
```

y que el papel está posicionado en la línea 52 del cuerpo de página. La ejecución de

```
WRITE LINEA-INFORME
  FROM WS-COMP-LINEA
  AFTER ADVANCING 10 LINES
  AT END-OF-PAGE
    MOVE WS-TOTAL TO WS-TOTAL-SALIDA
    WRITE LINEA-INFORME FROM WS-LINEA-TOTAL
      AFTER ADVANCING 5 LINES
```

imprime un total en el área de notas al pie en la parte inferior del cuerpo de página. Sin embargo, como "AFTER ADVANCING 10 LINES" ocasiona un revasamiento de página ( $52+10=62$ , que excede el tamaño del cuerpo de página dado en LINAGE 60), ocurre lo siguiente: (i) el papel avanza hasta la primera línea susceptible de impresión en la *siguiente* página lógica; (ii) se imprime el contenido de WS-COMP-LINEA; (iii) como ha tenido lugar un desbordamiento automático de página, la cláusula AT END-OF-PAGE... hace que el papel avance otras cinco líneas y se imprime el contenido de WS-LINEA-TOTAL.

Así pues, la línea de totales cae cerca de la parte superior del cuerpo de página, mezclada con líneas normales del informe —obviamente no era esto lo que se pretendía.

**EJEMPLO 6.17** El programador puede eliminar las dificultades que aparecen con la rutina de la cláusula AT END-OF-PAGE coordinando de forma conveniente los valores de BEFORE/AFTER ADVANCING... con el valor de la cláusula WITH FOOTING AT... y con la propia rutina AT END-OF-PAGE.

Supongamos que la cláusula LINAGE es la misma que en el Ejemplo 6.16. Supongamos también que en la parte superior del cuerpo de página se imprime una cabecera que deja el papel posicionado en la línea 10 del cuerpo de página. Por último, supongamos que se quiere imprimir las líneas del informe separadas por dos líneas en blanco. Considere la siguiente instrucción WRITE:

```
WRITE LINEA-INFORME
  FROM WS-COMP-LINEA
  AFTER ADVANCING 3 LINES
  AT END-OF-PAGE
    MOVE WS-TOTAL TO WS-TOTAL-SALIDA
    WRITE LINEA-INFORME
      FROM WS-LINEA-TOTAL
      AFTER ADVANCING 5 LINES
    WRITE LINEA-INFORME
      FROM WS-PRIMERA-LINEA-TITULOS
      AFTER ADVANCING PAGE
    WRITE LINEA-INFORME
      FROM WS-SEGUNDA-LINEA-TITULOS
      AFTER ADVANCING 9 LINES
```

La primera vez que se ejecute esta instrucción WRITE..., posiciona el papel al final de la cabecera (línea 10). Ahora (AFTER ADVANCING 3 LINES) la primera línea del informe caerá en la línea 13; la segunda en la línea 16,...; la decimocuarta en la línea 52. La decimoquinta ejecución de la instrucción WRITE traerá como consecuencia lo siguiente: (i) la decimoquinta línea del informe se imprimirá en la línea 55; (ii) como la variable LINAGE-COUNTER (55) excede el límite del área de notas al pie (WITH FOOTING AT 53), la rutina de la cláusula AT END-OF-PAGE hará avanzar el papel hasta la línea 60 (AFTER ADVANCING 5 LINES), se imprimirá el contenido de WS-LINEA-TOTAL en la línea 60 (última línea del cuerpo de página), se imprimirá la primera línea de títulos en la línea 1 del siguiente cuerpo de página (AFTER ADVANCING PAGE) y por último el resto del título se imprimirá en la línea 10 (AFTER ADVANCING 9 LINES desde la línea 1).

Observe que la rutina de la cláusula AT END-OF-PAGE no sólo imprime el total en el área de notas al pie de cada página, sino que también avanza el papel hasta la siguiente página lógica e imprime los títulos. El papel se posiciona en el extremo izquierdo de la línea 10 para que el ciclo vuelva a comenzar.

Como la cláusula AT END... (Sección 6.4), AT END-OF-PAGE... continúa hasta el siguiente punto.

#### EJEMPLO 6.18

```

READ FICHERO-CHEQUES-PEDIDOS
  AT END
    MOVE "T" TO INTER-FIN-FICHERO-CHEQUES
    MOVE ID-VENDEDOR-CP TO WS-ID-VENDEDOR
    MOVE IMPORTE-CP      TO WS-IMPORTE
    WRITE LINEA-CHEQUES-PEDIDOS
      FROM WS-LINEA
      AFTER ADVANCING 2 LINES
      AT END-OF-PAGE
        WRITE LINEA-CHEQUES-PEDIDOS
          FROM TITULO-CHEQUES-PEDIDOS
          AFTER ADVANCING PAGE
    ADD 1 TO NUMERO-CHEQUES-PEDIDOS

```

Este segmento de programa requiere un punto estructurado después de "MOVE "T"...", y otro después de "AFTER ADVANCING PAGE". Como puede apreciarse por el *desplazamiento de líneas*, la rutina de AT END se extiende hasta el final del segmento. Incluye, por tanto, la primera instrucción WRITE. Pero esta instrucción WRITE es *condicional*, porque contiene una cláusula (AT END-OF-PAGE) que se ejecuta sólo en determinadas condiciones. Por tanto, AT END... contiene un error sintáctico, puesto que sólo puede contener instrucciones *imperativas*, es decir, no condicionales (Sección 6.4).

## 6.6 ENTRADA: INSTRUCCION ACCEPT

La instrucción ACCEPT se utiliza sólo con ciertas entradas especiales.

#### Entrada de información desde el sistema operativo con ACCEPT

La sintaxis para la transferencia de información especial del sistema operativo al programa objeto COBOL es:

$$\text{ACCEPT identificativo } \text{FROM} \left\{ \begin{array}{c} \text{DATE} \\ \hline \text{DAY} \\ \hline \text{TIME} \end{array} \right\}$$

donde "identificativo" es un nombre definido adecuadamente en la división de datos.

#### EJEMPLO 6.19

- WORKING-STORAGE SECTION.  
01 FECHA-EJEMPLO-ACEPTADA      PIC 9(6).

```

PROCEDURE DIVISION.
  ACCEPT FECHA-EJEMPLO-ACEPTADA FROM DATE

```

El sistema operativo coloca el valor de la fecha actual en el elemento de datos FECHA-EJEMPLO-ACCEPTADA. La cláusula PICTURE adecuada para DATE es PIC 9(6). La fecha está en el formato aammdd; por ejemplo, el 12 de febrero de 1983 se representaría por el número 830212.

- 01 DIA-EJEMPLO-ACCEPTADO PIC 9(5).  
ACCEPT DIA-EJEMPLO-ACCEPTADO FROM DAY

El sistema operativo coloca la *fecha Juliana* en el elemento de datos DIA-EJEMPLO-ACCEPTADO. La cláusula PICTURE adecuada para DIA es PIC 9(5). La fecha Juliana es un número de 5 dígitos en la forma aadd, donde aa es el año y ddd es el día del año (donde los días se numeran de 1 a 365). El 12 de febrero de 1983 se representaría como 83043.

- 01 HORA-EJEMPLO-ACCEPTADA PIC 9(8).  
ACCEPT HORA-EJEMPLO-ACCEPTADA FROM TIME

El sistema operativo coloca la hora en el elemento de datos HORA-EJEMPLO-ACCEPTADA. La cláusula PICTURE adecuada para TIME es PIC 9(8). La hora es un número de 8 dígitos en la forma hhmmsscc, donde hh es la hora (de 0 a 23), mm es el minuto (de 0 a 59), ss es el segundo (de 0 a 59) y cc es la centésima de segundo (de 0 a 99). Por tanto, las 3:24 de la tarde se representarían como 15240000.

#### **Entrada de datos desde SYSIN o desde la consola del operador con ACCEPT**

La otra versión de la instrucción ACCEPT se puede utilizar para la entrada de pequeñas cantidades de información desde el fichero especial del sistema o, más frecuentemente, desde la consola del operador de la computadora. La sintaxis es:

ACCEPT identificador [FROM nombre-especial]  
SYSIN  
CONSOLE

El “nombre-especial” debe definirse en el párrafo SPECIAL-NAMES de la división del entorno y configuración (Sección 4.4). Los detalles exactos de esta versión de ACCEPT varían de un sistema a otro, pero en el COBOL del sistema IBM OS/VS las dos entradas en el párrafo SPECIAL-NAMES para ACCEPT son CONSOLE y SYSIN:

```
SPECIAL-NAMES.  
CONSOLE IS DISP-MENS-OPERADOR  
SYSIN    IS TARJETA-DE-CONTROL
```

CONSOLE es una palabra reservada de IBM que se refiere a la consola del operador de la computadora. La consola se usa para comunicarse con el sistema operativo y con los programas que se ejecutan *a través* de él. SYSIN es una palabra reservada de IBM que se refiere a un fichero especial de 80 bytes. Si el programador quiere hacer uso del fichero SYSIN, tendría que utilizar una instrucción del lenguaje de control de trabajos. En general, debe evitarse el uso de ACCEPT para acceder al fichero SYSIN a menos que la cantidad de información precisa sea muy pequeña. Es preferible preparar un fichero convencional de entrada con las instrucciones SELECT, FD, OPEN y READ.

#### **EJEMPLO 6.20**

```
SPECIAL-NAMES.  
CONSOLE IS DISP-MENS-OPERADOR  
.....  
WORKING-STORAGE SECTION.  
01 NUMERO-DE-ETIQUETAS PIC 9    DISPLAY.  
.....  
PROCEDURE DIVISION.  
    ACCEPT NUMERO-DE-ETIQUETAS FROM DISP-MENS-OPERADOR
```

Al imprimir etiquetas de correo, cheque y otros formularios pequeños con la impresora, se puede aumentar la velocidad de impresión considerablemente utilizando varios formularios por línea. El anterior segmento de programa ilustra el modo en que el operador del sistema podría indicar el número de etiquetas que deben

imprimirse a lo largo de la página. Cuando se ejecuta la instrucción ACCEPT, se suspende temporalmente la ejecución del resto del programa hasta que el operador escribe la información necesaria en el teclado terminal. Dicha información se sitúa en el elemento indicado en la instrucción ACCEPT (NUMERO-DE-ETIQUETAS). Como el dato procede de un terminal CRT o de impresión, su formato debe ser DISPLAY. *El operador de la computadora deberá escribir la información de acuerdo con la descripción que aparezca en la cláusula PICTURE del programa COBOL, en caso contrario se originará un error.*

**EJEMPLO 6.21** Uno de los escasos usos prácticos de ACCEPT para dar entrada desde SYSIN podría ser para leer una tarjeta de control. Las *tarjetas de control* son registros (a menudo perforadas en tarjetas) que especifican las opciones del programa que van a ejecutarse en un momento dado. Suponga que un programa que manipule etiquetas de correo pudiera clasificarlas por código postal o por orden alfabético y pueda opcionalmente incluir o excluir a los empleados de la compañía.

```

SPECIAL-NAMES.
  SYSIN IS DISPOSITIVO-ENTRADA-Opciones
  .....
  WORKING-STORAGE SECTION.
    01 OPCIONES-PARA-EJECUCION.
      05 TIPO-DE-CLASIFICACION      PIC X(4).
      05 INCLUSION-EMPLEADOS       PIC X(3).
  .....
  PROCEDURE DIVISION.
    ACCEPT OPCIONES-PARA-EJECUCION FROM DISPOSITIVO-ENTRADA-Opciones
  
```

Una tarjeta de control podría contener

$\overbrace{\text{N O M B R E N O } b}$	$\circ$	$\overbrace{\text{C O D P O S } b \text{ Sib}}$	
X(6)	X(3)	X(6)	X(3)

que harían que el programa clasifique las etiquetas por nombres con exclusión de los empleados o clasificar por código postal incluyendo a los empleados. Cada ejecución de ACCEPT... daría entrada a otro registro lógico del fichero SYSIN. Observe que aunque los registros de SYSIN tienen 80 bytes, no hay que tener en cuenta los restantes  $80 - (4+3) = 73$  bytes en la descripción de OPCIONES-PARA-EJECUCION.

## 6.7 SALIDA: INSTRUCCION DISPLAY

La instrucción DISPLAY... es adecuada para dar salida a información poco voluminosa hacia los ficheros especiales del sistema o la consola del operador de la computadora. Al igual que sucede con su homóloga ACCEPT, los detalles exactos varían de un sistema a otro. Para el COBOL del sistema IBM OS/VS la sintaxis de esta instrucción es

$$\text{DISPLAY} \left\{ \begin{matrix} \text{identificador-1} \\ \text{literal-1} \end{matrix} \right\} \left[ \begin{matrix} \text{identificador-2} \\ \text{literal-2} \end{matrix} \right] \dots [\text{UPON nombre-especial}]$$

donde "nombre-especial" debe ser definido en el párrafo SPECIAL-NAMES (Sección 4.4). Las entradas válidas en SPECIAL-NAMES para DISPLAY son:

```

SPECIAL-NAMES.
  CONSOLE IS...
  SYSOUT IS...
  
```

Aquí, como en la Sección 6.6, "CONSOLE" es la consola del operador y "SYSOUT" es un fichero de salida especial del sistema análogo a "SYSIN".

**EJEMPLO 6.22** DISPLAY se puede utilizar (en conjunción con ACCEPT) para conversar con el operador a través de la consola:

```

SPECIAL-NAMES.
  CONSOLE IS DISPOSITIVO-MENS-OPERADOR
  WORKING-STORAGE SECTION.
  
```

```
01 WS-RESPUESTAS-OPERADOR.
  05 WS-NUMERO-DE-ETIQUETAS-LINEA    PIC 9.
  05 WS-INCLUSION-EMPLEADOS          PIC X(3).
```

PROCEDURE DIVISION.

DISPLAY "POR FAVOR, FIJE EL NUMERO DE ETIQUETAS POR LINEA"

UPON DISPOSITIVO-MENS-OPERADOR

DISPLAY "ESCRIBA Dicho NUMERO (1 DIGITO)"

UPON DISPOSITIVO-MENS-OPERADOR

ACCEPT WS-NUMERO-DE-ETIQUETAS-LINEA  
FROM DISPOSITIVO-MENS-OPERADOR

DISPLAY "SE INCLUYEN EMPLEADOS (SI O NO)"  
UPON DISPOSITIVO-MENS-OPERADOR

ACCEPT WS-INCLUSION-EMPLEADOS  
FROM DISPOSITIVO-MENS-OPERADOR

Los *mensajes* impresos en la consola del operador le facilitan la realización de ciertas acciones y la introducción de la información que requieran las instrucciones ACCEPT.

Debe tenerse en cuenta que cuando se escriben mediante DISPLAY literales no numéricos en un dispositivo que admite un número de caracteres por línea (digamos 100), la salida se dividirá en líneas de 100 caracteres rompiendo las palabras donde cae. En el caso de constantes figurativas se imprime un carácter ("DISPLAY SPACES" imprime *un* espacio). La instrucción DISPLAY convierte automáticamente los elementos en formato COMP-3 o COMP a formato DISPLAY. En el COBOL del sistema IBM OS/VS se eliminan automáticamente los signos +, pero no los signos -. (véase la Tabla 5.1). No se imprimen los puntos decimales definidos con PIC "V".

**EJEMPLO 6.23** Un segundo uso importante de DISPLAY es en las *líneas de depuración* (véase la Sección 4.2):

```
SPECIAL-NAMES.  
D      SYSOUT IS DISPOSITIVO-DEPURACION
```

PROCEDURE DIVISION.

```
D      DISPLAY "EL VALOR INICIAL DE WS-NUM-DE-CLIENTES ES "
D      WS-NUM-DE-CLIENTES
D      UPON DISPOSITIVO-DEPURACION
```

```
.....  
D      DISPLAY "SI ID ES NULO, EL NOMBRE ES "
D      NOMBRE-MAESTRO-EMPLEADOS
D      UPON DISPOSITIVO-DEPURACION
```

Cuando en el párrafo SOURCE-COMPUTER se especifica WITH DEBUGGING MODE, las líneas de depuración (tienen una "D" en la columna 7) se traducen al programa objeto y hacen que se imprima información valiosa sobre el contenido de los datos en el fichero SYSOUT. Observe que los literales no numéricos incluyen un espacio extra (antes del cierre de las comillas) para separar los elementos en la línea de DISPLAY; la instrucción DISPLAY no proporciona automáticamente espacios de separación.

**EJEMPLO 6.24** El tercer uso de DISPLAY es para imprimir mensajes de error en el fichero SYSOUT de forma que se puedan conocer los errores o condiciones excepcionales que se han dado durante la ejecución del programa. (No debe hacerse esto cuando se espera un alto número de errores. Los datos que se introducen por el teclado se procesan mediante un *programa validador* cuya función principal consiste en detectar los errores en la entrada de datos e imprimir un informe de errores con suficiente información como para que puedan corregirse. En este caso, los errores deben imprimirse en un fichero convencional declarado mediante SELECT y FD.)

```
SPECIAL-NAMES.  
SYSOUT IS MENSAJE-ERROR-LOGICO
```

PROCEDURE DIVISION.

```

OPEN INPUT FICHERO-CHEQUES-PEDIDOS
IF COD-ESTADO-CHEQUES-PEDIDOS NOT EQUAL "00"
    DISPLAY "NO SE PUEDE ABRIR EL FICHERO"
    UPON MENSAJE-ERROR-LOGICO

```

Si no se elige la opción UPON en la instrucción DISPLAY, en el COBOL del sistema IBM OS/VS se asume por defecto.

DISPLAY... UPON SYSOUT

## 6.8 PROCESAMIENTO: INSTRUCCION MOVE

La instrucción "MOVE" induce a confusión porque su efecto es *copiar* el contenido de un elemento de datos en otro u otros. El *campo de envío* no modifica su valor por la ejecución de una instrucción MOVE. Los *campos de recepción* sustituyen sus contenidos anteriores por el contenido del campo de envío. La instrucción MOVE también *convierte* los datos desde el formato especificado en la cláusula USAGE para el campo de envío al formato especificado en la cláusula USAGE para el campo de recepción en el caso de que ambos sean diferentes. Más aún, si las longitudes de los campos de envío y recepción son distintas, MOVE producirá el *truncamiento* o *rellenado* precisos. Por último, la instrucción MOVE hará que los datos numéricos se *editen* si la cláusula PICTURE del campo de recepción contiene caracteres de edición.

La instrucción MOVE tiene la siguiente sintaxis:

MOVE {identificador-1} TO identificador-2 [identificador-3]...

(Una sintaxis alternativa se menciona en el Ejemplo 6.31, pero no será tratada en este libro.)

### EJEMPLO 6.25 Dado

```

01 ELEMENTOS-EJEMPLO-MOVE.
  05 A PIC X(5).
  05 B - PIC X(5).
  05 C PIC X(3).
  05 D PIC X(7).

```

suponga que el campo A contiene "ADIOS". Entonces:

- MOVE A TO B  
copia el contenido del elemento A ("identificador-1") en el elemento B ("identificador-2"). Puesto que A y B tienen los mismos formatos PIC y USAGE, ambos contendrán "ADIOS" después de la ejecución de MOVE. El anterior contenido de B se pierde.
- MOVE A TO C  
copia el contenido del elemento A en el elemento C. Como C es PIC X(3), el valor "ADIOS" se trunca por la derecha: después de la ejecución de MOVE, A contendrá "ADIOS" y C contendrá "ADI".
- MOVE A TO D  
copia el contenido del elemento A en el elemento D. Como D es PIC X(7), el valor de "ADIOS" se rellena con blancos por la derecha: después de la ejecución de MOVE, A contendrá "ADIOS" y D contendrá "ADIOSbb".
- MOVE A TO B C D  
copia el contenido del elemento A en los elementos B, C y D. Genera exactamente la misma traducción al lenguaje máquina que las tres instrucciones MOVE precedentes.
- MOVE SPACES TO A  
copia 5 espacios en blanco (aquí, "literal" es la constante figurativa SPACES) en el elemento A ("identificador-2"), que está definido como PIC X(5).
- MOVE ZEROS TO A C  
hace que A tome el valor "00000" y que C tome el valor "000", estando los ceros en formato DISPLAY.
- MOVE ALL "-" TO C D  
coloca en C el valor "\_\_\_" y en D el valor "\_\_\_\_\_". De nuevo en esta ocasión el uso de la constante figurativa (ALL "-") proporciona automáticamente el número correcto de caracteres.

**EJEMPLO 6.26** Dado

```
01 EJEMPLO-ELEM-NUMERICOS
 05 A PIC S9(5) VALUE+12345.
 05 B PIC S9(5).
 05 C PIC S9(3).
 05 D PIC S9(7).
```

## • MOVE A TO B

Como A y B son ambos PIC S9(5) y DISPLAY, contendrán el mismo valor después de la ejecución de MOVE, es decir: +12345.

## • MOVE A TO C

Con los elementos numéricos el truncamiento se produce por la izquierda. C contendrá +345 [PIC S9(3)] y A permanecerá igual.

## • MOVE A TO D

Con los elementos numéricos el relleno se produce por la izquierda y con ceros. A permanecerá inalterado y D contendrá +0012345.

**EJEMPLO 6.27** Dado

```
01 EJEMPLO-ELEM-NUMERICOS.
 05 A PIC S9(3)V99      COMP-3      VALUE +123.45.
 05 B PIC S9(3)V99      DISPLAY.
 05 C PIC S9(3)V99      COMP.
 05 D PIC S9(5)V99      COMP-3.
 05 E PIC S9(5)V9       DISPLAY.
 05 F PIC S9(5)V999     COMP.
 05 G PIC S9(2)V9       COMP-3.
```

## • MOVE A TO B

A y B tienen la misma PICTURE, pero distinto formato USAGE. El valor en A que es COMP-3 se convierte a formato DISPLAY al copiarse en B. Así, A y B contendrán +123.45 después de la ejecución de MOVE, pero el valor almacenado en A está en formato COMP-3, mientras que el valor en B está almacenado en formato DISPLAY.

## • MOVE A TO D

En este caso, el formato USAGE es el mismo, pero las cláusulas PICTURE son diferentes. Los puntos decimales están alineados y el campo de recepción se rellena con ceros por la izquierda. A permanecerá inalterado; D contendrá +00123.45 en formato COMP-3. Los puntos decimales entre los campos de recepción y envío permanecen siempre alineados.

## • MOVE A TO E

Se producirá una conversión automática desde COMP-3 a DISPLAY. Además, la parte entera del valor se rellenará con ceros (por la izquierda). Como E tiene sólo un decimal, su contenido será +00123.4 en formato DISPLAY después de la ejecución de MOVE. Observe que la instrucción MOVE no realiza ningún redondeo y que los puntos decimales permanecen alineados.

## • MOVE A TO F

En este caso se produce una conversión automática de COMP-3 a COMP. La parte entera del valor se rellena con ceros (por la izquierda). Puesto que el campo de recepción contiene más decimales que el campo de envío, el campo de recepción también se rellena con ceros (por la derecha). A permanecerá igual y F contendrá +00123.450 en formato COMP.

## • MOVE A TO G

Los dos formatos USAGE son COMP-3; por tanto, no se produce conversión. Sin embargo, como G es PIC S9(2)V9 se produce truncamiento por la izquierda y por la derecha. El contenido de G será +23.4 en formato COMP-3.

## • MOVE A TO B D E F G

Esta instrucción se traduce al lenguaje máquina del mismo modo que todas las instrucciones anteriores de este ejemplo.

## • MOVE ZEROS TO A B C

Como A, B y C tienen la misma PICTURE, su nuevo contenido será +000.00. Sin embargo, cada campo contendrá *distintos tipos* de ceros: A estará en formato COMP-3, B en formato DISPLAY y C en formato COMP. Cuando se utilizan constantes figurativas en la instrucción MOVE o en la cláusula VALUE, siempre representan un *número correcto* de datos y también un formato *USAGE correcto*.

**EJEMPLO 6.28** Dado

```
01 EJEMPLO-ELEM-NUMERICO.
  05 A          PIC S9(5)V99      VALUE +00010.08.
  05 A-EDITADO  PIC $$,$$$,.99-.
```

- MOVE A TO A-EDITADO

Cuando se mueve un elemento numérico a un campo con caracteres de edición, parte de la ejecución de MOVE consiste en la edición del valor en el campo de recepción. El valor en A permanecerá inalterado; el valor de A-EDITADO después de la ejecución de MOVE será:

*bbbb\$10.08b*

**EJEMPLO 6.29 (instrucciones MOVE inválidas)** COBOL clasifica los datos en seis *categorías de datos*. Dando valores a “identificativo-1” e “identificativo-2” se obtienen distintos modelos de instrucciones MOVE, de ellos los 14 de la lista siguiente son inválidos:

- MOVE DATO-ALFABETICO TO DATO-NUMERICO-ENTERO
- MOVE DATO-ALFABETICO TO DATO-NUMERICO-NO-ENTERO
- MOVE DATO-ALFABETICO TO DATO-NUMERICO-EDITADO
- MOVE DATO-ALFANUMERICO-EDITADO TO DATO-NUMERICO-ENTERO
- MOVE DATO-ALFANUMERICO-EDITADO TO DATO-NUMERICO-NO-ENTERO
- MOVE DATO-ALFANUMERICO-EDITADO TO DATO-NUMERICO-EDITADO
- MOVE DATO-NUMERICO-ENTERO TO DATO-ALFABETICO
- MOVE DATO-NUMERICO-NO-ENTERO TO DATO-ALFABETICO
- MOVE DATO-NUMERICO-NO-ENTERO TO DATO-ALFANUMERICO
- MOVE DATO-NUMERICO-NO-ENTERO TO DATO-ALFANUMERICO-EDITADO
- MOVE DATO-NUMERICO-EDITADO TO DATO-ALFABETICO
- MOVE DATO-NUMERICO-EDITADO TO DATO-NUMERICO-ENTERO
- MOVE DATO-NUMERICO-EDITADO TO DATO-NUMERICO-NO-ENTERO
- MOVE DATO-NUMERICO-EDITADO TO DATO-NUMERICO-EDITADO

Observe que todas las combinaciones ilegales se producen entre un elemento alfábético/alfanumérico/editado y un dato numérico. El resto de las combinaciones son válidas.

**EJEMPLO 6.30** Hasta el momento, todas las instrucciones MOVE se han producido con elementos simples. Considere la siguiente instrucción MOVE con elementos compuestos:

```
01 REGISTRO-ENTRADA-FACTURA.
  05 ID-VENDEDOR-ENTRADA      PIC X(5).
  05 ID-FACTURA-ENTRADA      PIC X(4).
  05 NUM-ELEMENTOS-ENTRADA   PIC S9(4).
  05 IMPORTE-ENTRADA         PIC S9(5)V99.

01 REGISTRO-FACTURA-PAGADA.
  05 ID-VENDEDOR-PAGADA       PIC X(5).
  05 ID-FACTURA-PAGADA       PIC X(4).
  05 NUM-ELEMENTOS-PAGADA    PIC S9(4)      COMP-3.
  05 IMPORTE-PAGADA          PIC S9(5)V99   COMP-3.
```

MOVE REGISTRO-ENTRADA-FACTURA TO REGISTRO-FACTURA-PAGADA

Cuando un elemento compuesto (de cualquier nivel) participa en una instrucción MOVE como campo de envío o de recepción, se trata como si fuera un elemento alfanumérico PIC X. En este caso, todos los campos del REGISTRO-ENTRADA-FACTURA están en formato DISPLAY; por tanto, la longitud del elemento compuesto participa en MOVE como si fuera PIC X(20). El REGISTRO-FACTURA-

PAGADA tiene 16 bytes ( $5+4+3+4$ ; observe la diferencia ocasionada por el uso del formato COMP-3 con los elementos numéricos); por tanto, a los efectos de la instrucción MOVE se trata como si estuviera definido mediante PIC X(16). La ejecución de la instrucción MOVE (i) no convierte los subcampos NUM-ELEMENTOS-ENTRADA ni IMPORTE-ENTRADA de DISPLAY a COMP-3 antes de copiarlos en NUM-ELEMENTOS-PAGADA e IMPORTE-PAGADA, y (ii) trunca los cuatro bytes más a la derecha de REGISTRO-ENTRADA-FACTURA [para pasar de PIC X(20) a PIC X(16)]. Como resultado se introducirá basura en los elementos NUM-ELEMENTOS-PAGADOS e IMPORTE-PAGADO, lo que hará abortar el programa si se utilizan dichos campos.

**EJEMPLO 6.31** Como consecuencia del Ejemplo 6.30 inducimos que utilizar MOVE con elementos compuestos no es bueno, a menos que:

- (1) todos los campos en los elementos de envío y recepción tengan formato DISPLAY y los efectos producidos por las distintas longitudes sean conocidos y buscados por el programador;
- o bien
- (2) no todos los campos sean DISPLAY, pero los elementos de envío y recepción se correspondan exactamente en término de la posición de sus subcampos y de las cláusulas PICTURE y USAGE.

Consideré el elemento compuesto

01	REGISTRO-FACTURA-PENDIENTES	
05	ID-FACTURA-PENDIENTES	PIC X(4).
05	ID-VENDEDOR-PENDIENTES	PIC X(5).
05	IMPORTE-PENDIENTES	PIC S9(5)V99.
05	NUM-ELEMENTOS-PENDIENTES	PIC S9(4).
01	REGISTRO-FACTURA-PAGADAS	
05	ID-VENDEDOR-PAGADAS	PIC X(5).
05	IMPORTE-PAGADAS	PIC S9(5)V99.
05	ID-FACTURA-PAGADAS	PIC X(4).
05	NUM-ELEMENTOS-PAGADAS	PIC S9(4).

Como no se satisface la condición (1),

MOVE REGISTRO-FACTURA-PENDIENTES TO REGISTRO-FACTURA-PAGADAS

no funcionará. Como consecuencia del distinto orden de los subcampos, el campo ID-FACTURA-PENDIENTES (4 bytes) y el primer byte de ID-VENDEDOR-PENDIENTES se copiarán en ID-VENDEDOR-PAGADAS (5 bytes); etc. Esto ocasionaría errores al hacer uso de los campos del REGISTRO-FACTURA-PAGADAS.

La única forma de obtener el resultado que se desea es usar MOVE con los subcampos individuales:

MOVE	ID-FACTURA-PENDIENTES	TO	ID-FACTURA-PAGADAS
MOVE	ID-VENDEDOR-PENDIENTES	TO	ID-VENDEDOR-PAGADAS
MOVE	IMPORTE-PENDIENTES	TO	IMPORTE-PAGADAS
MOVE	NUM-ELEMENTOS-PENDIENTES	TO	NUM-ELEMENTOS-PAGADAS

(En algunos sistemas se pueden efectuar ciertas instrucciones MOVE con elementos compuestos con

MOVE CORRESPONDING identificador-1 TO identificador-2

Se recomienda utilizar MOVE con los subcampos individuales.)

## 6.9 FIN DE PROGRAMA: INSTRUCCION STOP

### STOP RUN

La instrucción STOP RUN hace que el programa objeto devuelva el control al sistema operativo. Con ella se concluye definitivamente la ejecución del programa. *Todos los ficheros deben cerrarse con CLOSE antes de ejecutar STOP RUN*; en caso contrario, podrían tener lugar serios errores durante el posterior procesamiento de los ficheros. Cada programa debe tener al menos una instrucción STOP RUN y en muchas instalaciones se exige como norma que sólo haya una.

**STOP literal**

Esta versión de la instrucción STOP (i) hace que el valor del “literal” se escriba en la consola del operador de la computadora, (ii) *suspende temporalmente* la ejecución del programa objeto (mientras que se cambian las cintas, el papel, etc.) hasta que el operador indique al sistema operativo, a través de la consola, que la ejecución debe continuar.

**EJEMPLO 6.32**

```
PROCEDURE DIVISION.
  OPEN INPUT FICHERO-FACTURA-PAGADAS
  STOP "POR FAVOR, MONTE EL PAPEL DE CHEQUES"
  OPEN OUTPUT FICHERO-CHEQUES-FACT-PAGADAS
```

Después de leer el mensaje “POR FAVOR, MONTE EL PAPEL DE CHEQUES” en la consola, el operador probablemente cambiará el papel y continuará el programa en “OPEN OUTPUT...”.

### **6.10 EJECUCION CONTROLADA DE UN PARRAFO: INSTRUCCION PERFORM**

La instrucción PERFORM hace que se ejecute un párrafo completo una o varias veces desde un punto del programa que no coincide con aquel desde el cual se ejecutaría en la secuencia normal de instrucciones. La versión simple de la instrucción PERFORM es:

PERFORM nombre-de-párrafo

donde “nombre-de-párrafo” es un nombre de párrafo definido por el programador en la división de procedimientos. La ejecución de esta versión hace que se lleven a cabo *todas las instrucciones* del párrafo en la secuencia normal (de la primera a la última). Después se continúa con la instrucción siguiente a PERFORM.

**EJEMPLO 6.33**

```
PROCEDURE DIVISION.

  OPEN   INPUT      FICHERO-TIEMPO-TARJETAS
        OUTPUT     LISTADO-TIEMPO-TARJETAS

  PERFORM INICIA-CONTADORES-INTER

  WRITE LINEA-LISTADO
    FROM WS-LINEA-CABECERA
    AFTER ADVANCING PAGE

  PERFORM ENTRADA-REGISTRO-TIEMPO

  MOVE WS-TIEMPO-TARJETA-ID TO WS-LISTADO-ID
  .....
  INICIA-CONTADORES-INTER.

  MOVE "F" TO INTER-FIN-TIEMPO-TARJETA
  MOVE ZEROS TO WS-CONTADOR-EMPLEADOS
    WS-IMPORTE-HORAS-EXTRA
    WS-TOTAL-HORAS
```

## ENTRADA-REGISTRO-TIEMPO.

```

READ FICHERO-TIEMPO-TARJETAS
  INTO WS-REGISTRO-TIEMPO-TARJETAS
  AT END
    MOVE "T" TO INTER-FIN-TIEMPO-TARJETAS

```

Debido a la presencia de las dos instrucciones PERFORM, el orden de ejecución será: (1) OPEN..., (2) PERFORM INICIA-CONTADORES-INTER, (3) MOVE "F"..., (4) MOVE ZEROS..., (5) WRITE..., (6) PERFORM ENTRADA-REGISTRO-TIEMPO, (7) READ..., (8) MOVE WS-....

La mayor parte de los programas de aplicación necesitan repetir el ciclo entrada-procesamiento-salida para cada registro lógico del fichero de entrada. Esto implica que la *lógica del programa* debe contener un *bucle* por el que se ejecutan una serie de instrucciones una y otra vez. La sintaxis de la instrucción más útil para la creación de bucles en COBOL es

PERFORM nombre-de-párrafo UNTIL condición

Con una instrucción como ésta, se ejecutará el párrafo indicado una y otra vez hasta que se cumpla la “condición” especificada. Las “condiciones” se estudiarán en el Capítulo 7; por ahora, una condición es de la forma “nombre de dato EQUAL literal”.

## EJEMPLO 6.34

## PROCEDURE DIVISION.

```

OPEN   INPUT      FICHERO-TIEMPO-TARJETAS
       OUTPUT     LISTADO-TIEMPO-TARJETAS

MOVE "F" TO INTER-FIN-TIEMPO-TARJETAS

PERFORM PRODUCE-LINEA-LISTADO
  UNTIL INTER-FIN-TIEMPO-TARJETAS EQUAL "T"

CLOSE FICHERO-TIEMPO-TARJETAS
      LISTADO-TIEMPO-TARJETAS

STOP RUN

```

## PRODUCE-LINEA-LISTADO.

```

READ FICHERO-TIEMPO-TARJETAS
  INTO WS-REGISTRO-TIEMPO
  AT END
    MOVE "T" TO INTER-FIN-TIEMPO-TARJETAS

```

IF INTER-FIN-TIEMPO-TARJETAS EQUAL "F"

```

MOVE WS-REG-TIEMPO-ID      TO WS-LISTADO-ID
MOVE WS-REG-TIEMPO-NOMBRE  TO WS-LISTADO-NOMBRE
MOVE WS-REG-TIEMPO-HORAS   TO WS-LISTADO-HORAS

```

```

WRITE LINEA-LISTADO-TIEMPO-TARJETA
  FROM WS-LINEA-LISTADO
  AFTER ADVANCING 2 LINES

```

Supongamos que sólo hay dos registros lógicos en el FICHERO-TIEMPO-TARJETAS. El orden lógico de ejecución de las instrucciones será:

- |   |  |
|---|--|
| <b>1.<sup>er</sup><br/>PERFORM<br/>de<br/>PRODUCE-<br/>LINEA-<br/>LISTADO</b> | 1. OPEN<br>2. MOVE "F" TO INTER-FIN-TIEMPO-TARJETAS<br>3. PERFORM PRODUCE-LINEA-LISTADO<br><i>(Como INTER-FIN-TIEMPO-TARJETAS no es "T" se ejecuta el párrafo la primera vez.)</i><br>4. READ FICHERO-TIEMPO-TARJETAS... AT END MOVE "T" TO INTER-FIN-TIEMPO-TARJETAS<br><i>(Entrada del primer registro lógico; el interruptor sigue siendo "F".)</i><br>5. IF INTER-FIN-TIEMPO-TARJETAS EQUAL "F"<br><i>(Como la condición es verdadera, se ejecutan todas las instrucciones hasta el punto.)</i><br>6. MOVE WS-REG-TIEMPO-ID...<br>7. MOVE WS-REG-TIEMPO-NOMBRE...<br>8. MOVE WS-REG-TIEMPO-HORAS...<br>9. WRITE LINEA-LISTADO-TIEMPO-TARJETA FROM...<br><i>(Después de la última instrucción del párrafo, se evalúa la condición. Como el interruptor no vale "T", se vuelve a ejecutar el párrafo.)</i><br>10. READ FICHERO-TIEMPO-TARJETAS... AT END MOVE "T" TO INTER-FIN-TIEMPO-TARJETAS<br><i>(Entrada del segundo registro lógico; el interruptor sigue valiendo "F".)</i><br>11. IF INTER-FIN-TIEMPO-TARJETAS EQUAL "F"<br><i>(Como la condición es cierta, se ejecutan todas las instrucciones hasta el punto.)</i><br>12. MOVE WS-REG-TIEMPO-ID...<br>13. MOVE WS-REG-TIEMPO-NOMBRE...<br>14. MOVE WS-REG-TIEMPO-HORAS...<br>15. WRITE LINEA-LISTADO-TIEMPO-TARJETA FROM...<br><i>(Después de la última instrucción del párrafo, se evalúa la condición. Como el interruptor no vale "T", se vuelve a ejecutar el párrafo.)</i><br>16. READ FICHERO-TIEMPO-TARJETAS... AT END MOVE "T" TO INTER-FIN-TIEMPO-TARJETAS<br><i>(En este tercer intento de leer en un fichero que sólo tiene dos registros lógicos, tiene lugar la condición fin de fichero. Se ejecuta AT END... haciendo que el interruptor tome el valor "T"; PERFORM continuará hasta el final del párrafo.)</i><br>17. IF INTER-FIN-TIEMPO-TARJETAS EQUAL "F"<br><i>(El interruptor vale ahora "T", por lo que se saltan todas las instrucciones hasta el punto que marca el final de IF... se saltan. Este punto marca también el final del párrafo.)</i><br>18. CLOSE...<br><i>(Después de ejecutar la última instrucción del párrafo de PERFORM, la computadora examina el interruptor. Como ahora contiene "T", se detiene PERFORM y se pasa a la siguiente instrucción, CLOSE...)</i><br>19. STOP RUN<br><i>(Después de STOP RUN se detiene la ejecución del programa.)</i> |
| <b>2.<sup>o</sup><br/>PERFORM</b>   |  |
| <b>3.<sup>er</sup><br/>PERFORM</b>  |  |

Observe cómo la lógica del programa evita cuidadosamente el procesamiento de un registro después de que haya tenido lugar la condición de fin de fichero (Ejemplo 6.10).

**EJEMPLO 6.35** Despues de conocer los bucles, se está en disposición de examinar un programa COBOL completo. Considere una aplicación de nóminas en la que los empleados llenan sus tarjetas de tiempo semanal con su número de identificación, nombre y el total de horas trabajadas en la semana. Estas tarjetas de tiempo son examinadas por un supervisor y después enviadas al departamento de proceso de datos, donde la información se introduce en el disco mediante teclados. Se desea disponer de un *programa de edición* que imprima en papel un listado de los registros de tiempo semanal para su inspección y corrección.

A continuación se da el listado del programa fuente para esta aplicación. Se espera que el lector identifique los principios de programación vistos hasta el momento. En particular:

1. Cláusulas de ORGANIZATION y ACCESS en las instrucciones SELECT (redundantes porque el valor por defecto es SEQUENTIAL) que se incorporan para mejorar la documentación del programa.
2. Uso de la cláusula de IBM "BLOCK CONTAINS 0..." para el fichero de entrada en disco.
3. Se omite la cláusula BLOCK CONTAINS para el fichero de impresión.
4. Para el fichero en disco se especifica LABEL RECORDS ARE STANDARD, para el fichero de impresión OMITTED.
5. Todo el procesamiento de registros se lleva a cabo en el almacenamiento de trabajo utilizando "READ... INTO..." y "WRITE... FROM..."; las descripciones de registros en la sección de ficheros se limitan a PIC X(80) y PIC X(132), el verdadero contenido se describe para la copia en el almacenamiento de trabajo.
6. Los elementos del WORKING-STORAGE se agrupan bajo categorías al nivel 01; los elementos se clasifican de manera sistemática.
7. La cláusula VALUE se utiliza para inicializar las dos líneas de cabeceras y las líneas ID nombre horas en el almacenamiento de trabajo.
8. Con los elementos que no son utilizados en la división de procedimientos se usa "FILLER".
9. En la división de procedimientos se utilizan abundantemente PERFORM... y PERFORM... UNTIL... para controlar el flujo lógico del programa y crear el bucle entrada-procesamiento-salida.
10. Compruebe por usted mismo que este programa no procesa registros con basura después de alcanzar el fin de fichero (y que *procesa todos* los registros lógicos).
11. Observe el desplazamiento de las líneas en la división de procedimientos y la alineación por columnas en la división de datos.

```

00001      IDENTIFICATION DIVISION.

00003      PROGRAM-ID. HRVERIFY.

00004
00005      AUTHOR. LARRY NEWCOMER.
00006      INSTALLATION. PENN STATE UNIVERSITY, YORK CAMPUS.
00007
00008      DATE-WRITTEN. MAY 1983.
00009      DATE-COMPILED. MAY 9,1983.

00011      SECURITY. NONE.

00012
00013      *      HRVERIFY PRODUCE UN LISTADO IMPRESO DEL FICHERO SEMANAL
00014      *      TIEMPO NOMINAS (PREPARADO EN EL DISCO A PARTIR DE LAS
00015      *      TARJETAS DE TIEMPO SEMANAL). ESTE LISTADO SE USA PARA
00016      *      VERIFICAR VISUALMENTE QUE LOS DATOS INTRODUCIDOS:
00017      *      NOMBRE, ID Y HORAS SEMANALES, SON CORRECTOS.
00018      *      LA UNICA SALIDA IMPRESA ES EL INFORME PARA LA
00019      *      EDICION VISUAL

00020      ENVIRONMENT DIVISION.

00022      CONFIGURATION SECTION.

00024      SOURCE-COMPUTER. IBM-3081.
00025      OBJECT-COMPUTER. IBM-3081.

00027      INPUT-OUTPUT SECTION.

00028
00029      FILE-CONTROL.

00031      SELECT FICHERO-ENTRADA-TIEMPO-TARJETA
00032          ASSIGN TO TIEMTARJ
00033          ORGANIZATION IS SEQUENTIAL
00034          ACCESS IS SEQUENTIAL
00035
00036
00037      SELECT INFORME-VERIFICACION-TIEMPO

```

```

00038      ASSIGN TO TIEMLIST
00039      ORGANIZATION IS SEQUENTIAL
00040      ACCESS IS SEQUENTIAL
00041

00043      DATA DIVISION.

00045      FILE SECTION.

00047      FD  FICHERO-ENTRADA-TIEMPO-TARJETA
00048          BLOCK CONTAINS 0 RECORDS
00049          RECORD CONTAINS 80 CHARACTERS
00050          LABEL RECORDS ARE STANDARD

00051
00052
00053      01 REG-ENTRADA-TIEMPO-TARJETA    PIC X(80).

00055      FD  INFORME-VERIFICACION-TIEMPO
00056          RECORD CONTAINS 132 CHARACTERS
00057          LABEL RECORDS ARE OMITTED
00058          LINAGE IS 50
00059              WITH FOOTING AT 45
00060              LINES AT TOP 8
00061              LINES AT BOTTOM 8

00062
00063
00064      01 LINEA-VERIFICACION-TIEMPO    PIC X(132).

00066      WORKING-STORAGE SECTION.

00068      01 BANDERAS-E-INTERRUPTORES.
00069          05 WS-FIN-HOJA-TIEMPO        PIC X.

00071      01 AREAS-TRAB-ENTRADA-REG.

00073      05 WS-REG-TIEMPO.
00074          10 WS-ID-TIEMPO            PIC X(4).
00075          10 WS-ID-NOMBRE-TIEMPO    PIC X(20).
00076          10 WS-HORAS-TIEMPO       PIC S9(2)V9.
00077          10 FILLER                PIC X(53).

00079      01 AREAS-TRAB-SALIDA-REG.

00081      05 WS-CABECERA.
00082          10 FILLER                PIC X(4)      VALUE "-ID-".
00083          10 FILLER                PIC X(4)      VALUE SPACES.
00084          10 FILLER                PIC X(20)     VALUE "NOMBRE".
00085          10 FILLER                PIC X(4)      VALUE SPACES.
00086          10 FILLER                PIC X(5)      VALUE "HORAS".
00087          10 FILLER                PIC X(95)     VALUE SPACES.

00089      05 WS-LINEA.
00090          10 WS-ID                  PIC X(4)      VALUE SPACES.
00091          10 FILLER                PIC X(4)      VALUE SPACES.
00092          10 WS-NOMBRE               PIC X(20).
00093          10 FILLER                PIC X(4)      VALUE SPACES.
00094          10 WS-HORAS               PIC Z9.9-.
00095          10 FILLER                PIC X(95)     VALUE SPACES.

00097      PROCEDURE DIVISION.

00099      PERFORM INICIA-FICHEROS-INTERR
00100
00101      PERFORM PRODUCE-LINEA
00102          UNTIL WS-FIN-HOJA-TIEMPO
00103

```

```

00104      CLOSE   FICHERO-ENTRADA-TIEMPO-TARJETA
00105          INFORME-VERIFICACION-TIEMPO
00106
00107      STOP RUN
00108
00110      INICIA-FICHEROS-INTERRUPT.
00112          MOVE "F" TO WS-FIN-HOJA-TIEMPO
00113          OPEN    INPUT   FICHERO-ENTRADA-TIEMPO-TARJETA
00114              OUTPUT  INFORME-VERIFICACION-TIEMPO
00115          PERFORM LEE-FICHERO-TIEMPO
00116          WRITE LINEA-VERIFICACION-TIEMPO
00117              FROM WS-CABECERA
00118              AFTER ADVANCING PAGE
00119
00121      LEE-FICHERO-TIEMPO.
00123          READ FICHERO-ENTRADA-TIEMPO-TARJETA
00124              INTO WS-REG-TIEMPO
00125              AT END
00126                  MOVE "T" TO WS-FIN-HOJA-TIEMPO
00127
00129      PRODUCE-LINEA.
00131          MOVE WS-ID-TIEMPO      TO WS-ID
00132          MOVE WS-NOMBRE-TIEMPO  TO WS-NOMBRE
00133          MOVE WS-HORAS-TIEMPO  TO WS-HORAS
00134          WRITE LINEA-VERIFICACION-TIEMPO
00135              FROM WS-LINEA
00136              AFTER ADVANCING 3 LINES
00137              AT END-OF-PAGE
00138                  WRITE LINEA-VERIFICACION-TIEMPO
00139                      FROM WS-CABECERA
00140                      AFTER ADVANCING PAGE
00141
00142          PERFORM LEE-FICHERO-TIEMPO
00143

```

Una línea de salida del programador anterior podría ser:

0027      FOLKERS DEIRDRE      45.6

## 6.11 PROCESAMIENTO: INSTRUCCION ADD

La ejecución de la instrucción ADD hace que se sume el contenido de dos elementos y el resultado se sitúe en otro(s) elemento(s). Hay dos versiones de la instrucción, ADD... TO... y ADD... GIVING... (no se recomienda una tercera versión ADD CORRESPONDING... y no se verá en este libro).

### ADD... TO...

Esta opción tiene la estructura sintáctica que aparece en la Figura 6-4.

```

ADD { identificador-1 } [ identificador-2 ] ...
    | literal-1           | literal-2
    |                     |
TO identificador-m [ ROUNDED ] [ identificador-n [ ROUNDED ] ]
    | ON SIZE ERROR instrucción imperativa]

```

Fig. 6-4

**EJEMPLO 6.36**

## ● ADD HORAS-SEMANA TO TOTAL-HORAS

El contenido de HORAS-SEMANA se suma al contenido de TOTAL y la suma reemplazará al antiguo valor de TOTAL-HORAS; HORAS-SEMANA permanece como estaba. Ambos elementos deben ser numéricos (no editados). El compilador genera automáticamente el lenguaje máquina necesario para alinear los puntos decimales y convertir de un formato USAGE a otro.

## ● ADD ELEMENTO-A ELEMENTO-B TO ELEMENTO-C

Se suman los contenidos de ELEMENTO-A, ELEMENTO-B y ELEMENTO-C y el resultado reemplaza al anterior contenido de ELEMENTO-C; ELEMENTO-A y ELEMENTO-B permanecen como estaban.

## ● ADD ELEMENTO-A ELEMENTO-B ELEMENTO-C TO ELEMENTO-D ELEMENTO-E

La suma de ELEMENTO-A, ELEMENTO-B, ELEMENTO-C y ELEMENTO-D reemplaza al contenido original de ELEMENTO-D; la suma de ELEMENTO-A, ELEMENTO-B, ELEMENTO-C y ELEMENTO-E reemplaza el contenido anterior de ELEMENTO-E. ELEMENTO-A, ELEMENTO-B y ELEMENTO-C permanecen con el mismo contenido.

## ● ADD 1 TO NUMERO-DE-EMPLEADOS

Se suma 1 al contenido del campo NUMERO-DE-EMPLEADOS. El uso de literales numéricos en los cálculos es muy común.

**ADD... GIVING...**

La sintaxis de esta opción aparece en la Figura 6.5.

```
ADD { identificador-1 } { identificador-2 } [ identificador-3 ] ...
      literal-1   literal-2   literal-3
      GIVING identificador-m [ROUNDED] [identificador-n [ROUNDED]]...
      [ON SIZE ERROR instrucción-imperativa]
```

Fig. 6-5

**EJEMPLO 6.37**

## ● ADD HORAS-NORMALES HORAS-EXTRAS GIVING TOTAL-HORAS

La opción GIVING requiere al menos dos elementos a la izquierda de la palabra "GIVING" y al menos un elemento a su derecha. Se suman los contenidos de HORAS-NORMALES y HORAS-EXTRAS y el resultado reemplaza al anterior contenido de TOTAL-HORAS. Los campos HORAS-NORMALES y HORAS-EXTRAS conservan su antiguo valor. El anterior contenido de TOTAL-HORAS *no* forma parte de la suma. Los elementos a la izquierda de "GIVING" deben ser numéricos y no editados. Los elementos a la derecha de "GIVING" reciben el resultado de la suma, pero no toman parte en ella; por tanto, aunque también tienen que ser numéricos *pueden estar editados*.

● ADD IMPORTE-FACTURA-PAGADO  
FLETE-FACTURA-PAGADO  
ESPECIAL-FACTURA-PAGADO

GIVING IMPORTE-CHEQUE-SALIDA

El estilo de la escritura mejora la legibilidad y la facilidad de modificación. Todos los campos FACTURA-PAGADO deben ser numéricos y no estar editados. Se suman y el resultado reemplaza el contenido original de IMPORTE-CHEQUE-SALIDA (que no participa en la suma). El campo IMPORTE-CHEQUE-FACTURA está editado para su impresión.

● ADD IMPUESTOS-ESTADO  
IMPUESTOS-FEDERALES GIVING INFO-NOMINAS-IMP-TOTAL  
MAES-EMPLEADOS-IMP-TOTAL

Todos los campos deben ser numéricos; sin embargo, INFO-NOMINAS-IMP-TOTAL está editado (correcto, puesto que recibe el resultado). Los campos IMPUESTOS-ESTADO e IMPUESTOS-FEDERALES se suman y el resultado reemplaza el contenido original de INFO-NOMINAS-IMP-TOTAL y de MAES-EMPLEADOS-IMP-TOTAL.

**EJEMPLO 6.38 (Errores habituales con ADD)**● ADD IMPUESTOS-ESTADO TO IMPUESTOS-FEDERALES GIVING IMPUESTOS-TOTAL  
Error sintáctico; las opciones TO Y GIVING son excluyentes.

- ADD FLETE TO IMPORTE-FACTURA AND TOTAL-ENVIO

La palabra "AND" es inválida en las dos versiones de ADD...

- ADD HORAS-DIA GIVING HORAS-SEMANAS HORAS-MES

La opción GIVING requiere por lo menos dos elementos a la izquierda de la palabra "GIVING". Presumiblemente la intención era:

- ADD HORAS-DIA TO HORAS-SEMANA HORAS-MES

En las Secciones 6.12 y 6.13 se estudian las opciones ROUNDED y ON SIZE ERROR para las dos versiones de ADD.

## 6.12 PROCESAMIENTO: TRUNCAMIENTO Y OPCION ROUNDED

Todas las instrucciones aritméticas (ADD, SUBTRACT, MULTIPLY, DIVIDE y COMPUTE) generan automáticamente código máquina para (i) alinear los puntos decimales de los valores que se están sumando o restando; (ii) convertir de un formato USAGE a otro y volver a convertir al primitivo. Es posible que el resultado de un cálculo tenga más decimales que el campo al que se destina. En este caso, lo que se hace por defecto es eliminar (*truncar*) el número necesario de dígitos por la derecha.

Para modificar el proceso de truncamiento, se puede colocar, detrás del nombre de cualquier campo que vaya a recibir el resultado, la palabra "ROUNDED" (repase las Figs. 6-4 y 6-5). Con ROUNDED se indica:

- (1) si el dígito más a la izquierda de los que se truncan es 5 o superior, se añade 1 al dígito más a la derecha de los que no se truncan (*redondeo hacia arriba*);
- (2) si el dígito más a la izquierda de los que se truncan es 4 o inferior, el resultado se trunca sin alterar el dígito más a la derecha de los que no se truncan (*redondeo hacia abajo*).

### EJEMPLO 6.39

05 RESPUESTA PIC S99V9

.....

ADD X Y Z GIVING RESPUESTA

Si la suma de X, Y y Z es 54.34999, los cuatro últimos lugares se pierden y RESPUESTA contendrá 54.3.

.....

ADD X Y Z GIVING RESPUESTA ROUNDED

Con la opción ROUNDED también se obtiene como resultado 54.3 para RESPUESTA; al ser el dígito más a la izquierda de los que se van a truncar 4, el resultado *no* se redondea a 54.4.

### EJEMPLO 6.40

ADD HORAS-TRABAJADAS TO TOTAL-SEMANA  
TOTAL-MES ROUNDED  
TOTAL-AÑO ROUNDED

Se puede especificar ROUNDED sólo para algunos campos de recepción del cálculo.

## 6.13 PROCESAMIENTO: DESBORDAMIENTO Y LA CLAUSULA ON SIZE ERROR

Si la *parte entera* de un número calculado no cupiera en el campo de recepción, se produciría *desbordamiento* u ON SIZE ERROR. La cláusula ON SIZE ERROR permite que el programador especifique una serie de instrucciones que sólo se ejecutarán cuando ocurra dicho error.

**EJEMPLO 6.41**

```

ADD HORAS-SEMANA TO TOTAL-HORAS-AÑO
ON SIZE ERROR
MOVE ID-EMPLEADO TO ERROR-LINEA-ID
MOVE HORAS-SEMANA TO ERROR-LINEA-HORAS
MOVE "REBASAMIENTO HORAS ANUALES--TOTAL SIN HORAS"
    TO LINEA-MENSAJE-ERROR
WRITE LINEA-INFORME
    FROM ERROR-LINEA
    AFTER ADVANCING 3 LINES

```

PERFORM CALCULO-SALARIO-BRUTO

Si el resultado de sumar HORAS-SEMANA a TOTAL-HORAS-AÑO encajase perfectamente en TOTAL-HORAS-AÑO, las instrucciones de la cláusula ON SIZE ERROR... serían ignoradas. Observe que la cláusula ON SIZE ERROR (como AT END... y AT END-OF-PAGE) finaliza con un punto. La omisión del punto estructurado que precede a la instrucción PERFORM haría que PERFORM... pasará a formar parte de la cláusula ON SIZE ERROR, siendo ejecutado sólo cuando tuviéra lugar un error de desbordamiento y no en todos los casos como se pretende.

**EJEMPLO 6.42 Dado**

```

05 A PIC S99V99 VALUE +50.34.
05 B PIC S99V99 VALUE +50.23.
05 C PIC S99V9 VALUE +12.3.

```

- ADD A B GIVING C

Como la suma de A y B es +100.57, la parte entera del resultado es demasiado grande para caber en el campo de recepción C. Sin una cláusula ON SIZE ERROR, la computadora trunca el resultado (por la izquierda) y continúa la ejecución del programa. Por tanto, C contendrá +00.5, que es incorrecto obviamente.

- ADD A B GIVING C
   
ON SIZE ERROR
   
DISPLAY "ERROR DE DESBORDAMIENTO EN C--C NO SE HA MODIFICADO"

Con la cláusula ON SIZE ERROR, la suma truncada no se coloca en C; C continúa conteniendo +12.3. Más aún, DISPLAY imprime un mensaje de error (en el fichero especial del sistema SYSOUT si se trata del COBOL para el sistema IBM OS/VS).

**Directrices relativas a ON SIZE ERROR...**

1. Si es posible, fije la longitud suficiente en las cláusulas PICTURE para que, en los cálculos realizados por el programa, el desbordamiento sea matemáticamente imposible. Así se podrá omitir la cláusula ON SIZE ERROR sin peligro.
2. Si es ligeramente posible que ocurra un error de desbordamiento, utilice la cláusula ON SIZE ERROR.
3. Cuando se utilice la cláusula ON SIZE ERROR..., la rutina que la sigue debe (a) imprimir un mensaje de error con suficiente información como para subsanar el problema; (b) hacer que se salte total o parcialmente el procesamiento del registro que cause el error de desbordamiento; (c) permitir que el procesamiento continúe (no se debe parar la ejecución del programa).

**6.14 PROCESAMIENTO: INSTRUCCION SUBTRACT**

En la Figura 6-6 se dan los dos formatos principales de la instrucción SUBTRACT. Como SUBTRACT... FROM... es completamente análoga a ADD... TO..., sólo es necesario comentar SUBTRACT... FROM... GIVING...

SUBTRACT { identificador-1 } [ identificador-2 ] ...  
 literal-1 [ literal-2 ] ...  
FROM identificador-m [ ROUNDED ] [ identificador-n [ ROUNDED ] ] ...  
[ON SIZE ERROR instrucción-imperativa]  
 (a)  
  
SUBTRACT { identificador-1 } [ identificador-2 ] ...  
 literal-1 [ literal-2 ] ...  
FROM { identificador-m }  
 literal-m  
GIVING identificador-n [ ROUNDED ] [ identificador-o [ ROUNDED ] ] ...  
[ON SIZE ERROR instrucción-imperativa]  
 (b)

Fig. 6-6

**EJEMPLO 6.43**

- SUBTRACT DESCUENTO FROM PRECIO-CORRIENTE GIVING NUEVO-PRECIO

El contenido de DESCUENTO se resta del contenido de PRECIO-CORRIENTE y la diferencia reemplaza al contenido anterior de NUEVO-PRECIO. NUEVO-PRECIO puede ser un campo editado, puesto que no toma parte en el cálculo. Los campos DESCUENTO y PRECIO-CORRIENTE no se ven modificados.

- SUBTRACT 10 FROM NUMERO GIVING NUMERO-MENOS-DIEZ

Los literales numéricos, a menudo, toman parte en las instrucciones aritméticas.

- SUBTRACT IMPUESTOS-ESTADO  
IMPUESTOS-FEDERALES  
OTROS-IMPUESTOS FROM SALARIO-BRUTO  
GIVING SALARIO-NETO ROUNDED

SALARIO NETO es el único campo cuyo contenido se modifica. La suma de IMPUESTOS-ESTADO, IMPUESTOS-FEDERALES y OTROS-IMPUESTOS se resta de SALARIO-BRUTO, y el resultado, después de redondearse (ROUNDED), se coloca en SALARIO-NETO.

- SUBTRACT A B C  
FROM D  
GIVING E F ROUNDED G  
ON SIZE ERROR  
PERFORM RUTINA-DE-REBASAMIENTO.

La suma de A, B y C se resta de D; una copia del resultado se sitúa en E, una copia se redondea y se sitúa en F y otra copia se coloca en G. Si ocurriera un error de rebasamiento, se ejecutaría el párrafo RUTINA-DE-REBASAMIENTO antes de que la computadora pase a la siguiente instrucción.

- 01 EJEMPLO-DATOS.  
05 A PIC S9(3)V9.  
05 B PIC S9V99 COMP-3.  
05 C PIC ZZZ.99.

SUBTRACT B FROM C GIVING A

La instrucción SUBTRACT es inválida porque intenta utilizar en el cálculo un campo editado.

- SUBTRACT A B FROM C D GIVING E F

Error sintáctico: sólo un elemento puede seguir a "FROM" si se usa "GIVING".

**6.15 PROCESAMIENTO: INSTRUCCION MULTIPLY**

La instrucción MULTIPLY se usa para multiplicar dos elementos numéricos y tiene dos formatos.

**MULTIPLY... BY...**

La sintaxis de esta instrucción aparece en la Figura 6-7.

MULTIPLY { identificador-1 } BY identificador-2 [ROUNDED] [identificador-3 [ROUNDED]]...  
 literal-1  
 [ON SIZE ERROR instrucción-imperativa]

Fig. 6-7

**EJEMPLO 6.44**

- MULTIPLY PRECIO-UNITARIO BY CANTIDAD-COMPRADA  
GIVING IMPORTE-TOTAL

El contenido corriente de CANTIDAD-COMPRADA se multiplica por el contenido corriente de COSTE-UNITARIO y el producto se coloca en COSTE-UNITARIO. CANTIDAD-COMPRADA permanece invariable.

- MULTIPLY CANTIDAD-COMPRADA  
BY COSTE-UNITARIO ROUNDED  
ON SIZE ERROR  
PERFORM DEMASIADO-COSTE

El producto se redondea antes de colocarse en COSTE-UNITARIO. Si tiene lugar un SIZE ERROR, se ejecuta el párrafo DEMASIADO-COSTE antes de pasar a la siguiente instrucción.

- MULTIPLY .90 BY PRECIO-MAYORISTA  
PRECIO-DETALLE ROUNDED

El contenido de PRECIO-MAYORISTA se multiplica por .90 y el resultado sustituye al antiguo contenido de PRECIO-MAYORISTA (los decimales en exceso se truncan), después se multiplica por .90 el contenido de PRECIO-DETALLE (ROUNDED) sustituyendo el producto al viejo contenido de PRECIO-DETALLE.

- MULTIPLY PRECIO ACTUAL BY .90

Error *muy común*. Desgraciadamente la computadora tendría que multiplicar PRECIO-ACTUAL por .90 y colocar el resultado en el literal .90, lo cual resulta imposible. Lo correcto sería

MULTIPLY .90 BY PRECIO-ACTUAL

suponiendo que no importa perder el contenido antiguo.

**MULTIPLY... BY... GIVING...**

La sintaxis de esta opción aparece en la Figura 6-8.

MULTIPLY { identificador-1 } BY { identificador-2 }  
 literal-1  
 GIVING identificador-3 [ROUNDED] [identificador-4 [ROUNDED]]...  
 [ON SIZE ERROR instrucción-imperativa]

Fig. 6-8

**EJEMPLO 6.45**

- MULTIPLY PRECIO UNITARIO BY CANTIDAD-COMPRADA  
GIVING PRECIO-TOTAL

Los contenidos de PRECIO-UNITARIO y CANTIDAD-COMPRADA se multiplican y el producto sustituye a PRECIO-TOTAL. El resto de los campos permanecen invariables. Todos los datos tienen que ser numéricos; PRECIO-TOTAL podría estar editado, puesto que sólo recibe el resultado del cálculo sin tomar parte en él.

- MULTIPLY A BY B  
GIVING C  
D ROUNDED  
E  
ON SIZE ERROR  
PERFORM RUTINA REBASAMIENTO

El contenido de A se multiplica por el contenido de B; el producto sustituye al contenido original de C (los decimales de más se truncan), al contenido original de D (después de ser redondeado) y al contenido original de E (con truncamiento). Si tuviera lugar un error de rebasamiento, se ejecutaría la RUTINA-REBASAMIENTO antes de que la computadora pasase a la siguiente instrucción.

- MULTIPLY HORAS-TRABAJADAS SALARIO-HORA BY FACTOR-EXTRA  
GIVING SALARIO-HORAS-EXTRA

Error sintáctico: *no* se multiplican HORAS-TRABAJADAS, SALARIO-HORA y FACTOR-EXTRA; ni tampoco funcionaría

MULTIPLY HORAS-TRABAJADAS SALARIO-HORA BY FACTOR-EXTRA ..

A diferencia de ADD... y SUBTRACT..., MULTIPLY sólo puede multiplicar dos valores cada vez. Una solución al problema sería:

MULTIPLY HORAS-TRABAJADAS BY SALARIO-HORA GIVING TEMPORAL  
MULTIPLY TEMPORAL BY FACTOR-EXTRA  
GIVING IMPORTE-TOTAL

donde TEMPORAL es un área definida en el almacenamiento de trabajo por el programador. Para una solución mejor, véase la Sección 6.17.

## 6.16 PROCESAMIENTO: INSTRUCCION DIVIDE

La instrucción DIVIDE se utiliza para dividir el contenido de dos elementos numéricos. Tiene tres formatos, uno de los cuales produce *dos* resultados, el cociente y el resto. Ningún formato permite dividir por cero. Cualquier intento de dividir por cero causará que el programa aborte a menos que se haya incluido la cláusula ON SIZE ERROR, en cuyo caso se produciría un error de rebasamiento.

### DIVIDE... INTO...

La sintaxis se detalla en la Figura 6-9.

DIVIDE {identificador-1}  
  |  
  literal-1  
INTO identificador-2 [ROUNDED] [identificador-3 [ROUNDED]]...  
  [ON SIZE ERROR instrucción-imperativa]

Fig. 6-9

### EJEMPLO 6.46

- DIVIDE 100 INTO PORCENTAJE-PERDIDA

El literal numérico 100 divide al contenido de PORCENTAJE-PERDIDA cuyo contenido se ve sustituido por el resultado de la división. El campo PORCENTAJE-PERDIDA debe ser numérico no editado.

- DIVIDE A INTO B

C ROUNDED  
D  $\frac{A}{B}$ ,  
ON SIZE ERROR  
PERFORM RUTINA-MENSAJE-ERROR

B se divide entre A, el resultado (posiblemente truncado por la derecha) reemplaza al contenido anterior de B; C se divide entre A y el resultado (después de redondearse) reemplaza al contenido anterior de C; D se divide entre A y el resultado (truncado) reemplaza al contenido anterior de D. Si ocurriera un error de rebasamiento, se ejecutaría la instrucción PERFORM; en otro caso se saltaría.

- DIVIDE A B C INTO D

Error sintáctico: sólo se permite un identificador a cualquier lado de "INTO". (MULTIPLY... y DIVIDE... *no* tienen la misma sintaxis que ADD... y SUBTRACT...)

**DIVIDE... INTO/BY... GIVING...**

La sintaxis aparece en la Figura 6-10.

```
DIVIDE {identificador-1} {INTO} {identificador-2}
    {literal-1} {BY} {literal-2}
    GIVING identificador-3 [ROUNDED] [identificador-4 [ROUNDED]]...
    [ON SIZE ERROR instrucción-imperativa]
```

Fig. 6-10

**EJEMPLO 6.47**

- DIVIDE 100 INTO PORCENTAJE-PERDIDA GIVING FRACCION-PERDIDA

El valor del literal 100 divide al contenido de PORCENTAJE-PERDIDA y el resultado reemplaza al contenido original de FRACCION-PERDIDA. Todos los elementos deben ser numéricos; el campo FRACCION-PERDIDA puede estar editado, puesto que no toma parte en los cálculos sino que se limita a recibir el resultado.

- DIVIDE PORCENTAJE-PERDIDA BY 100 GIVING FRACCION-PERDIDA

El efecto es idéntico al de la instrucción precedente.

- DIVIDE NUMERO-EMPLEADOS INTO SALARIO-TOTAL  
GIVING SALARIO-MEDIO *Resultado*  
ON SIZE ERROR  
PERFORM IMP-MENSAJE-ERROR

100 ÷ 8 = 12 R 4

El contenido de SALARIO-TOTAL se divide entre NUMERO-EMPLEADOS y el resultado reemplaza al contenido anterior de SALARIO-MEDIO (que puede estar editado). NUMERO-EMPLEADOS y SALARIO-TOTAL permanecen inalterados. Si ocurriera un error de rebasamiento, se ejecutaría IMP-MENSAJE-ERROR.

- DIVIDE B BY A GIVING C  
D ROUNDED  
E  
ON SIZE ERROR  
PERFORM RUTINA-ERROR

El contenido de B se divide entre A; el resultado reemplaza al contenido original de C (con truncamiento de la parte fraccionaria si fuera necesario), al contenido de D (con redondeo) y al contenido de E (posiblemente con truncamiento). Si tuviera lugar un error de rebasamiento, se ejecutaría el párrafo RUTINA-ERROR. A y B permanecen inalterados; C, D y E pueden estar editados si se desea.

- DIVIDE A INTO B ROUNDED GIVING C

Inválido: "ROUNDED" sólo se puede utilizar con los campos que reciben el cociente.

**DIVIDE... INTO/BY... GIVING... REMAINDER...**

Esta versión tiene la sintaxis que se muestra en la Figura 6-11.

```
DIVIDE {identificador-1} {INTO} {identificador-2}
    {literal-1} {BY} {literal-2}
    GIVING identificador-3 [ROUNDED]
    REMAINDER identificador-4
    [ON SIZE ERROR instrucción-imperativa]
```

Fig. 6-11

**EJEMPLO 6.48**

- DIVIDE A INTO B GIVING C REMAINDER D

Todos los elementos deben ser numéricos; C y D pueden estar editados. El contenido de B se divide entre A, el *cociente* reemplaza al contenido original de C; el *resto* reemplaza al contenido original de D. Por "cociente"

se entiende el resultado de la división con tantos decimales como se especifique en la cláusula PICTURE de C; el "resto" también queda definido por

$$\text{resto} = \text{dividendo} - (\text{divisor} \times \text{cociente})$$

Por ejemplo, si A contiene 3.3, B contiene 10.2 y C puede acomodar dos dígitos decimales, entonces

$$\text{cociente} = 3.09 \quad \text{resto} = 10.2 - (3.09)(3.3) = .003$$

Si se especifica "ROUNDED" para el elemento de "GIVING" (que recibe el cociente), el resto se calcula *antes* de efectuar el redondeo. Si se incluye la cláusula ON SIZE ERROR... y tiene lugar un error de rebasamiento para el cociente (elemento GIVING), entonces *ambos* elementos (GIVING y REMAINDER) *no se cambian* y se ejecuta la rutina de ON SIZE ERROR. Si ocurriera un error de rebasamiento *sólo* para el resto, el elemento de GIVING se reemplaza por el cociente, pero el campo REMAINDER *no se modifica*. El programador puede conocer si el cociente o el resto causaron el error de rebasamiento, para ello, debería comprobar si se ha *modificado* el campo cociente (en cuyo caso el resto causó el error de rebasamiento) o *no se ha modificado* (en cuyo caso el cociente causó el error de rebasamiento). Recuerde: si no se incluye la cláusula ON SIZE ERROR, la computadora pasa a la siguiente instrucción, aunque se hayan almacenado resultados incorrectos en el elemento GIVING y/o en el REMAINDER.

- DIVIDE B BY A GIVING C REMAINDER D  
Esto es precisamente equivalente a la instrucción anterior.

- DIVIDE 24 INTO TOTAL-HORAS GIVING NUMERO-DE-DIAS  
REMAINDER RESTO-Horas

El cociente es el número de días completos contenidos en TOTAL-HORAS; el resto representa las horas sobrantes.

- DIVIDE SALDO BY IMPORTE-CADA-PAGO  
GIVING NUM-PAGOS-NORMALES  
REMAINDER IMPORTE-ULTIMO-PAGO

En las aplicaciones de créditos comerciales, el último pago de un préstamo es, por lo general, distinto de los demás. La instrucción DIVIDE calcula el *número* de pagos normales y el *importe* (menor) del último pago.

- DIVIDE A INTO B GIVING C ROUNDED REMAINDER D ROUNDED  
Error sintáctico: El adjetivo "ROUNDED" sólo puede usarse para el elemento de "GIVING", *no* con el campo REMAINDER.

- DIVIDE IMPORTE PAGO INTO SALDO  
REMAINDER ULTIMO-PAGO

Este es un error común: la opción GIVING debe usarse cuando se especifica REMAINDER. Si no se tiene interés en el cociente, se puede utilizar la siguiente técnica:

```
DIVIDE IMPORTE-PAGO INTO SALDO
  GIVING CAMPO-MUDO
  REMAINDER ULTIMO-PAGO
```

donde CAMPO-MUDO se define en el almacenamiento de trabajo (pero nunca se usa).

## 6.17 PROCESAMIENTO: INSTRUCCION COMPUTE

La instrucción COMPUTE permite muchas operaciones aritméticas de una vez. Genera una traducción al programa objeto más eficiente que si los mismos cálculos se escribieran mediante varias instrucciones más de un tipo de operación aritmética. En estos casos se debe utilizar la instrucción COMPUTE...

La instrucción COMPUTE tiene sólo un formato (Fig. 6-12).

```
COMPUTE identificador-1 [ROUNDED]
  [identificador-2 [ROUNDED]]...
   =expresión-aritmética
  [ON SIZE ERROR instrucciones-imperativas]
```

Fig. 6-12

La "expresión-aritmética" en la instrucción COMPUTE es similar a las expresiones del álgebra elemental. Se compone de elementos numéricos y/o constantes numéricas combinadas por medio de uno o más de los cinco *operadores aritméticos* de COBOL.

+	<i>Suma</i>	/	<i>División</i>
-	<i>Resta</i>	**	<i>Potenciación</i> (elevación a un exponente)
*	<i>Multiplicación</i>		

Estos son *operadores binarios*, es decir, que requieren *dos operandos* para el cálculo. COBOL también permite el uso del *signo menos* para cambiar el signo de un elemento. Los elementos situados a la derecha “=”, como participan en el cálculo, no pueden estar editados; los elementos al lado izquierdo, como reciben el resultado, pueden estar editados. En la instrucción COMPUTE, cada operador aritmético debe estar precedido y seguido por un espacio en blanco o bien precedido por “)” y seguido por “(”.

#### EJEMPLO 6.49

- COMPUTE A = B + C

es el equivalente de

ADD B C GIVING A

- COMPUTE A = B - C

es el equivalente de

SUBTRACT C FROM B GIVING A

- COMPUTE A = B \* C

es el equivalente de

MULTIPLY BY C GIVING A

- COMPUTE A = B / C

es el equivalente de

DIVIDE B BY C GIVING A

- COMPUTE A = B \*\* C

no tiene equivalente de otro tipo. “B \*\* C” representa “B elevado a C”, es decir,  $B^C$ . Por ejemplo, B \*\* 3 es B al cubo.

- COMPUTE A = - B

(signo menos) es el equivalente de

SUBTRACT B FROM ZERO GIVING A

Se puede utilizar el paréntesis para controlar el orden de ejecución de las operaciones dentro de la instrucción COMPUTE: las operaciones en el paréntesis más interior se realizan primero, después las operaciones del siguiente paréntesis, etc. En ausencia de paréntesis, el compilador sigue las siguientes reglas:

- (1) todos los signos menos unitarios se realizan en primer lugar de izquierda a derecha;
- (2) a continuación \*\* (potenciación) de izquierda a derecha;
- (3) a continuación \* y / de izquierda a derecha;
- (4) por último + y - de izquierda a derecha.

#### EJEMPLO 6.50

- COMPUTE A ROUNDED = (B + C) \* (D + E)

Las operaciones se realizan en este orden: (i) se calcula la suma de B y C, (ii) se calcula la suma de D y E, (iii) los resultados de (i) e (ii) se multiplican, (iv) el producto se redondea y (v) el producto redondeado

reemplaza al contenido original de A. Los elementos B, C, D y E deben ser numéricos no editados; A podría estar editado.

- COMPUTE A ROUNDED  
 B  
 C                   = A + B \* C / D  
 ON SIZE ERROR  
 PERFORM IMPRIME-MENSAJE-ERROR

En ausencia de paréntesis, ocurrirá lo siguiente: (i) se calcula B \* C, (ii) se divide el resultado de (i) por D, (iii) el resultado de (ii) se suma a A, (iv) el resultado de (iii) reemplaza al contenido de A (con redondeo), (v) el resultado de (iii) reemplaza al contenido de B (con truncamiento), (vi) el resultado de (iii) sustituye al contenido de C (con truncamiento). Si tuviera lugar un error de rebasamiento al almacenar el resultado en A, B o C, se ejecutaría IMPRIME-MENSAJE-ERROR.

- COMPUTE SALARIO-BRUTO = (40 \* SALARIO-HORA) +  
 (1.5 \* SALARIO-HORA \* (HORAS-TRABAJADAS - 40))

Esta es una fórmula típica cuando existan horas extras. Para las primeras 40 horas se paga el salario normal ("40 \* SALARIO-HORA"). Las horas que excedan de las cuarenta ("HORAS-TRABAJADAS - 40") se pagan una vez y media el precio de la hora normal ("1.5 \* SALARIO-HORA"). El salario bruto es la suma de las dos componentes.

- COMPUTE NUMERO-DE-EMPLEADOS = NUMERO-DE-EMPLEADOS + 1  
 Esta instrucción tan rara es equivalente a

ADD 1 TO NUMERO-DE-EMPLEADOS

- COMPUTE EXISTENCIAS = EXISTENCIAS (100)  
 + CANTIDAD-RECIBIDA (10)  
 + CANTIDAD-DEVUELTA (20)  
 - CANTIDAD-ENVIADA (30)  
 - CANTIDAD-PERDIDA (40)

Al contenido de EXISTENCIAS se le suman CANTIDAD-RECIBIDA y CANTIDAD-DEVUELTA, después al resultado se le restan CANTIDAD-ENVIADA y CANTIDAD-PERDIDA. El resultado final vuelve a ponerse en EXISTENCIAS, destruyendo el contenido anterior. Si cada elemento tuviera *al principio* el contenido detallado entre paréntesis, el contenido *final* de EXISTENCIAS sería 60. El resto de los campos no cambiarían.

- COMPUTE SALARIO-MEDIO =  
 (SALARIO-TOTAL + OTROS-CONCEPTOS)/NUMERO-DE-EMPLEADOS

El uso de paréntesis es esencial en este caso. Sin ellos, OTROS-CONCEPTOS se dividiría por NUMERO-DE-EMPLEADOS y el resultado se sumaría a SALARIO-TOTAL, lo que produciría un distorsión del SALARIO-MEDIO.

- COMPUTE PRECIO-EN-LIQUIDACION ROUNDED =  
 PRECIO-NORMAL - .15 \* PRECIO-NORMAL

No se necesitan paréntesis en este caso, puesto que de todos modos la multiplicación precede a la resta. Una versión más simple de este cálculo sería:

COMPUTE PRECIO-EN-LIQUIDACION ROUNDED = .85 \* PRECIO-NORMAL

- COMPUTE C = (A \*\* 2 + B \*\* 2) \*\* .5

Elevar a 0.5 es lo mismo que extraer la raíz cuadrada; por tanto, se calcula "la raíz cuadrada de la cantidad A al cuadrado más la cantidad B al cuadrado". En notación algebraica,

$$c = \sqrt{a^2 + b^2}$$

**EJEMPLO 6.51** Supóngase que un préstamo se paga mensualmente. Se desea calcular el importe pagado en concepto de principal e intereses y el nuevo saldo.

COMPUTE INTERESES = SALDO-ANTERIOR \* (TASA-INTERESES / 12)  
 COMPUTE PRINCIPAL-PAGADO = IMPORTE-PAGADO - INTERESES  
 COMPUTE SALDO-NUEVO = SALDO-ANTERIOR - PRINCIPAL-PAGADO

Observe que la tasa de interés se divide por 12 para obtener la tasa de interés mensual.

**EJEMPLO 6.52 (Usos incorrectos de COMPUTE)**

- (a) Un error común entre principiantes consiste en omitir un espacio, por lo menos, antes y después de cada operador aritmético que no esté rodeado por ")" y "(", o bien antes y después de cada "=":

```
COMPUTE A = B+C
COMPUTE A = B+C*D
COMPUTE A = (A+B)/(A-B)
```

son todos incorrectos, aunque "COMPUTE A = (A + B)/(A - B)" es válido.

- (b) La falta de los espacios en blanco podría hacer que un signo menos se considerase como un guión:

```
COMPUTE A = B-C
```

El compilador buscará un elemento denominado "B-C"; no considerará que es una resta.

- (c) Un error frecuente consiste en omitir paréntesis cuando son necesarios. *No hay desventajas en hacer uso de los paréntesis* —por tanto, si existen dudas sobre su necesidad, utilícelos para estar seguro.

```
COMPUTE SALDO-NUEVO = SALDO-ANTERIOR
                     - IMPORTE-PAGADO
                     + SALDO-ANTERIOR*
                     TASA-INTERES / 12
```

es correcto. Sin embargo, lo siguiente también es correcto y no es menos eficiente:

```
COMPUTE SALDO-NUEVO =
      SALDO-ANTERIOR
      - (IMPORTE-PAGADO
         - (SALDO ANTERIOR * (TASA-INTERES / 12)))
```

Observe cómo los paréntesis se disponen en pares; la omisión de un elemento de la pareja [ "(" o ")" ] es un error común.

### **6.18 EFICIENCIA EN LOS CALCULOS ARITMETICOS**

En general se obtiene un programa objeto más eficiente cuando:

1. Los elementos que participan en el cálculo tienen PICTURE con los puntos decimales alineados.
2. Los elementos que participan en el cálculo tienen el mismo formato USAGE.
3. Los elementos tienen formato USAGE, COMP o COMP-3.
4. (En el COBOL del sistema IBM OS/VS.) Los campos en formato COMP-3 tienen un número impar de dígitos (especialmente cuando se utiliza la cláusula ON SIZE ERROR).
5. Si un campo que participa en bastantes cálculos está en formato DISPLAY, el campo se lleva a un área del almacenamiento de trabajo que sea COMP o COMP-3 y se utiliza dicha copia en los cálculos (así se produce una sola conversión de formato USAGE en vez de una por cálculo).
6. Se utiliza "S" en las cláusulas PICTURE de los elementos numéricos (a menos que se quiera reflejar un valor absoluto).
7. Se evita el uso de la cláusula ON SIZE ERROR... cuidando el diseño de las cláusulas PICTURE.

### **6.19 NORMAS DE CODIFICACION ADICIONALES PARA LA DIVISION DE PROCEDIMIENTOS**

11. Utilice sólo *una* instrucción STOP RUN por programa.
12. Rompa las instrucciones largas al principio de una cláusula y desplace la línea siguiente para hacer visible que pertenece a la misma instrucción.
13. Coloque las cláusulas AT END..., AT END-OF-PAGE... y ON SIZE ERROR... en linea independientes pero desplazadas.

14. Alinee verticalmente las instrucciones y sus partes.
15. Utilice literales o constantes sólo si está seguro que su valor no cambiará durante la vida del programa.

#### EJEMPLO 6.53

- MOVE ZERO TO NUMERO-DE-EMPLEADOS

Como el campo NUMERO-DE-EMPLEADOS siempre se pone a cero al principio de la ejecución del programa, el uso del literal (constante figurativa) ZERO está justificado.

- MULTIPLY PRECIO-NORMAL BY .08 GIVING DESCUENTO-EMPLEADO

La tasa de descuento para los empleados podría modificarse; el literal no debe, por tanto, utilizarse. En su lugar, el programador debe definir un elemento en el almacenamiento de trabajo:

WORKING-STORAGE SECTION.

```
01 CONSTANTES-PROGRAMA.  
    05 TASA-DESCUENTO-EMPLEADOS    PIC SV99  
                                VALUE +.08.
```

y debería sustituir la instrucción MULTIPLY anterior por

```
MULTIPLY PRECIO-NORMAL BY TASA-DESCUENTO-EMPLEADOS GIVING  
DESCUENTO-EMPLEADO
```

Ahora, si se cambia la tasa de descuento a los empleados, sólo tiene que cambiarse una cláusula VALUE y no el literal a lo largo de toda la división de procedimientos.

### Preguntas de repaso

- 6.1 ¿Cuál es el propósito de la división de procedimientos?
- 6.2 ¿Explique la estructura de la división de procedimientos definiendo: sección, párrafo, frase, instrucción y verbo.
- 6.3 Comente las normas de codificación para la estructura de la división de procedimientos.
- 6.4 ¿Cuál es la función de la instrucción OPEN?
- 6.5 Explique el significado de los modos de OPEN: INPUT, OUTPUT, I-O y EXTEND.
- 6.6 Comente las diferencias entre el uso de una instrucción OPEN para abrir todos los ficheros y el uso de una instrucción OPEN para cada fichero.
- 6.7 ¿Qué pasa si el programa se refiere a un registro lógico antes de que el fichero correspondiente se abra o después que se haya cerrado?
- 6.8 ¿Cuál es la función de la instrucción CLOSE?
- 6.9 Comente las diferencias entre el uso de una instrucción CLOSE para cerrar todos los ficheros y el uso de una instrucción CLOSE para cada fichero.
- 6.10 Comente cómo se podría abrir y cerrar un fichero varias veces por el mismo programa.
- 6.11 ¿Cuál es la función de la cláusula "WITH LOCK" en una instrucción CLOSE?
- 6.12 ¿Qué es una cinta multifichero?

- 6.13 Explique el uso de STANDARD LABELS para una cinta multifichero.
- 6.14 ¿Qué es un fichero multivolumen?
- 6.15 ¿Qué se entiende por “cambio de volumen”?
- 6.16 ¿Cuál es el propósito de la instrucción READ?
- 6.17 Explique la función de la cláusula AT END en una instrucción READ.
- 6.18 ¿Qué es la condición de fin de fichero? ¿Cuándo tiene lugar?
- 6.19 ¿Por qué es fundamental no procesar registros lógicos después de que haya tenido lugar la condición de fin de fichero?
- 6.20 ¿Por qué algunos programadores realizan el procesamiento de los registros lógicos utilizando copias de los mismos en el almacenamiento de trabajo?
- 6.21 Explique el uso de READ... INTO...
- 6.22 ¿Qué se indica con “lee un fichero, escribe un registro”?
- 6.23 ¿Cuál es la función de la instrucción WRITE?
- 6.24 ¿Qué hay en el área del registro lógico para un fichero inmediatamente después de una instrucción WRITE correcta?
- 6.25 ¿Cuáles son algunas de las ventajas de la construcción de los registros lógicos en el almacenamiento de trabajo y después mediante WRITE... FROM... trasladarlos al buffer para darlos salida?
- 6.26 Si no se pueden utilizar READ... INTO... y WRITE... FROM... con registros de longitud variable, ¿cómo se pueden procesar estos registros en el almacenamiento de trabajo?
- 6.27 Comente los diversos modos de controlar el espaciado en la impresora.
- 6.28 ¿Qué sucede si una instrucción WRITE para la impresora no contiene cláusulas de espaciado?
- 6.29 Dé algunos ejemplos de cuándo debe utilizarse un elemento para especificar el número de líneas en la opción BEFORE/AFTER ADVANCING de la instrucción WRITE.
- 6.30 ¿Cuáles son los significados posibles de BEFORE/AFTER ADVANCING PAGE?
- 6.31 Explique cómo pueden utilizarse los canales de control del carro especificados en el párrafo SPECIAL-NAMES para controlar el espaciado en la instrucción WRITE.
- 6.32 Explique las áreas de una página lógica definidas en la cláusula LINAGE.
- 6.33 Diga bajo qué *dos* condiciones se ejecutará la rutina de AT END-OF-PAGE dentro de la instrucción WRITE.
- 6.34 ¿Qué puede suceder si la cláusula LINAGE, los valores de BEFORE/AFTER ADVANCING y la rutina de AT END-OF-PAGE no están adecuadamente coordinados?
- 6.35 ¿Qué es el desbordamiento automático de página?
- 6.36 Defina “instrucción-condicional” e “instrucción-imperativa”.
- 6.37 ¿Cuándo la instrucción READ es condicional?, ¿y cuándo es imperativa?
- 6.38 ¿Cuándo la instrucción WRITE es condicional?, ¿y cuándo es imperativa?

- 6.39 ¿Qué tipo de información puede obtenerse desde el sistema operativo por medio de la instrucción ACCEPT? ¿Cómo puede utilizarse dicha información?
- 6.40 Dé algunos ejemplos de cuándo podría utilizarse la instrucción ACCEPT para dar entrada a información desde la consola del operador de la computadora.
- 6.41 ¿Cuándo *no* debe usarse ACCEPT... en lugar de un fichero regular con instrucciones SELECT, FD, OPEN, READ y CLOSE?
- 6.42 ¿Qué son las tarjetas de control?
- 6.43 ¿Son frecuentes las instrucciones DISPLAY para depurar el programa?
- 6.44 ¿Cuándo podría usarse DISPLAY... para visualizar mensajes de error (en lugar de un fichero regular de salida definido con SELECT, FD, OPEN, WRITE, CLOSE)?
- 6.45 Dé algunos ejemplos de cuándo se podría utilizar DISPLAY... para comunicarse con el operador de la computadora a través de la consola del terminal.
- 6.46 Además de "trasladar" datos, ¿qué otras funciones puede llevar a cabo la instrucción MOVE?
- 6.47 Defina: (a) campo de envío, (b) campo de recepción, (c) conversión, (d) relleno, (e) truncamiento, (f) edición.
- 6.48 ¿Qué reglas de truncamiento y relleno se aplican cuando se manipulan (a) datos numéricos, (b) datos no numéricos?
- 6.49 ¿Qué sucede en COBOL cuando se manipula un elemento compuesto mediante la instrucción MOVE?  
¿Qué errores comunes se producen?
- 6.50 ¿Cuándo se puede ejecutar MOVE para un elemento compuesto?
- 6.51 Diga cuáles son las combinaciones de datos ilegales en una instrucción MOVE. Dé algunos ejemplos de instrucciones MOVE inválidas.
- 6.52 ¿Cuál es la función de STOP RUN?
- 6.53 ¿Por qué deben cerrarse todos los ficheros antes de ejecutar STOP RUN?
- 6.54 ¿Qué haría la instrucción 'STOP "DAME UN BESO"'?
- 6.55 ¿Cuál es el propósito de la instrucción PERFORM?
- 6.56 Distinga entre las "órdenes físicas" de la división de procedimientos y las "órdenes lógicas".
- 6.57 ¿Qué es un bucle?
- 6.58 Explique con detalle cómo funciona la instrucción PERFORM.
- 6.59 Explique con detalle cómo funciona la instrucción PERFORM... UNTIL...
- 6.60 ¿Qué sucede si la condición que marca el fin de PERFORM... UNTIL... se satisface en medio de la ejecución del párrafo?
- 6.61 ¿Qué es un programa de edición?
- 6.62 ¿Qué es una copia impresa?
- 6.63 ¿Cuál es la función de la instrucción ADD?
- 6.64 Indique las diferencias entre los efectos del truncamiento y la opción ROUNDED cuando se obtienen resultados fraccionarios al hacer operaciones aritméticas.

- 6.65 ¿Qué se entiende por “desbordamiento”?
- 6.66 Explique el uso de la cláusula ON SIZE ERROR en las operaciones aritméticas.
- 6.67 ¿Qué puede hacerse para que no sea necesaria la inclusión de la cláusula ON SIZE ERROR?
- 6.68 Indique las reglas para la escritura de la rutina de ON SIZE ERROR.
- 6.69 ¿Cuándo se pueden utilizar datos numéricos editados en las instrucciones aritméticas?
- 6.70 Explique el propósito de la instrucción MULTIPLY. ¿En qué difiere su sintaxis de la de las instrucciones ADD y SUBTRACT?
- 6.71 ¿Cuál es la función de la instrucción DIVIDE?
- 6.72 Defina: (a) cociente, (b) resto.
- 6.73 ¿Qué sucede en COBOL si se intenta dividir por cero?
- 6.74 ¿Cuáles son algunas de las ventajas de utilizar la instrucción COMPUTE?
- 6.75 Diga cuáles son los cinco operadores binarios utilizados en la instrucción COMPUTE.
- 6.76 ¿Qué se entiende por signo menos unitario?
- 6.77 ¿Cuál es el papel de los paréntesis en la instrucción COMPUTE?
- 6.78 ¿Cuál es el orden de operaciones por defecto si no hay paréntesis?
- 6.79 Indique algunos errores frecuentes en el uso de COMPUTE.
- 6.80 Comente qué puede hacerse en COBOL para hacer que las operaciones aritméticas sean lo más eficiente posible.
- 6.81 ¿Cuándo es apropiado el uso de literales en un programa?
- 6.82 ¿Qué técnica se puede utilizar en lugar del uso de literales en la división de procedimientos? ¿Cuándo debe utilizarse esta técnica? ¿Por qué con ella es más sencillo modificar después los programas?

### Problemas resueltos

- 6.83 Explique la frase: “La mayor parte de los algoritmos en los negocios consisten en la repetición, hasta el final del fichero, del ciclo entrada-proceso-salida”.

Un *algoritmo* es un conjunto detallado de pasos que hay que llevar a cabo para solucionar un problema. Los problemas típicos en el mundo de los negocios consisten en el procesamiento de todos los registros lógicos de un fichero de entrada. Estos problemas se construyen alrededor de un *bucle principal* en el cual (1) se da entrada al siguiente registro lógico; (2) se procesa dicho registro lógico y (3) si procede se da salida a dicho registro lógico. Los pasos 1, 2 y 3 se repiten hasta que se llega al fin del fichero en la ejecución del paso primero.

- 6.84 Un programa debe leer un fichero de entrada que contiene registros históricos de ventas y añadirlos a un fichero ya existente. Muestre cómo deben abrirse ambos ficheros.

OPEN	INPUT	TRANS-VENTAS-HISTORICAS
	EXTEND	MAESTRO-VENTAS-HISTORICAS

La opción EXTEND permite añadir los nuevos registros al final del fichero ya existente y organizado secuencialmente.

- 6.85 En el Problema 6.84, supóngase que se desea borrar todo el contenido del fichero y reemplazarlo por registros totalmente nuevos. Indique las instrucciones OPEN.

```
OPEN   INPUT    TRANS-VENTAS-HISTORICAS
      OUTPUT   MAESTRO-VENTAS-HISTORICAS
```

Cuando un fichero ya existente se abre en modo OUTPUT, se borran todos los registros lógicos que pudiera contener y se crea una versión totalmente nueva escribiendo nuevos registros lógicos.

- 6.86 ¿Qué está equivocado en la siguiente división de procedimientos?

```
FD  FICHERO-ENTRADA...
01  REGISTRO-ENTRADA...
```

```
FD  FICHERO-INFORME...
01  LINEA-INFORME...
```

PROCEDURE-DIVISION.

```
MOVE SPACES TO REGISTRO-ENTRADA
WRITE LINEA-INFORME FROM WS-LINEA-CABECERA
      AFTER ADVANCING PAGE
OPEN   INPUT    FICHERO-ENTRADA
      OUTPUT   FICHERO-INFORME
```

Se dan dos errores comunes:

- (1) MOVE SPACES TO REGISTRO-ENTRADA accede al área del registro lógico *antes de que el fichero haya sido abierto*. Esto siempre ocasiona errores, puesto que no se han creado todavía los buffers.
- (2) WRITE LINEA-INFORME FROM... accede al área del registro lógico antes de que el fichero haya sido abierto (puesto que WRITE... FROM... es equivalente a la instrucción MOVE seguida de WRITE) e (incluso peor) intenta dar salida a un registro antes de que se haya abierto el fichero. Este último error haría abortar al programa.

- 6.87 OPEN, READ, WRITE y CLOSE sitúan una “clave de estado” en los 2 bytes del área del FILE STATUS para el fichero (en el supuesto de que dicha área se hubiera definido para el fichero). ¿Qué valor de la clave de estado indica la ejecución con éxito de cualquier instrucción de entrada/salida?

En el COBOL del sistema IBM OS/VS, “00” siempre indica la ejecución con éxito de una instrucción de entrada/salida. Para el código de estado puede utilizarse PIC XX o PIC 99.

- 6.88 ¿Qué está equivocado en:

```
05  AREA-CODIGO-ESTADO  PIC XX.
```

```
.....
```

```
OPEN INPUT FICHERO-EJEMPLO
IF AREA-CODIGO-ESTADO EQUAL 00 ...
```

El área de clave de estado definida con PIC XX es no numérica; en este caso debe utilizarse un literal no numérico.

```
IF AREA-CODIGO-ESTADO EQUAL "00"...
```

- 6.89 ¿Qué puede hacer que falle una instrucción CLOSE y que por tanto se coloque un código no nulo en el área del FILE STATUS?

Si el fichero tuviera STANDARD LABELS, un fallo del equipo podría ocasionar errores incorregibles durante el procesamiento de etiquetas. Otra posibilidad es que el fichero podría no haberse abierto con éxito, en cuyo caso no podría cerrarse.

- 6.90 Muestre cómo se pueden cerrar tres ficheros minimizando (a) tiempo de ejecución, (b) bytes de la memoria ocupados.

(a)     CLOSE FICHERO-TRANS-INVENTARIO  
           FICHERO-MAESTRO-INVENTARIO  
           FICHERO-INVENTARIO-ACTUAL

Cuando se cierran muchos ficheros con una instrucción CLOSE, se minimiza el tiempo de ejecución (aunque se ocupa más memoria de la computadora).

(b)     CLOSE FICHERO-TRANS-INVENTARIO  
           CLOSE FICHERO-MAESTRO-INVENTARIO  
           CLOSE FICHERO-INVENTARIO-ACTUAL

Cerrar cada fichero individualmente ocupa menos memoria, pero toma más tiempo.

- 6.91 Suponga un programa que procese dos ficheros en la misma cinta (cinta multifichero). Abre el primer fichero, lo procesa y después lo cierra y abre el segundo. Escriba las instrucciones OPEN y CLOSE.

```
OPEN INPUT PRIMER-FICHERO-EN-CINTA
.....
CLOSE PRIMER-FICHERO-EN-CINTA WITH NO REWIND
OPEN INPUT SEGUNDO-FICHERO-EN-CINTA
```

- 6.92 Cuando un programa procesa secuencialmente un fichero multivolumen, el sistema operativo, por lo general, manipula el *cambio automático de volumen*. Explique cómo funciona esto para (a) un fichero de entrada, (b) un fichero de salida.

- (a) El programa empieza leyendo registros lógicos con el primer volumen de un fichero volumen. Si hubiera 500 registros lógicos en el primer volumen, la ejecución de la instrucción 501 causaría un cambio automático de volumen en el que el sistema operativo (i) se asegura de que el segundo volumen del fichero está montado, (ii) procesa las etiquetas estándar del segundo volumen y (iii) pone a disposición del programa el primer registro lógico del segundo volumen.
- (b) Cuando se abra un fichero multivolumen en modo OUTPUT, el fichero se posiciona al principio del primer volumen. El programa ejecuta una instrucción WRITE pero no hay más espacio en el primer volumen; el sistema operativo automáticamente (i) finaliza el procesamiento de etiquetas para el primer volumen, (ii) comienza el procesamiento de la etiqueta del segundo volumen y (iii) escribe el registro lógico al principio del segundo volumen.

- 6.93 Relacione la instrucción CLOSE... UNIT con el cambio automático de volumen.

Se ha visto (Problema 6.92) cómo el sistema operativo inicia el cambio automático de volumen cuando se encuentra un *fin de volumen* durante el procesamiento normal. La instrucción CLOSE... UNIT se puede utilizar para *forzar* el cambio de volumen en un punto arbitrario del programa. Esto puede ser útil cuando se está creando un fichero multivolumen, porque permite que el programa controle directamente cuantos registros lógicos se escriban en cada volumen.

- 6.94 ¿Qué está equivocado en lo siguiente?

```
OPEN INPUT FICHERO-EJEMPLO
READ FICHERO-EJEMPLO
AT END
MOVE "T" TO INTER-FIN-FICHERO
MOVE REGISTRO-EJEMPLO TO AREA-SALIDA
WRITE REGISTRO-SALIDA
FROM AREA-SALIDA
```

Hay varios errores. (1) No hay puntos que marquen el final de la cláusula AT END. (2) Se ejecuta WRITE REGISTRO-SALIDA... sin que esté abierto el fichero. (3) Al programa le hace falta comprobar la condición fin de fichero antes de realizar "MOVE REGISTRO-EJEMPLO TO AREA-SALIDA". Si tiene lugar el fin de fichero, no hay REGISTRO-EJEMPLO que procesar (y el programa procesa un *registro lleno de basura*).

- 6.95 En la siguiente división de procedimientos se intenta copiar un fichero de tarjetas de 80 bytes en disco. Corrija los errores.

PROCEDURE DIVISION.

```
OPEN INPUT FICHERO-TARJETAS
MOVE "NO" TO FIN-DE-FICHERO
PERFORM COPIA-UN-REGISTRO
    UNTIL FIN-DE-FICHERO EQUAL "SI"
CLOSE FICHERO-DISCO
```

COPIA-UN-REGISTRO.

```
READ FICHERO-TARJETAS
    AT END MOVE "SI" TO FIN-DE-FICHERO
MOVE REGISTRO-TARJETA TO REGISTRO-DISCO
WRITE REGISTRO-DISCO
```

Los errores, por orden de ocurrencia, son: (1) No se ha abierto el FICHERO-DISCO. (2) No se ha cerrado FICHERO-TARJETAS. (3) No existe la instrucción STOP RUN. (4) Se echa en falta el punto después de AT END... (5) No se comprueba la condición fin de fichero antes de ejecutar MOVE y WRITE con un fichero de tarjetas. La versión correcta es:

PROCEDURE DIVISION.

```
OPEN INPUT FICHERO-TARJETAS
    OUTPUT FICHERO-DISCO
MOVE "NO" TO FIN-DE-FICHERO
PERFORM COPIA-UN-REGISTRO
    UNTIL FIN-DE-FICHERO EQUAL "SI"
CLOSE FICHERO-TARJETAS
    FICHERO-DISCO
STOP RUN
```

COPIA-UN-REGISTRO

```
READ FICHERO-TARJETAS
    AT END MOVE "SI" TO FIN-DE-FICHERO

IF FIN-DE-FICHERO EQUAL "NO"
    MOVE REGISTRO-TARJETA TO REGISTRO-DISCO
    WRITE REGISTRO-DISCO
```

- 6.96 Suponiendo que se hayan abierto todos los ficheros, ¿qué está equivocado en lo siguiente?

```
WRITE REGISTRO-EJEMPLO-SALIDA
    FROM WS-AREA-REGISTRO-SALIDA
```

\* COMENTARIO -- MUESTRA REGISTRO RECIEN ESCRITO:  
POR DEBUGGING

DISPLAY REGISTRO-EJEMPLO-SALIDA

Después de que se haya ejecutado la instrucción WRITE, la variable REGISTRO-EJEMPLO-SALIDA no vuelve a referirse al registro lógico que acaba de escribirse. En su lugar, se refiere al espacio de un *nuevo* registro lógico en el buffer del fichero, que puede utilizarse para construir un nuevo registro lógico. En este caso la instrucción DISPLAY da salida a basura, *no* al registro que acaba de escribirse.

**6.97** Corrija el error del Problema 6.96.

Como se utilizó la instrucción WRITE... FROM..., la corrección consiste únicamente en sustituir  
DISPLAY WS-AREA-REGISTRO-SALIDA

(Después de la ejecución de WRITE, WS-AREA-REGISTRO-SALIDA todavía contiene el registro lógico que acaba de escribirse.)

Si el Problema 6.96 tuviera una instrucción WRITE..., por ejemplo,

```
WRITE REGISTRO-EJEMPLO-SALIDA
DISPLAY REGISTRO-EJEMPLO-SALIDA
```

la corrección en este caso sería:

```
DISPLAY REGISTRO-EJEMPLO-SALIDA
WRITE REGISTRO-EJEMPLO-SALIDA
```

**6.98** ¿Qué error hay en lo siguiente?

```
FD FICHERO-ENTRADA
RECORD CONTAINS 50 TO 200 CHARACTERS
.....
OPEN INPUT FICHERO-ENTRADA
READ FICHERO-ENTRADA
    INTO WS-AREA-REGISTRO-ENTRADA
    AT END...
```

No se puede utilizar READ... INTO... con registros de longitud variable (véase el Apéndice C para las consideraciones del COBOL de 1980).

**6.99** Muestre cómo se puede corregir el error del Problema 6.98.

```
FD FICHERO-ENTRADA
RECORD CONTAINS 50 TO 200 CHARACTERS
.....
OPEN INPUT FICHERO-ENTRADA
READ FICHERO-ENTRADA
    AT END...
MOVE REGISTRO-ENTRADA TO WS-AREA-REGISTRO-ENTRADA
```

Para registros de longitud variable, utilice READ... seguido de MOVE... en lugar de READ... INTO... De forma similar, utilice MOVE... seguido de WRITE... en lugar de WRITE... FROM...

**6.100** ¿Qué error hay en lo siguiente?

```
READ FICHERO-EJEMPLO-ENTRADA
    AT END
    MOVE "SI" TO FIN-DE-FICHERO
    WRITE LINEA-INFORME
        FROM WS-LINEA-TOTAL
        AT FIN-DE-FICHERO
        ADD 1 TO NUMERO-DE-PAGINAS
```

Todas las instrucciones de la rutina AT END... deben ser imperativas. Aquí, WRITE... AT END-OF-PAGE... es condicional. Esto ocasiona un error sintáctico.

**6.101** Corrija el error del Problema 6.100.

Si fuera necesario ejecutar una instrucción condicional cuando las reglas obligan a que sea imperativa, se puede incluir la instrucción condicional en un párrafo y después utilizar PERFORM (que es imperativa) para ejecutar el párrafo.

```
READ FICHERO-EJEMPLO-ENTRADA
  AT END
    PERFORM RUTINA-AT-END

.....
RUTINA-AT-END.
```

```
MOVE "SI" TO FIN-DE-FICHERO
WRITE LINEA-INFORME
  FROM WS-LINEA-TOTAL
  AT END OF PAGE
    ADD 1 TO NUMERO-DE-PAGINAS
```

- 6.102** Se desea producir un listado a partir de un paquete de tarjetas de 80 caracteres, con una copia exacta de cada tarjeta en papel de 80 columnas. Cada página lógica debe reproducir exactamente 10 tarjetas y contener una cabecera con título, número de página, las fechas normal y juliana y la hora de la ejecución. También debe contener al pie de cada página la nota "HASTA EL MOMENTO SE HAN IMPRESO \_\_\_\_ TARJETAS". Comente cómo lleva a cabo esta tarea el programa de la Figura 6-13.

```
00001 IDENTIFICATION DIVISION.
00002   PROGRAM-ID. CARDLIST.
00003   AUTHOR. LARRY NEWCOMER.
00004   INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS.
00005   DATE-WRITTEN. MAY 1983.
00006   DATE-COMPILED. MAY 9,1983.
00007   SECURITY. NONE.
00008   * CARDLIST PRODUCE UN LISTADO 80/80 DE TARJETAS.
00009   * SU PROPOSITO ES ILUSTRAR VARIAS FORMAS DE LA
00010   * INSTRUCCION WRITE ... BEFORE/AFTER ADVANCING ...
00011   * CON LA CLAUSULA LINAGE.
00012 ENVIRONMENT DIVISION.
00013   CONFIGURATION SECTION.
00014     SOURCE-COMPUTER. IBM-370.
00015     OBJECT-COMPUTER. IBM-370.
00016   INPUT-OUTPUT SECTION.
00017     FILE-CONTROL.
00018       SELECT FICHERO-TARJETAS           ASSIGN TO TARJET.
00019       SELECT FICHERO-IMPRESION        ASSIGN TO IMPRES.
00020 DATA DIVISION.
00021   FILE SECTION.
00022     FD FICHERO-TARJETAS
00023       RECORD CONTAINS 80 CHARACTERS
00024       LABEL RECORDS ARE OMITTED
00025
00026     01 ENTRADA-TARJETA          PIC X(80).
00027   *
00028     FD FICHERO-IMPRESION
00029       RECORD CONTAINS 132 CHARACTERS
00030       LABEL RECORDS ARE OMITTED
00031       LINAGE IS 13
00032         WITH FOOTING AT 11
```

Fig. 6-13

```

00034      LINES AT TOP 2
00035      LINES AT BOTTOM 2
00036
00037
00038      01 LINEA-IMPRESION          PIC X(132).
00039      * -----
00040
00041      WORKING-STORAGE SECTION.
00042
00043      *
00044      01 WS-AREAS-FECHA-HORA.
00045          05 WS-FECHA-NORMAL.
00046              10 WS-AA           PIC 99.
00047              10 WS-MM           PIC 99.
00048              10 WS-DD           PIC 99.
00049      *
00050
00051      01 INTERRUPTORES-Y-CONTADORES.
00052          05 INTER-FIN-TARJETAS    PIC X(3).
00053          05 WS-NUM-PAGINA       PIC S9(3)   COMP-3.
00054          05 WS-CONT-TARJ          PIC S9(3)   COMP-3.
00055      01 WS-AREA-LINEA.
00056          05 WS-AREA-TARJETA     PIC X(80).
00057          05 FILLER            PIC X(52)   VALUE SPACES.
00058
00059      *
00060      01 WS-AREA-CABECERA.
00061          05 FILLER            PIC X(6)    VALUE "FECHA".
00062          05 CABECERA-MM        PIC Z9.
00063          05 FILLER            PIC X       VALUE "/".
00064          05 CABECERA-DD        PIC 99.
00065          05 FILLER            PIC X       VALUE "/".
00066          05 CABECERA-AA        PIC 99.
00067          05 FILLER            PIC X(9)   VALUE ", JULIANA ".
00068          05 CABECERA-JULIANA    PIC Z9B999.
00069          05 FILLER            PIC X(7)   VALUE ", HORA".
00070          05 CABECERA-HORA       PIC 99B99B99B99.
00071          05 FILLER            PIC X(6)   VALUE " PAGINA ".
00072          05 WS-CAB-NUM-PAGINA   PIC ZZ9.
00073          05 FILLER            PIC X(76)  VALUE SPACES.
00074      *
00075
00076      01 WS-AREA-PIE.
00077          05 FILLER            PIC X(7)   VALUE "HASTA EL MOMENTO ".
00078          05 WS-CONT-PIE        PIC ZZ9.
00079          05 FILLER            PIC X(122) VALUE " SE HAN IMPRESO".
00080
00081
00082      PROCEDURE DIVISION.
00083
00084          MOVE "NO " TO INTER-FIN-TARJETAS
00085          MOVE ZERO TO WS-NUM-PAGINA
00086              WS-CONT-TARJ
00087
00088          ACCEPT WS-FECHA-NORMAL      FROM FECHA
00089          ACCEPT CABECERA-JULIANA    FROM DIA
00090          ACCEPT CABECERA-HORA       FROM HORA
00091
00092          MOVE WS-MM             TO CABECERA-MM
00093          MOVE WS-DD             TO CABECERA-DD
00094          MOVE WS-AA             TO CABECERA-AA
00095
00096          OPEN   INPUT           FICHERO-TARJETAS
00097              OUTPUT          FICHERO-IMPRESION
00098
00099          PERFORM ENTRADA-REGISTRO
00100

```

Fig. 6-13 (cont.)

```

00101      PERFORM PROD-CABECERA-PAGINA
00102
00103      ——PERFORM PROD-LISTADO-80-80
00104          UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00105
00106          CLOSE    FICHERO-TARJETAS
00107              FICHERO-IMPRESION
00108          STOP RUN
00109

00111      ENTRADA-REGISTRO.
00112
00113          READ FICHERO-TARJETAS
00114              INTO WS-AREA-TARJETA
00115              AT END
00116                  MOVE "SI" TO INTER-FIN-TARJETAS
00117

00119      PROD-LISTADO-80-80.
00120          ADD 1 TO WS-CONT-TARJ.
00121              IND1.
00122          WRITE LINEA-IMPRESION
00123              FROM WS-AREA-LINEA
00124              AFTER ADVANCING 1 LINE
00125              AT END-OF-PAGE
00126                  PERFORM PROD-PIE-PAGINA
00127                  PERFORM PROD-CABECERA-PAGINA
00128
00129
00130          PERFORM ENTRADA-REGISTRO
00131
00132

00134      PROD-CABECERA-PAGINA.
00135
00136          ADD 1 WS-NUM-PAGINA
00137          MOVE WS-NUM-PAGINA TO WS-CAB-NUM-PAGINA
00138          WRITE PRINT-LINE
00139              FROM WS-AREA-CABECERA
00140              AFTER ADVANCING PAGE
00141
00142
00143      PROD-PIE-PAGINA.
00144
00145          MOVE WS-CONT-TARJ TO WS-CONT-PIE
00146
00147          WRITE LINEA-IMPRESION
00148              FROM WS-AREA-PIE
00149              AFTER ADVANCING 2 LINES
00150

```

Fig. 6-13 (cont.)

- (1) El programa empieza leyendo la primera tarjeta (después de abrir los ficheros) y debe empezar escribiendo la cabecera en la parte superior de la primera página. Creando un párrafo con la rutina de la cabecera de página se podría ejecutar al principio de la división de procedimientos y después cuando fuera necesario en la rutina AT END-OF-PAGE de PROD-LISTADO-80-80. Como la impresión empieza en la parte superior de la primera página lógica, la condición END-OF-PAGE no se detecta en la primera instrucción WRITE. La línea 101 de la Figura 6-13 se asegura de que la primera página tendrá una cabecera.
- (2) El sistema comprueba si se ha producido rebasamiento de pie antes de comprobar el rebasamiento de página. Cuando se imprime la décima tarjeta (en la línea 11 del papel, por lo que ocupa la línea de cabecera), se reconoce el rebasamiento del pie y se ejecuta la rutina de AT END-OF-PAGE causando la impresión del pie de página y después de la cabecera al principio de la siguiente página lógica. Así las cosas, cuando se va a imprimir la tarjeta undécima, el papel ya está

posicionado en la parte superior de una página y el rebasamiento de página no tendrá lugar. Se ve que utilizando un área de pie de página no se producen rebasamientos de página.

- (3) El valor de LINAGE (línea 32 de la Fig. 6-13) se determina como sigue. Como la cabecera requiere 1 línea y se desean 10 tarjetas por página, el valor de LINAGE debe ser al menos 11 (sin el pie de página). Si el pie va a imprimirse AFTER ADVANCING 2 LINES, estas dos líneas adicionales configuran 13 líneas en el cuerpo de página. La Figura 6-14 muestra un ejemplo de página impresa.

```

FECHA 5/09/83, JULIANA 83 129, HORA 13 08 12 41 PAGINA 2
CARD 11
CARD 12
CARD 13
CARD 14
CARD 15
CARD 16
CARD 17
CARD 18
CARD 19
CARD 20

HASTA EL MOMENTO 20 SE HAN IMPRESO

```

Fig. 6-14

- (4) A propósito de las líneas 88-90 de la Figura 6-13, los campos CABECERA-JULIANA y CABECERA-HORA están editados. La fecha normal entra en WS-FECHA-NORMAL porque el sistema tiene la fecha en formato aammdd (que es la forma más conveniente para clasificar por fechas). Los subcampos mes, día y año se mueven por separado al área de cabecera (véanse las líneas 92-94).

- 6.103** El programa de la Figura 6-15 es el mismo que el de la Figura 6-13 excepto que está en modo de depuración ("WITH DEBUGGING MODE") como se especifica en el párrafo SOURCE-COMPUTER y se han añadido instrucciones DISPLAY en la división de procedimientos para ayudar en la depuración del programa. ¿Qué se muestra mediante estas instrucciones DISPLAY?

```

00084      PROCEDURE DIVISION.

00086      MOVE "NO " TO INTER-FIN-TARJETAS
00087      MOVE ZERO TO WS-NUM-PAGINA
00088          WS-CONT-TARJ
00089
00090          ACCEPT WS-FECHA-NORMAL           FROM FECHA
00091          ACCEPT CABECERA-JULIANA        FROM DIA
00092          ACCEPT CABECERA-HORA          FROM HORA
00093
00094      D    DISPLAY "VALOR INICIAL DE INTER-FIN-TARJETAS="
00095      D    INTER-FIN-TARJETAS
00096      D    DISPLAY "VALORES INICIALES DE WS-NUM-PAGINA Y WS-CONT-TARJ="
00097      D    WS-NUM-PAGINA WS-CONT-TARJ
00098      D    DISPLAY "VALOR DE WS-FECHA-NORMAL=" WS-FECHA-NORMAL
00099      D    DISPLAY "VALOR DE CABECERA HORA=" CABECERA-HORA
00100      D    DISPLAY "VALOR DE FECHA JULIANA=" CABECERA-JULIANA
00101
00102
00103      MOVE WS-MM                      TO CABECERA-MM
00104      MOVE WS-DD                      TO CABECERA-DD
00105      MOVE WS-AA                      TO CABECERA-AA
00106
00107      OPEN   INPUT                   FICHERO-TARJETAS
00108      OUTPUT                         FICHERO-IMPRESION
00109

```

Fig. 6-15

```
00110      PERFORM ENTRADA-REGISTRO
00111
00112      PERFORM PROD-CABECERA-PAGINA
00113
00114      PERFORM PROD-LISTADO-80-80
00115          UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00116
00117      CLOSE    FICHERO-TARJETAS
00118          FICHERO-IMPRESION
00119
00120      STOP RUN

00122      ENTRADA-REGISTRO.
00123
00124      READ FICHERO-TARJETAS RECORD
00125          INTO WS-AREA-TARJETA
00126          AT END
00127              MOVE "SI" TO INTER-FIN-TARJETAS
00128
00129      D      DISPLAY "TARJETA RECIEN LEIDA"
00130      D      DISPLAY "INTER-FIN-TARJETAS ES:"
00131      D
00132

00134      PROD-LISTADO-80-80.
00135
00136      ADD 1 TO WS-CONT-TARJ
00137
00138      WRITE LINEA-IMPRESION
00139          FROM WS-AREA-LINEA
00140          AFTER ADVANCING 1 LINE
00141          AT END-OF-PAGE
00142              PERFORM PROD-PIE-PAGINA
00143              PERFORM PROD-CABECERA-PAGINA
00144
00145
00146      PERFORM ENTRAD-REGISTRO
00147

00149      PROD-CABECERA-PAGINA.
00150
00151      D      DISPLAY "** AL ENTRAR EN PROD-CABECERA-PAGINA, CONT. PAGINA="
00152          WS-CONT-TARJ
00153
00154      ADD 1 TO WS-NUM-PAGINA
00155      MOVE WS-NUM-PAGINA TO WS-CAB-NUM-PAGINA
00156      WRITE LINEA-IMPRESION
00157          FROM WS-AREA-CABECERA
00158          AFTER ADVANCING PAGE
00159
00160
00161      PROD-PIE-PAGINA.
00162
00163      D      DISPLAY "** AL ENTRAR EN PROD-CABECERA-PAGINA, CONT. PAGINA="
00164          WS-CONT-TARJ
00165
00166      MOVE WS-CONT-TARJ TO WS-CONT-PIE
00167
00168      WRITE LINEA-IMPRESION
00169          FROM WS-AREA-PIE
00170          AFTER ADVANCING 2 LINES
00171
```

Fig. 6-15 (cont.)

Observe la Figura 6-16. Como puede ver, el compilador COBOL del sistema IBM OS/VS elimina el signo de los *campos positivos* en formato DISPLAY; así sucede con WS-CONT-MRJ. Observe también que los números en formato COMP-3 se convierten a DISPLAY para su impresión con la instrucción DISPLAY.

Estudie este uso de DISPLAY y la salida que produce: puede ahorrarle mucho tiempo en la depuración de programas. En la Figura 6-16 se refleja también el orden de ejecución de las instrucciones del programa.

```

VALOR INICIAL DE INTER-FIN-TARJETAS=NO
VALORES INICIALES DE WS-NUM-PAGINA Y WS-CONT-TARJ=000000
VALOR DE WS-FECHA-NORMAL=830509
VALOR DE CABECERA-HORA=13 14 00 72
VALOR DE FECHA JULIANA=83 129
TARJETA RECIEN LEIDA: CARD 1
INTER-FIN-TARJETAS ES: NO
** AL ENTRAR EN PROD-CABECERA-PAGINA, CONT PAGINAS=000
TARJETA RECIEN LEIDA: CARD 2
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 3
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 4
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 5
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 6
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 7
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 8
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 9
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 10
INTER-FIN-TARJETAS ES: NO
** AL ENTRAR EN PROD-PIE-PAGINA, CONT. PAGINAS=010
** AL ENTRAR EN PROD-CABECERA-PAGINA, CONT. PAGINAS=010
TARJETA RECIEN LEIDA: CARD 11
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 12
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 13
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 14
INTER-FIN-TARJETAS ES: NO
TARJETA RECIEN LEIDA: CARD 15

```

Fig. 6-16

- 6.104 Muestre cómo se puede trasladar A que es PIC SV999 a B que es PIC SV9 con redondeo.

COMPUTE B ROUNDED=A

Aunque es menos elegante,

ADD ZERO A GIVING B ROUNDED

también funcionaría.

- 6.105 Muestre cuál es el contenido del campo de recepción después de cada una de las siguientes instrucciones MOVE:

	<i>Valor campo de envío</i>	<i>PICTURE del campo de recepción</i>
(a)	123.456	S9(5)V9(4)
(b)	123.456	S9(5)V9
(c)	"STRING"	X(4) JUSTIFIED RIGHT
(d)	"STRING"	X(4)
(e)	"DOG"	X(5)
(f)	"DOG"	X(5) JUSTIFIED RIGHT
(g)	"DOG"	9(3)
(h)	HIGH-VALUES	9(3)
(i)	HIGH-VALUES	X(3)
(j)	SPACES	9(3)
(k)	ALL "\$"	X(5)
(l)	123.456	S99V99
(m)	-123.456	999V999
(n)	-123.456	\$\$\$\$\$.99-
(o)	-123456	\$\$\$\$\$.99BCR

(a) 00123.45600; (b) 00123.4; (c) "RING"; (d) "STRI"; (e) "DOGbb"; (f) "bbDOG"; (g) inválido (mezcla datos numéricos y alfanuméricos); (h) inválido (la constante figurativa HIGH-VALUES es alfanumérica y sólo puede mover hacia campos del tipo PIC X); (i) 3 bytes del carácter mayor en la secuencia de orden; (j) inválido (la constante figurativa SPACES es alfanumérica y sólo se puede mover hacia campos del tipo PIC X o PIC A); (k) "\$\$\$\$\$"; (l) 23.45 (el truncamiento se produce por la izquierda y por la derecha); (m) 123.456 (valor absoluto, puesto que el campo de recepción no tiene signo); (n) \$123.45-; (o) \$23456.00bCR (el dígito más a la izquierda se trunca).

#### 6.106 Dado

```

01 A.
  05 B    PIC X(4)  VALUE "THIS".
  05 C    PIC X(2)  VALUE "IS".
  05 D    PIC X(3)  VALUE "FUN".

05 W.
  05 X    PIC X(3).
  05 Y    PIC X(2).
  05 Z    PIC X(3).

```

indique el contenido de (a) X, (b) Y y (c) Z, después de

MOVE A TO W

COBOL trata las instrucciones MOVE aplicadas a elementos compuestos como si se aplicaran a largos elementos del tipo PIC X. En este caso, A tiene 9 bytes de longitud y W tiene 8; por tanto, el byte más a la derecha de A (que es el byte más a la derecha de D) se truncará:

- (a) "THI"      (b) "SI"      (c) "SFU"

#### 6.107 Critique lo siguiente:

```

01 A.
  05 B    PIC S(3)V99  COMP-3.
  05 C    PIC S99      COMP.
  05 D    PIC S99V9    DISPLAY.

```

MOVE ZEROS TO A

Como A es un elemento compuesto, se trata como si fuera PIC X DISPLAY y por tanto se rellenan los campos de A con ceros en formato DISPLAY. Esto está bien para D (por que es DISPLAY), pero no funciona con B y C.

El procedimiento correcto es:

MOVE ZEROS TO B C D

que mueve ceros a cada campo individual efectuando automáticamente el cambio de formato USAGE.

- 6.108** Critique lo siguiente:

.....  
STOP RUN  
DISPLAY "TODOS LOS FICHEROS ESTAN CERRADOS"

Las instrucciones son sintácticamente correctas, pero DISPLAY no se ejecutará nunca. Después de ejecutar STOP RUN la computadora deja de ejecutar las instrucciones del programa.

- 6.109** Suponga que una compañía paga como máximo 55 dólares por hora. Suponga también que el máximo número de horas por semana es 60 (a partir de 40, extras que se pagan a una vez y media las normales). Si la compañía tiene 1.000 empleados, escriba la cláusula PICTURE para TOTAL-NOMINA-SEMANA de forma que no pueda ocurrir ningún error de rebasamiento.

Si un empleado trabaja el máximo número de horas, su salario semanal será

$$(40 \times 55.00) + (20 \times 55.00 \times 1.5) = 3850.00 \text{ dólares}$$

Entonces, TOTAL-NOMINA-SEMANA no puede superar

$$1000 \times 3850.00 = 3,850,000.00 \text{ dólares}$$

por tanto, PIC S9(7)V99 será adecuada.

- 6.110** Exprese las siguientes fórmulas algebraicas utilizando COMPUTE...:

$$(a) \quad a = \frac{b}{c} + b - (c \times d) \quad (b) \quad a = \frac{b+c}{d} - \frac{c}{e+f} \quad (c) \quad d = \frac{(a+b)^2}{a-b} + a - (b+c)$$

- (a) COMPUTE A = B / C + B - C \* D  
 (b) COMPUTE A = (B + C) / D - C / (E + F)  
 (c) COMPUTE D = (A + B) \*\* 2 / (A - B) + A - (B + C)

- 6.111** Un cálculo típico del salario bruto cuando hay horas extras es:

COMPUTE SALARIO-BRUTO = (HORAS-TRABAJADAS - 40) \*  
 SALARIO-HORA \* 1.5 + 40 \* SALARIO-HORA

Indique cómo podría realizarse este cálculo sin hacer uso de la instrucción COMPUTE (de nombres descriptivos para las áreas que necesite).

```
MULTIPLY 40 BY SALARIO-HORA GIVING SALARIO-NORMAL
MULTIPLY SALARIO-HORA BY 1.5 GIVING SALARIO-HORA-EXTRA
SUBTRACT 40 FROM HORAS-TRABAJADAS GIVING HORAS-EXTRAS
MULTIPLY SALARIO-HORA-EXTRA BY HORAS-EXTRAS
GIVING SALARIO-EXTRA
ADD SALARIO-NORMAL SALARIO-EXTRA GIVING SALARIO-BRUTO
```

La instrucción COMPUTE es más fácil de escribir y produce un programa objeto más eficiente.

En ambos casos, hubiera sido mejor utilizar constantes definidas en el almacenamiento de trabajo en lugar de literales numéricos como 1.5 y 40 que podrían tomar nuevos valores durante la vida del programa.

### Problemas complementarios

- 6.112** Escriba un programa que imprima etiquetas de correo (dos a lo largo de cada página). Después debe indicar una línea con el total de etiquetas impresas. La entrada es igual que en el Problema 2.36.

*Respuesta.* Véase la Figura 6-17. El programa procesa los registros de entrada en el área del registro lógico para el fichero; las líneas de impresión son construidas en el almacenamiento de trabajo, como siempre. En la Figura 6-18 se muestra un ejemplo de salida del programa.

```

00001 IDENTIFICATION DIVISION.
00002
00003 PROGRAM-ID. MAILING.
00004 AUTHOR. LARRY NEWCOMER.
00005 INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS.
00006 DATE-WRITTEN. MAY 1983.
00007 DATE-COMPILED. MAY 9,1983.
00008 SECURITY. NONE.
00009 * MAILING PRODUCE DOS COLUMNAS DE ETIQUETAS DE
00010 * CORREO. TAMBIEN IMPRIME EL NUMERO DE ETIQUETAS
00011 * IMPRESAS. ANTES DE ESCRIBIR LAS ETIQUETAS, EL
00012 * PROGRAMA ESCRIBE UNA MUESTRA QUE PUEDE USARSE
00013 * PARA ALINEAR EL PAPEL DE LA IMPRESORA.

00015 ENVIRONMENT DIVISION.
00016
00017 CONFIGURATION SECTION.
00018 SOURCE-COMPUTER. IBM-370.
00019 OBJECT-COMPUTER. IBM-370.
00020 INPUT-OUTPUT SECTION.
00021 FILE-CONTROL.
00022   SELECT MAESTRO-CORREO      ASSIGN TO CORREO.
00023   SELECT FICHERO-ETIQUETAS-CORREO  ASSIGN TO ETIQUETA.

00025 DATA DIVISION.
00026
00027 FILE SECTION.
00028
00029 FD MAESTRO-CORREO
00030   RECORD CONTAINS 80 CHARACTERS
00031   LABEL RECORDS ARE OMITTED
00032
00033 01 ENTRADA-CORREO.
00034    05 NOMBRE-CORREO          PIC X(20).
00035    05 DOMICILIO-CORREO      PIC X(20).
00036    05 CIUDAD-CORREO         PIC X(20).
00037    05 PAIS-CORREO           PIC X(2).
00038    05 CODIGO-POSTAL-CORREO PIC X(5).
00039    05 FILLER                PIC X(13).

00041 FD FICHERO-ETIQUETAS-CORREO
00042   RECORD CONTAINS 132 CHARACTERS
00043   LABEL RECORDS ARE OMITTED
00044
00045 01 LINEA-ETIQUETA          PIC X(132).

00047 WORKING-STORAGE-SECTION.
00048
00049 01 INTERRUMPTORES-Y-CONTADORES.
00050    05 INTER-FIN-FICHERO      PIC X(3).
00051    05 WS-CONT-ETIQ          PIC S9(3)      COMP-3.
00052
00053 01 WS-LINEA-NOMBRE.
00054    05 NOMBRE-1              PIC X(30)      VALUE ALL "*".
00055    05 FILLER                PIC X(10)      VALUE SPACES.

```

Fig. 6-17

```

00056      05 NOMBRE-2          PIC X(30)    VALUE ALL "★".
00057      05 FILLER           PIC X(62)    VALUE SPACES.
00058
00059      01 WS-LINEA-DOMICILIO.
00060          05 DOMICILIO-1    PIC X(30)    VALUE ALL "★".
00061          05 FILLER         PIC X(10)    VALUE SPACES.
00062          05 DOMICILIO-2    PIC X(30)    VALUE ALL "★".
00063          05 FILLER         PIC X(62)    VALUE SPACES.
00064
00065      01 WS-LINEA-RESTO.
00066          05 CIUDAD-1       PIC X(20)    VALUE ALL "★".
00067          05 FILLER         PIC X        VALUE SPACES.
00068          05 PAIS-1         PIC XX      VALUE ALL "★".
00069          05 FILLER         PIC X(2)    VALUE SPACES.
00070          05 CODIGO-POSTAL-1 PIC X(5)     VALUE ALL "★".
00071          05 FILLER         PIC X(10)    VALUE SPACES.
00072          05 CIUDAD-2       PIC X(20)    VALUE ALL "★".
00073          05 FILLER         PIC X        VALUE SPACES.
00074          05 PAIS-2         PIC XX      VALUE ALL "★".
00075          05 FILLER         PIC X(2)    VALUE SPACES.
00076          05 CODIGO-POSTAL-2 PIC X(5)     VALUE ALL "★".
00077          05 FILLER         PIC X(62)    VALUE SPACES.

00079      01 WS-CONT-LINEA.
00080          05 CONTADOR-IMPRESO PIC Z,ZZ9.
00081          05 FILLER         PIC X(15)    VALUE " ETIQUETAS IMPRESAS".
00082          05 FILLER         PIC X(112)   VALUE SPACES.

00085      PROCEDURE DIVISION.

00087          MOVE "NO " TO INTER-FIN-FICHERO
00088          MOVE ZERO TO WS-CONT-ETIQ
00089
00090          OPEN   INPUT          MAESTRO-CORREO
00091          OUTPUT          FICHERO-ETIQUETAS-CORREO
00092
00093          PERFORM 030-IMPRIME-2-ETIQUETAS
00094              2 TIMES
00095
00096          PERFORM 020-PROD-ETIQ-CORREO
00097              UNTIL INTER-FIN-FICHERO IS EQUAL TO "SI"
00098
00099          PERFORM 040-PROD-LINEA-CONT
00100
00101          CLOSE  MAESTRO-CORREO
00102              FICHERO-ETIQUETAS-CORREO
00103          STOP RUN
00104

00106      010-ENTRADA-2-REGISTROS.

00107          MOVE SPACES TO WS-LINEA-NOMBRE
00108              WS-LINEA-DOMICILIO
00109              WS-LINEA-RESTO
00110
00111          READ MAESTRO-CORREO RECORD
00112              AT END
00113                  MOVE "SI" TO INTER-FIN-FICHERO
00114
00115          IF INTER-FIN-FICHERO = "NO"
00116
00117          ADD 1 TO WS-CONT-ETIQ
00118              MOVE NOMBRE CORREO      TO NOMBRE-1
00119              MOVE DOMICILIO-CORREO  TO DOMICILIO-1
00120              MOVE CIUDAD-CORREO     TO CIUDAD-1
00121

```

Fig. 6-17 (cont.)

```

00122      MOVE PAIS-CORREO      TO PAIS-1
00123      MOVE CODIGO-POSTAL-CORREO  TO CODIGO-POSTAL-1
00124
00125      READ MAESTRO-CORREO RECORD
00126          AT END
00127          MOVE "SI" TO INTER-FIN-FICHERO
00128
00129      IF INTER-FIN-FICHERO EQUAL "NO"
00130
00131          ADD 1 TO WS-CONT-ETIQ
00132          MOVE NOMBRE-CORREO      TO NOMBRE-2
00133          MOVE DOMICILIO-CORREO   TO DOMICILIO-2
00134          MOVE CIUDAD-CORREO       TO CIUDAD-2
00135          MOVE PAIS-CORREO        TO PAIS-2
00136          MOVE CODIGO-POSTAL-CORREO TO CODIGO-POSTAL 2,
00137

00139      020-PROD-ETIQ-CORREO.
00140
00141      PERFORM 010-ENTRADA-2-REGISTRO
00142
00143      PERFORM 030-IMPRIME-2-ETIQUETAS
00144
00145
00146      030-IMPRIME-2-ETIQUETAS.
00147
00148          WRITE LINEA-ETIQUETA
00149              FROM WS-LINEA-NOMBRE
00150              BEFORE ADVANCING 1 LINE
00151          WRITE LINEA-ETIQUETA
00152              FROM WS-LINEA-DOMICILIO
00153              BEFORE ADVANCING 1 LINE
00154          WRITE LINEA-ETIQUETA
00155              FROM WS-LINEA-RESTO
00156              BEFORE ADVANCING 2 LINES
00157
00158
00159      040-PROD-LINEA-CONT.
00160
00161      MOVE WS-CONT-ETIQ TO CONTADOR-IMPRESO
00162      WRITE LINEA-ETIQUETA
00163          FROM WS-CONT-LINEA
00164          AFTER ADVANCING 2 LINES
00165

```

Fig. 6-17 (cont.)

NATHAN CONVERSE  
709 N. SIMPSON ST.  
MIAMI

FL 10022

BARNEY PERELMAN  
782 N. SOUTH ST.  
NEW YORK

NY 10012

ABE SHARP  
783 S. NORTH ST.  
BALTIMORE

MD 22145

MIKE PERELMAN  
213 E. SOUTH ST.  
HANOVER

MD 23451

KATHY BUCHER  
666 W. EAST ST.  
PHILADELPHIA

PA 34256

5 ETIQUETAS IMPRESAS

Fig. 6-18

- 6.113 Escriba un programa que calcule parámetros estadísticos de las ventas basados en una muestra de las ventas mensuales de almacenes al detalle. La entrada consiste en registros lógicos contenido: identificativo del almacén, código de localidad e importe mensual vendido. La salida debe consistir en un informe impreso con una línea detallando información para cada establecimiento, seguidas de una última línea que dé para la muestra las *ventas medias* y la *desviación típica* de las ventas.

Estas cantidades se obtienen a partir de las fórmulas

$$\text{total muestra} = s_1 + s_2 + \dots + s_n$$

$$\text{media} = \frac{\text{total muestra}}{n} = \frac{s_1 + s_2 + \dots + s_n}{n}$$

$$\text{desviación típica} = \sqrt{\frac{n(s_1^2 + s_2^2 + \dots + s_n^2) - (s_1 + s_2 + \dots + s_n)^2}{n(n-1)}}$$

donde  $n$  es el número de almacenes de la muestra y  $s_i$  es el importe de las ventas de  $i$ -ésimo almacén.

*Respuesta.* Véase la Figura 6-19 (y la salida típica de la Fig. 6-20). Aunque no era necesario, el programa imprime una linea con las ventas totales.

```

00001 IDENTIFICATION DIVISION.
00002
00003 PROGRAM-ID. SALESTAT.
00004 AUTHOR. LARRY NEWCOMER.
00005 INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS.
00006 DATE-WRITTEN. MAY 1983.
00007 DATE-COMPILED. MAY 9,1983.
00008 SECURITY. NONE.
00009
00010 * SALESTAT PRODUCE UN LISTADO DE VENTAS MENSUALES POR
00011 * ALMACENES. IMPRIME UNA ULTIMA LINEA CON EL TOTAL DE
00012 * VENTAS, MEDIA Y DESVIACION TIPICA.
00013
00014 * ESTE PROGRAMA ILUSTRADA CONVENIENTEMENTE EL USO DE
00015 * LA INSTRUCCION COMPUTE.
00016
00017 ENVIRONMENT DIVISION.
00018
00019 CONFIGURATION SECTION.
00020 SOURCE-COMPUTER. IBM-370.
00021 OBJECT-COMPUTER. IBM-370.
00022 INPUT-OUTPUT SECTION.
00023 FILE-CONTROL.
00024   SELECT FICHERO-VENTAS      ASSIGN TO VENTAS.
00025   SELECT INFORME-VENTAS     ASSIGN TO INFOVENT.
00026
00027 DATA DIVISION.
00028
00029 FILE SECTION.
00030
00031 FD  FICHERO-VENTAS
00032   RECORD CONTAINS 80 CHARACTERS
00033   LABEL RECORDS ARE OMITTED
00034
00035 01  ENTRADA-VENTAS          PIC X(80).
00036
00037 FD  INFORME-VENTAS
00038   RECORD CONTAINS 132 CHARACTERS
00039   LABEL RECORDS ARE OMITTED
00040   LINAGE IS 10
00041     WITH FOOTING AT 8
00042     LINES AT TOP 2
00043     LINES AT BOTTOM 2
00044
00045 01  LINEA-VENTAS          PIC X(132).
00046
00047 WORKING-STORAGE SECTION.
00048
00049 01  WS-AREAS-FECHA-HORA.
00050    05  WS-FECHA-NORMAL.

```

Fig. 6-19

```

00052      10 WS-AA          PIC 99.
00053      10 WS-MM          PIC 99.
00054      10 WS-DD          PIC 99.
00055
00056 01 INTERRUPTORES-CONTADORES-SUMAS.
00057      05 INTER-FIN-VENTAS    PIC X(3).
00058      05 WS-NUM-PAGINA     PIC S9(3)   COMP-3.
00059      05 WS-TOTAL-VENTAS   PIC S9(7)V99  COMP-3.
00060      05 WS-NUM-ALMACENES  PIC S9(5)   COMP-3.
00061      05 WS-SUMA-VENTAS-CUADRADO  PIC S9(15)V99  COMP-3.
00062
00063 01 WS-AREA-VENTAS.
00064      05 WS-ID-ALMACEN-VENTAS  PIC X(5).
00065      05 WS-LOCALIZACION-VENTAS  PIC XX.
00066      05 WS-VENTAS-VENTAS    PIC S9(6)V99.
00067      05 FILLERA          PIC X(65).
00068
00069 01 WS-AREA-LINEA.
00070      05 WS-ID-ALMACEN-LINEA  PIC X(5).
00071      05 FILLER            PIC X(5)    VALUE SPACES.
00072      05 WS-LOCALIZACION-LINEA  PIC XX.
00073      05 FILLER            PIC X(5)    VALUE SPACES.
00074      05 WS-VENTAS-LINEAS    PIC ZZZ,ZZZ.99.
00075      05 FILLER            PIC X(105)  VALUE SPACES.
00076
00077 01 WS-AREA-CABECERA-1.
00078      05 FILLER            PIC X(1)    VALUE SPACES.
00079      05 FILLER            PIC X(20)
00080
00081      05 CABECERA-MM        PIC Z9.
00082      05 FILLER            PIC X        VALUE "/".
00083      05 CABECERA-DD        PIC 99.
00084      05 FILLER            PIC X        VALUE "/".
00085      05 CABECERA-AA        PIC 99.
00086      05 FILLER            PIC X(4)    VALUE SPACES.
00087      05 FILLER            PIC X(6)    VALUE "PAGINA".
00088      05 WS-CABECERA-NUM-PAGINA  PIC ZZ9.
00089      05 FILLER            PIC X(90)   VALUE SPACES.
00090
00091 01 WS-AREA-CABECERA-2.
00092      05 FILLER            PIC X(8)    VALUE "ID ALMAC".
00093      05 FILLER            PIC X(2)    VALUE SPACES.
00094      05 FILLER            PIC X(12)   VALUE "LOCALIDAD".
00095      05 FILLER            PIC X(10)   VALUE "VENTAS".
00096      05 FILLER            PIC X(100)  VALUE SPACES.
00097
00098 01 WS-AREA-PIE.
00099      05 FILLER            PIC X(15)   VALUE SPACES.
00100      05 WS-TOTAL-PIE      PIC Z,ZZZ,ZZZ.99.
00101      05 FILLER            PIC X(105)  VALUE " **".
00102
00103 01 WS-FINAL-LINEA-1.
00104      05 FILLER            PIC X(15)   VALUE SPACES.
00105      05 WS-FINAL-TOTAL    PIC Z,ZZZ,ZZZ.99.
00106      05 FILLER            PIC X(105)  VALUE " ***".
00107
00108 01 WS-FINAL-LINEA-2.
00109      05 FILLER            PIC X(10)   VALUE "MEDIA--".
00110      05 WS-MEDIA-VENTAS   PIC ZZZ,ZZZ.99.
00111      05 FILLER            PIC X(6)    VALUE " SD-- ".
00112      05 WS-SD              PIC ZZZ,ZZZ.99.
00113      05 FILLER            PIC X(96)   VALUE SPACES.
00115
PROCEDURE DIVISION.
MOVE "NO " TO INTER-FIN-VENTAS

```

Fig. 6-19 (cont.)

```

00118      MOVE ZERO TO WS-NUM-PAGINA
00119          WS-TOTAL-VENTAS
00120          WS-NUM-ALMACENES
00121          WS-SUMA-VENTAS-CUADRADO
00122
00123      ACCEPT WS-FECHA-NORMAL      FROM FECHA
00124
00125      MOVE WS-MM              TO CABECERA-MM
00126      MOVE WS-DD              TO CABECERA-DD
00127      MOVE WS-AA              TO CABECERA-AA
00128
00129      OPEN    INPUT           FICHERO-VENTAS
00130          OUTPUT            INFORME-VENTAS
00131
00132      PERFORM COGE-REGISTRO-VENTAS
00133
00134      PERFORM PROD-CABECERA-PAGINA
00135
00136      PERFORM PROD-LINEA-VENTAS
00137          UNTIL INTER-FIN-VENTAS IS EQUAL TO "SI"
00138
00139      PERFORM PROD-LINEAS-FINAL
00140
00141      CLOSE   FICHERO-VENTAS
00142          INFORME-VENTAS
00143      STOP RUN
00144
00145
00146      COGE-REGISTRO-VENTAS.
00147
00148      READ FICHERO-VENTAS RECORD
00149          INTO WS-AREA-VENTAS
00150          AT END
00151          MOVE "SI" TO INTER-FIN-VENTAS
00152
00153
00154
00155      PROD-LINEA-VENTAS.
00156
00157      ADD 1 TO WS-NUM-ALMACENES
00158      COMPUTE WS-SUMA-VENTAS-CUADRADO = WS-SUMA-VENTAS-CUADRADO
00159          + WS-VENTAS-VENTAS ** 2
00160
00161      ADD WS-VENTAS-VENTAS TO WS-TOTAL-VENTAS
00162
00163      MOVE WS-ID-ALMACEN-VENTAS      TO WS-ID-ALMACEN-LINEA
00164      MOVE WS-LOCALIZACION-VENTAS   TO WS-LOCALIZACION-LINEA
00165      MOVE WS-VENTAS-VENTAS       TO WS-VENTAS-LINEA
00166
00167      WRITE LINEA-VENTAS
00168          FROM WS-AREA-LINEA
00169          AFTER ADVANCING 2 LINES
00170          AT END-OF-PAGE
00171          PERFORM PROD-PIE-PAGINA
00172          PERFORM PROD-CABECERA-PAGINA
00173
00174
00175      PERFORM COGE-REGISTRO-VENTAS
00176
00177
00178      PROD-CABECERA-PAGINA.
00179
00180      ADD 1 TO WS-NUM-PAGINA
00181      MOVE WS-NUM-PAGINA TO WS-CABECERA-NUM-PAGINA
00182      WRITE LINEA-VENTAS
00183          FROM WS-AREA-CABECERA-1

```

Fig. 6-19 (cont.)

```

00184          AFTER ADVANCING PAGE
00185          WRITE LINEA-VENTAS
00186          FROM WS-AREA-CABECERA-2
00187          AFTER ADVANCING 1 LINE
00188
00189
00190          PROD-PIE-PAGINA.
00191
00192          MOVE WS-TOTAL-VENTAS TO WS-TOTAL-PIE
00193
00194          WRITE LINEA-VENTAS
00195          FROM WS-AREA-PIE
00196          AFTER ADVANCING 2 LINES
00197
00198
00199          PROD-LINEAS-FINAL.
00200
00201          MOVE WS-TOTAL-VENTAS      TO WS-FINAL-TOTAL
00202          WRITE LINEA-VENTAS
00203          FROM WS-FINAL-LINEA-1
00204          AFTER ADVANCING 2 LINES
00205
00206          COMPUTE WS-MEDIA-VENTAS = WS-TOTAL-VENTAS / WS-NUM-ALMACENES
00207
00208          COMPUTE WS-SD =((WS-NUM-ALMACENES * WS-SUMA-VENTAS-CUADRADO
00209          - WS-TOTAL-VENTAS ** 2)
00210          /
00211          (WS-NUMERO-ALMACENES * (WS-NUMERO-ALMACENES - 1))) ** .5
00212
00213          WRITE LINEA-VENTAS
00214          FROM WS-FINAL-LINEA-2
00215          AFTER ADVANCING 2 LINES
00216

```

Fig. 6-19 (cont.)

ESTADISTICA VENTAS		5/09/83	PAGINA	1
ID ALMAC	LOCALIDAD	VENTAS		
00001	AA	1,000.00		
00002	BB	2,000.00		
00003	CC	3,000.00		
		6,000.00 *		
ESTADISTICA VENTAS		5/09/83	PAGINA	2
ID ALMAC	LOCALIDAD	VENTAS		
00004	DD	4,000.00		
00005	EE	5,000.00		
00006	FF	6,000.00		
		21,000.00 *		
ESTADISTICA VENTAS		5/09/83	PAGINA	3
ID ALMAC	LOCALIDAD	VENTAS		
00007	GG	7,000.00		
00008	HH	8,000.00		
		36,000.00 **		
MEDIA--	4,500.00	SD--	2,449.49	

Fig. 6-20

## Ejercicios de programación

- 6.114** Dado un principal P, invertido en una cuenta de ahorros a un tipo de interés anual I que se compone C veces al año, el importe total acumulado en N años es

$$T = P * \left(1 + \frac{I}{C}\right)^{C \cdot N}$$

Escriba un programa COBOL en el que se dé entrada a P, I, C y N e imprima un informe mostrando P, I, C, N y T. Debe escribir tantas líneas como valores de entrada haya. Diseñe sus propios registros de entrada y el formato del informe, pero utilice la cláusula LINAGE para asegurarse de que: (i) haya 20 líneas a doble espacio por página lógica, (ii) haya márgenes superior e inferior de 3 líneas cada una, (iii) se imprima un título del informe y cabeceras para las columnas en la parte superior de cada cuerpo de página (incluido el número de página, la fecha y la hora).

- 6.115** Escriba un programa en COBOL que dé entrada a registros de 80 bytes de un inventario con el siguiente formato:

columnas 1-6 : número de artículo
7-26 : descripción del artículo
27-31 : existencias
32-38 : precio (2 lugares decimales)
resto : sin uso

Imprima un informe con una línea por cada artículo mostrando:

artículo	descripción	existencias	precio	valor-existencias
#				

donde valor-existencias es existencias \* precio. Al final del informe imprima una línea mostrando: (1) el valor total del inventario (suma del valor de todos los artículos) y (2) precio medio de un artículo (suma de los valores dividido por suma de las existencias). Diseñe su propia página lógica que incluya: (i) cabeceras, (ii) un área de pie de página en la que se imprima el valor de todas las existencias hasta ese momento (Fig. 6-20) y (iii) número de página, fecha y hora en la parte superior de cada página.

- 6.116** Una firma de consultoría tiene para cada empleado un *salario-hora* para el empleado y un *precio-hora* que cobra a los clientes por una hora de trabajo de ese empleado. Dada la siguiente entrada:

columnas 1-4 : ID empleado
5-9 : salario-hora (2 posiciones decimales)
10-14 : precio-hora (2 posiciones decimales)
15-17 : horas trabajadas a la semana (1 posición decimal)

imprima un informe de la nómina mostrando:

ID	horas	salario	salario	precio	importe	beneficio
empleado	trabajadas	hora	total	hora	total	bruto

donde

$$\begin{aligned} \text{salario-total} &= \text{horas} * \text{salario-hora} \\ \text{importe-total} &= \text{horas} * \text{precio-hora} \\ \text{beneficio-bruto} &= \text{importe-total} - \text{salario-total} \end{aligned}$$

Al final del informe, imprima una línea mostrando el salario total de todos los empleados, el importe total y el beneficio medio por empleado. La página lógica debe incluir (i) cabecera, (ii) un área de pie de página con el número de empleados procesados hasta el momento, (iii) número de página, fecha y hora en la parte superior de la página.

# Capítulo 7

## Lógica de los programas

### 7.1 DISEÑO LOGICO: DIAGRAMAS DE FLUJO ESTRUCTURADOS

La *lógica del programa* es el orden en el que la computadora ejecuta las instrucciones del mismo (a diferencia del orden físico en el listado del programa fuente). Un *diagrama de flujo estructurado* es un gráfico que describe el orden de operaciones en el algoritmo. Un diagrama de flujo estructurado puede, por tanto, utilizarse para diseñar la lógica de un problema, *antes* de escribir el programa en COBOL (o en cualquier otro lenguaje). Si se va actualizando el diagrama de flujo con todos los cambios en el programa, sirve también como documentación.

Los símbolos que suelen utilizarse en los diagramas de flujo se muestran en la Figura 7-1.

	<i>Símbolo terminal</i> , con las palabras “PRINCIPIO” o “FIN” y que identifica el comienzo o el final del algoritmo. También puede escribirse el nombre del algoritmo en el interior.
	<i>Símbolo de procesamiento</i> , que indica que la computadora lleva a cabo algún procesamiento de la información. La naturaleza del procesamiento se describe en el interior del símbolo.
	<i>Símbolo de decisión</i> , que indica una decisión de la computadora que debe elegir entre dos alternativas del algoritmo. Dos flechas salen del símbolo para indicar las dos alternativas.
	<i>Símbolo de procesamiento predefinido</i> , que representa un procesamiento suficientemente complejo como para requerir un diagrama separado que lo describa. En el símbolo se escribe el nombre de la rutina a la que representa.
	<i>Símbolo de entrada/salida</i> , que sirve para indicar una operación de entrada/salida.
	<i>Símbolo conector</i> , que representa la unión entre dos o más partes de un diagrama de flujo.
	Las <i>flechas</i> se utilizan para conectar los símbolos anteriores en el orden en el que deban ejecutarse.

Fig. 7-1

## 7.2 ESTRUCTURAS LOGICAS DE PROGRAMACION

Se puede demostrar que la lógica de cualquier algoritmo o programa se puede realizar como combinación de las tres *estructuras lógicas* fundamentales.

### Estructura secuencial

Implica la ejecución secuencial de procedimientos o instrucciones, uno después de otro. En un programa con estructura secuencial, el orden lógico de las instrucciones coincide con su orden físico. El diagrama de flujo de la Figura 7-2 muestra una estructura secuencial.

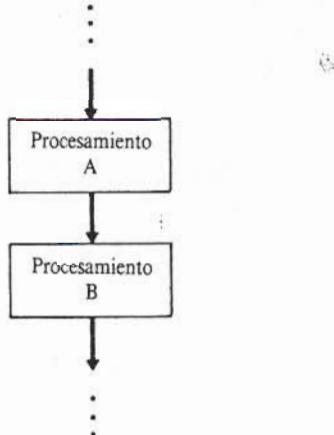


Fig. 7-2

### Estructura de selección

Esta estructura muestra exactamente la elección de una entre dos "ramas" diferentes (procesamientos o conjuntos de instrucciones). La rama que se elige depende del valor de una "condición" que es siempre *verdadera* o *falsa*.

**EJEMPLO 7.1** En un programa de nóminas se necesita un conjunto de instrucciones para calcular el salario bruto si un empleado trabaja 40 horas o menos, y se precisa de otro conjunto de instrucciones si el empleado hace horas extras. La estructura de selección se muestra en el diagrama de flujo de la Figura 7-3.

El símbolo de decisión debe contener una *condición* que sea *verdadera* o *falsa*. Como en este caso, en la condición se suelen comparar dos elementos (o un elemento y un literal) para determinar si se da una determinada *relación* (en este caso "menor o igual que"). Si la condición es verdadera, se sigue la ejecución por la rama marcada con **SI** (o con **VERDADERO**); en otro caso se continúa con la rama marcada con **NO** (o **FALSO**).

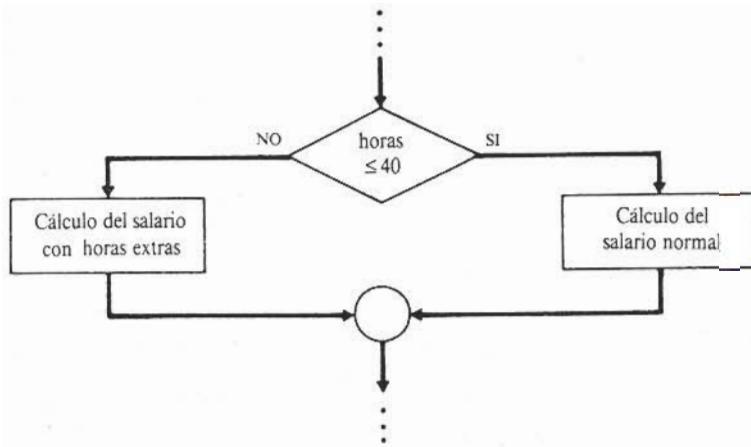


Fig. 7-3

**EJEMPLO 7.2** El diagrama de flujo estructurado de la Figura 7-4 combina la estructura secuencial con la estructura de selección. Los bloques 1, 2 y 9 forman una estructura secuencial, el bloque 2 es una estructura de selección (con alternativas a través de los bloques 3 y 4). El bloque 4 es un procesamiento predefinido que representa un conjunto de procesamientos (posiblemente complejos) definidos en otro diagrama de flujo. El bloque 3 es la cabeza de otra estructura de selección con caminos alternativos a través de los bloques 5 ó 6, 7 y 8 (estos últimos representan una estructura secuencial).

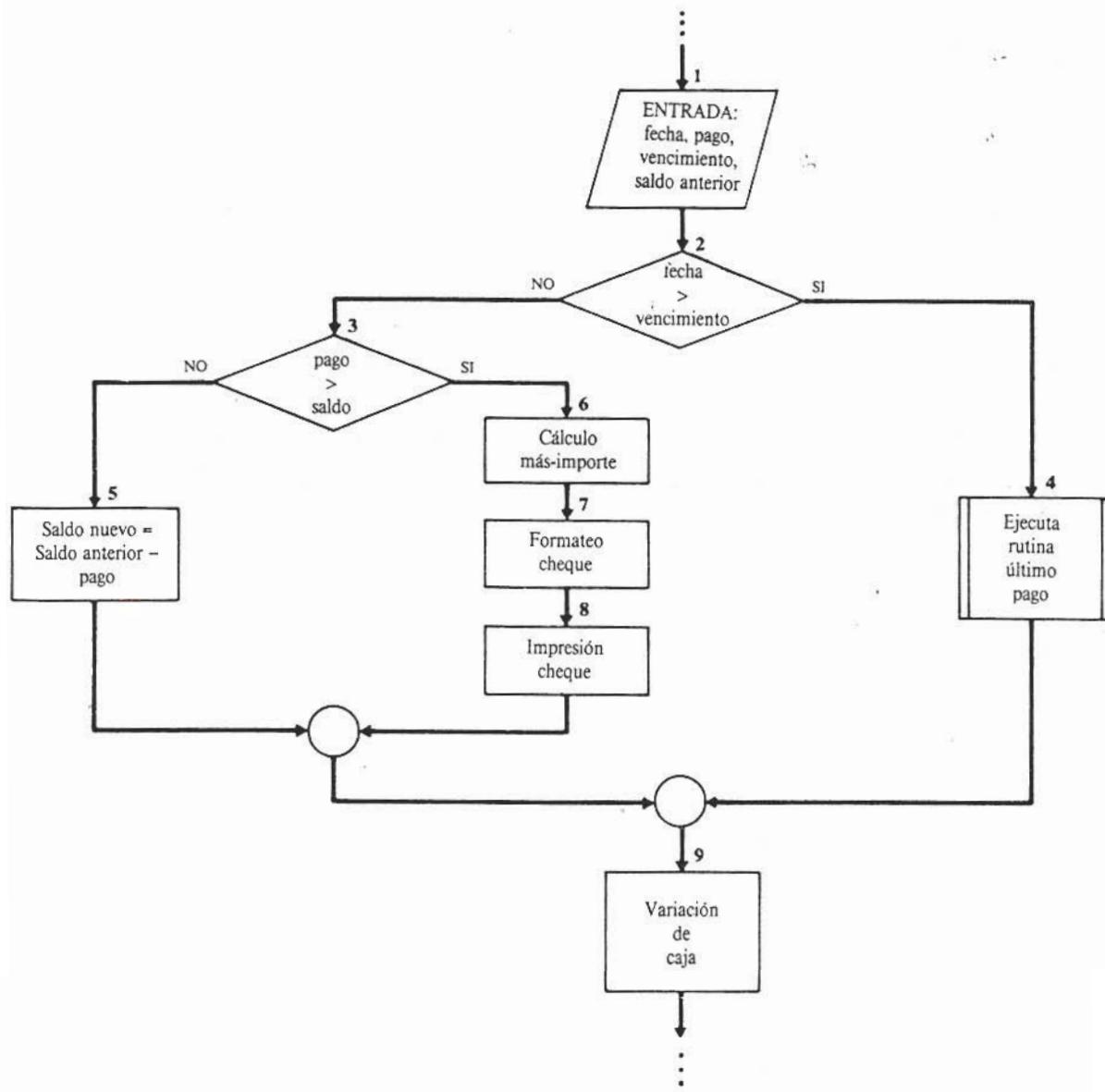


Fig. 7-4

### Estructura iterativa

La estructura iterativa se utiliza cuando un procedimiento o grupo de procesamientos tienen que *repetirse* un cierto número de veces; es lo que también se conoce como *bucle*.

**EJEMPLO 7.3** La versión de la estructura iterativa del diagrama de flujo de la Figura 7-5 se conoce como *estructura DO WHILE*. Al entrar por primera vez en la estructura lo primero que se hace es decidir si se "repite" una vez más. Si la decisión es NO (la condición es falsa), se abandona la estructura sin ejecutar el procesamiento A. Si la decisión es SI (la condición es verdadera), se ejecuta el procesamiento A y se vuelve a tomar la misma decisión.

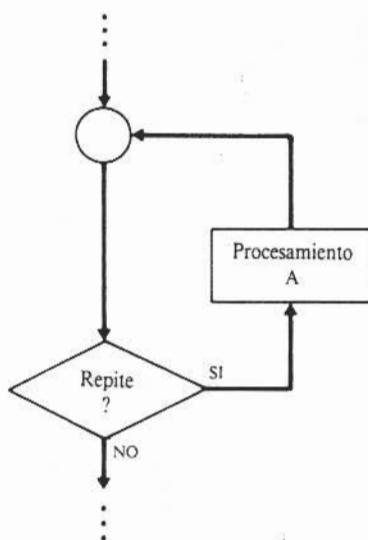


Fig. 7-5

En la Figura 7-6 se muestra una estructura DO WHILE particular; la misma hace que 'Pon "NO"' en A e 'IMPRIME: "Esto realmente imprime"' se ejecuta *una vez*. El hecho de que A tome el valor "NO" durante el bucle no hace que se aborte la ejecución. Sólo cuando se llega al bloque de decisión la condición A = "NO" hace que no se produzcan nuevas repeticiones.

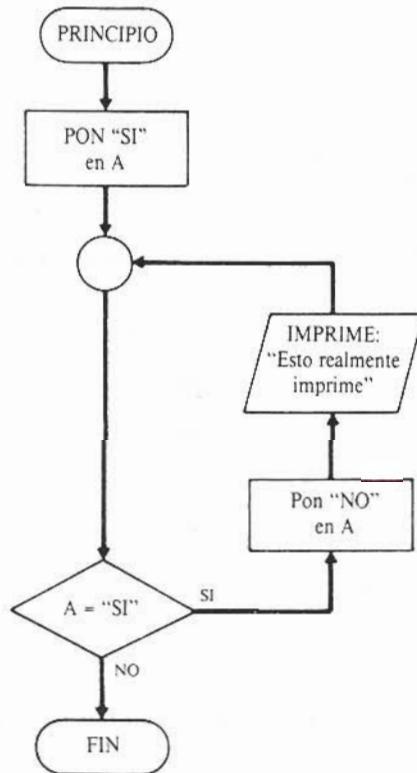


Fig. 7-6

**EJEMPLO 7.4** El diagrama de flujo estructurado para el programa del Ejemplo 2.17 se da en la Figura 7-7; contiene sólo estructuras secuenciales e iterativas.

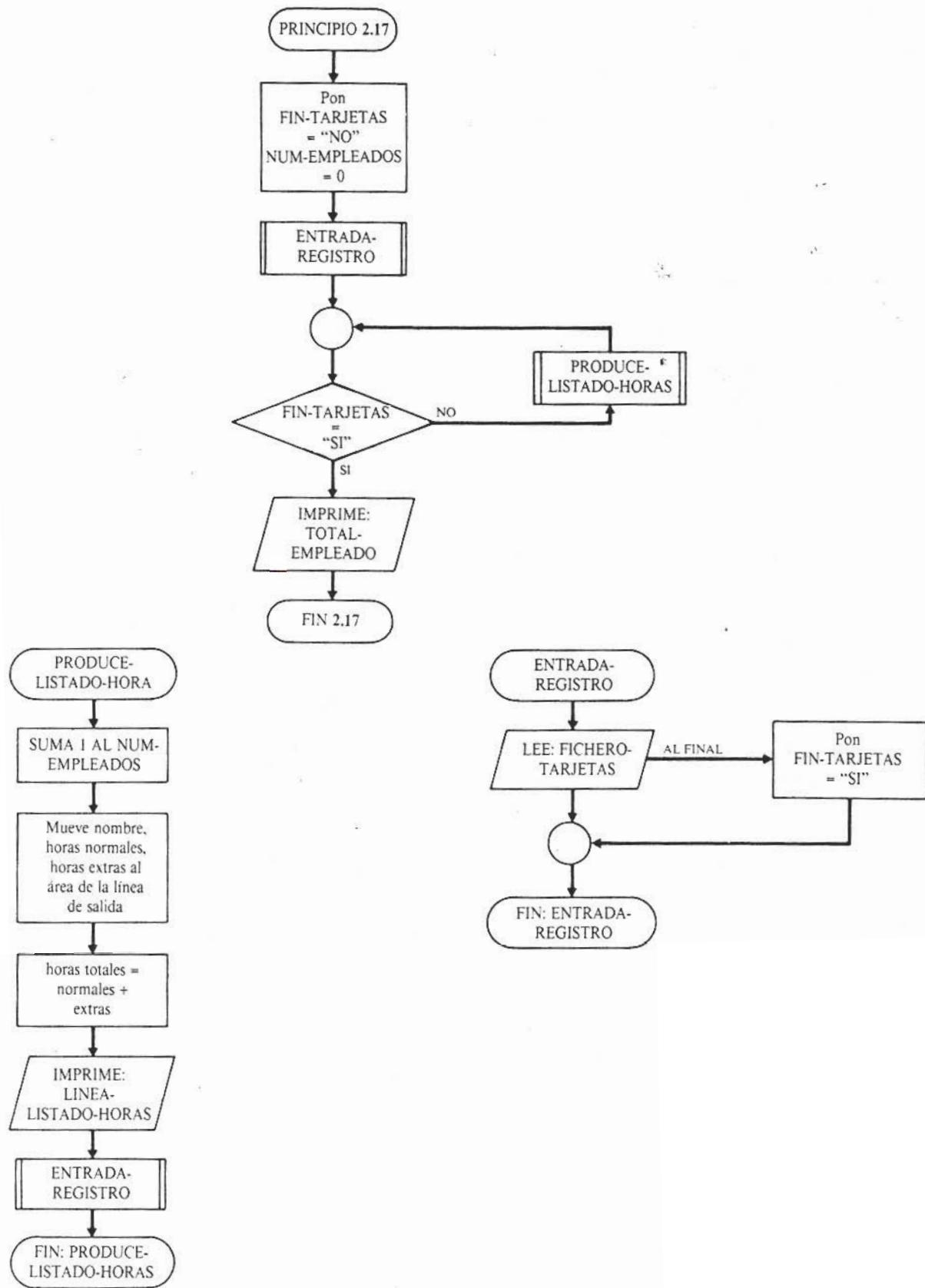


Fig. 7-7

Observe cómo cada procesamiento predefinido tiene su propio diagrama de flujo; esto se corresponde con los párrafos de la instrucción PERFORM en el programa en COBOL. Observe también que ciertas acciones de la división de procedimientos, como la apertura y cierre de ficheros, no aparecen en el diagrama de flujo.

### 7.3 LA ESTRUCTURA SECUENCIAL EN COBOL

La estructura secuencial es la más fácil de programar (en cualquier lenguaje). Para obtenerla en COBOL sólo hay que escribir una instrucción detrás de otra en la división de procedimientos.

### 7.4 LA ESTRUCTURA DE SELECCION EN COBOL

Esta estructura se implementa en COBOL por medio de la *instrucción IF*, cuya forma general es

IF condición { instrucción-1  
                  |  
                  | NEXT SENTENCE } [ ELSE instrucción-2  
                  |  
                  | ELSE NEXT SENTENCE ] .

Aquí, “condición” es una expresión más o menos complicada, que se tratará con detalle en las Secciones 7-5, 7-9.

#### EJEMPLO 7.5

```
IF HORAS-TRABAJADAS GREATER THAN 40.0
    COMPUTE SALARIO-BRUTO = 40.0 * SALARIO-HORA +
                           (HORAS-TRABAJADAS - 40.0) * 1.5 * SALARIO-
                           HORA
ELSE
    COMPUTE SALARIO-BRUTO = HORAS-TRABAJADAS * SALARIO-HORA
```

La “condición” es “HORAS-TRABAJADAS GREATER THAN 40.0”. Es verdadera o falsa dependiendo del contenido de HORAS-TRABAJADAS. Si la condición fuera verdadera, se ejecutaría la “instrucción-1” (“COMPUTE... 40.0\*...”) y la computadora se dirigiría después a cualquier instrucción que siga al punto que marca el final de la instrucción IF. Si la condición fuera falsa, se ignoraría la “instrucción-1” y se ejecutaría la “instrucción-2” (“COMPUTE... HORAS-TRABAJADAS \*...”), después se pasaría a la instrucción que sigue al punto que marca el final de la instrucción IF.

El desplazamiento de líneas, aunque no lo requiere el compilador, es una norma de codificación esencial. Las palabras “IF” y “ELSE” se escriben en la misma columna; las rutinas de IF y de ELSE se desplazan de las líneas con “IF” y “ELSE”. Debe utilizarse siempre un punto estucturado para marcar *siempre* el final de cada instrucción IF.

La cláusula ELSE es opcional. Cuando se omite, las condiciones verdaderas hacen que se ejecute la “instrucción-1” antes de que la computadora pase a la instrucción que sigue al punto que marca el final de IF; una condición falsa haría que la computadora *saltara* la “instrucción-1” y fuera directamente a la instrucción que sigue al punto que marca el final de IF.

#### EJEMPLO 7.6 Nunca es necesario especificar “ELSE NEXT SENTENCE”, puesto que

```
IF...
    PERFORM...
ELSE
    NEXT SENTENCE
```

WRITE...

es de idénticos efectos que

```
IF...
    PERFORM...
WRITE...
```

En cualquiera de los dos casos, PERFORM... y después WRITE... se ejecutan si la condición es verdadera; si la condición es falsa, sólo se ejecuta la instrucción WRITE...

No obstante, la cláusula "ELSE NEXT SENTENCE" es *conveniente* en ocasiones; véase el Ejemplo 7.25.

#### EJEMPLO 7.7

```

IF IMPORTE-VENDIDO IS GREATER THAN CUOTA-VENTAS
    NEXT SENTENCE
ELSE
    MOVE "SI" TO INTER-CARTA-FELICITACION
    SUBTRACT IMPORTE-VENDIDO FROM CUOTA-VENTAS GIVING
        IMPORTE-BAJO-CUOTA

    WRITE LINEA-INFORME
        FROM WS-LINEA-COMPORT-VENTAS

```

Si IMPORTE-VENDIDO es mayor que CUOTA-VENTAS, se ejecuta "NEXT SENTENCE" haciendo que la computadora *se dirija a la instrucción que sigue al punto que marca el final de la instrucción IF ("WRITE LINEA-INFORME...")*. Si IMPORTE-VENDIDO no es mayor que CUOTA-VENTAS, la computadora ignora lo que sigue a IF y va a la parte de ELSE ejecutando "MOVE..." y "SUBTRACT..." antes de dirigirse a WRITE...

Observe las serias consecuencias de la omisión del punto al final de la instrucción IF: La instrucción WRITE se convierte en parte de ELSE... y la línea de impresión sólo se escribe para quienes están bajo cuota, en lugar de para todos los vendedores como se pretendía.

**EJEMPLO 7.8** Nunca es necesario especificar "NEXT SENTENCE" en la primera parte de la instrucción IF: siempre se puede negar la condición a intercambiar la parte de IF con la parte de ELSE. Después de hacer esto, la parte de ELSE consiste en "NEXT SENTENCE" y se puede omitir (véase el Ejemplo 7.6).

Aplicando esta técnica al Ejemplo 7.7, se obtiene:

```

IF IMPORTE-VENDIDO IS NOT GREATER THAN CUOTA-VENTAS
    MOVE "SI" TO INTER-CARTA-FELICITACION
    SUBTRACT IMPORTE VENDIDO FROM CUOTA-VENTAS GIVING
        IMPORTE-BAJO-CUOTA

    WRITE LINEA-INFORME
        FROM WS-LINEA-COMPORT-VENTAS

```

Un error pequeño en la instrucción IF del Ejemplo 7.7 es que se envía una carta de felicitación también a quienes hayan vendido exactamente lo que marcaba su cuota. Para evitarlo se debería poner:

```
IF IMPORTE-VENDIDO IS LESS THAN CUOTA-VENTAS
```

#### 7.5 ESTRUCTURA DE SELECCION: CONDICION DE CLASE DE DATOS

La condición de clase de datos se puede utilizar para determinar si un elemento contiene datos numéricos o alfanuméricos. La sintaxis de la condición es:

identificador IS [NOT] {NUMERIC  
                  ALPHABETIC}

Las comprobaciones "NUMERIC" y "NOT NUMERIC" sólo se pueden realizar con datos que tengan formatos PICTURE alfanuméricos ("X") o numéricos ("SV9") y cuyo USAGE sea DISPLAY (o COMP-3, si su formato PICTURE es numérico y se dispone de COMP-3). Las comprobaciones "ALPHABETIC" y "NOT ALPHABETIC" se pueden aplicar sólo a elementos cuyo formato PICTURE sea alfabético ("A") o alfanumérico ("X") y cuyo USAGE sea DISPLAY.

La condición

identificador IS NUMERIC

es verdadera si y sólo si el elemento contiene sólo los dígitos de 0 a 9. Si la cláusula PICTURE contiene un signo operativo (PIC "S"), el elemento debe contener también un signo para ser considerado numérico. Si no hay signo en PICTURE, el elemento no debe contener signo.

La condición

identificador IS ALPHABETIC

es verdadera si y sólo si el elemento contiene las letras de la A la Z y/o espacios en blanco.

El uso principal de la comprobación de clase es *validar* campos de entrada en registros tecleados por seres humanos. El programa que valida los datos debe generar mensajes de error descriptivos con suficiente información para corregir los errores.

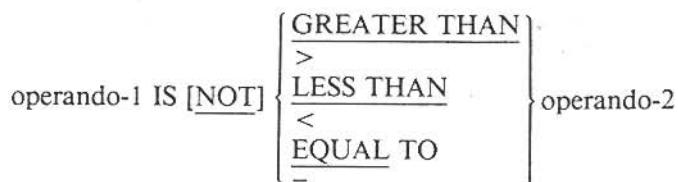
#### EJEMPLO 7.9

```
IF IMPORTE-PAGO-PARCIAL NUMERIC
    SUBTRACT IMPORTE-PAGO-PARCIAL FROM SALDO-ANTERIOR GIVING
        SALDO-NUEVO
ELSE
    MOVE SALDO-ANTERIOR TO SALDO-NUEVO
    MOVE "SI" TO INTER-PAGO-NO-PROCESADOS
```

IMPORTE-PAGO-PARCIAL es un campo de entrada tecleado por un ser humano y que se valida antes de ser utilizado en un cálculo. El campo SALDO-ANTERIOR no se valida porque procede de un fichero en disco creado por un programa de computadora. Si el programa está diseñado apropiadamente y completamente depurado, se puede asumir que los ficheros que crea contienen datos correctos (es posible, incluso, que el programa validase los datos antes de escribirlos en el fichero).

#### 7.6 ESTRUCTURA DE SELECCION: CONDICION RELACION

La sintaxis de la *condición relación* es:



A excepción de ciertas combinaciones prohibidas (véase el Ejemplo 7.13), "operando-1" y "operando-2" pueden ser elementos de datos, literales o expresiones aritméticas (como las que aparecen en el miembro derecho de la instrucción COMPUTE).

#### EJEMPLO 7.10

- IF A + B \*\* 2 LESS THAN (A - C) \* (A + B)

Cualquier expresión aritmética válida para la instrucción COMPUTE es también válida para la condición relación.

- IF NUM-ID-EMPLEADO EQUAL HIGH-VALUES

A menudo se utilizan constantes figurativas en las relaciones.

- IF SALARIO-BRUTO GREATER THAN 900.00

Pueden incluirse literales numéricos en la condición relación.

- IF COD-REGISTRO-ENTRADA = "A"

En esta condición relación se tiene un literal no-numérico y la opción de usar el signo "=".

- IF A NOT GREATER THAN B  
IF C NOT LESS THAN D

En COBOL, las relaciones "menor o igual que ( $\leq$ )" y "mayor o igual que ( $\geq$ )" deben expresarse mediante la negación de la relación contraria.

- IF COD-REGISTRO-ENTRADA NOT EQUAL "C"

Las palabras "IS" y "TO" son opcionales en la condición relación.

¿Qué se entiende en COBOL al decir que un elemento es menor que otro?

**Comparaciones numéricas.** Los formatos USAGE y PICTURE no afectan al resultado de la comparación, que se basa estrictamente en el valor algebraico de los elementos. Los puntos decimales se alinean automáticamente para la comparación. (Las mismas reglas que mejoran la eficiencia de los cálculos sirven para mejorar la eficiencia de las comparaciones numéricas; véase la Sección 6.18.)

#### EJEMPLO 7.11 Dado

```
01 EJEMPLO-DATOS.
 05 A    PIC S9999V99   COMP-3      VALUE +25.67.
 05 B    PIC S99V999    COMP        VALUE +25.670.
 05 C    PIC 99V99     DISPLAY     VALUE 25.67.
 05 D    PIC S99V99     VALUE       VALUE -99.00.
```

A, B y C se consideran *iguales* (todos contienen el valor algebraico +25.67). D es menor que cualquiera de ellos porque es negativo.

**Comparaciones no numéricas.** En este caso "es menor que" queda definido por la secuencia de orden de caracteres que esté en activo durante la ejecución del programa (Secciones 4.3 y 4.4). Las comparaciones no numéricas se efectúan *un byte cada vez y de izquierda a derecha*. Si coinciden los dos elementos byte a byte, se considerarán *iguales*. Si en alguna posición no coinciden, el resultado se determina de acuerdo con el orden de los bytes diferentes en la secuencia de orden. Si los dos elementos tienen distinta longitud, se considera que el más corto se expande con espacios en blanco por la derecha hasta igualar a la longitud del más largo.

#### EJEMPLO 7.12

- "CAB" es menor que "CAT" porque aunque coinciden los dos primeros bytes, los terceros son diferentes y "B" es menor que (precede a) "T" en la secuencia de orden.
- "CAB" es igual que "CABbb" porque el elemento más corto "CAB" se expande con dos bytes por la derecha para realizar la comparación.
- "COB" es mayor que "CAT" porque no coinciden los segundos caracteres y "O" es mayor que "A" en la secuencia de orden. Observe que "B" y "T" no llegan a compararse, puesto que el resultado queda determinado por el segundo byte.
- "CAB" es menor que "CABBY" porque "CAB" se expande con dos blancos por la derecha para efectuar la comparación, y la primera diferencia tiene lugar con el primero de estos dos espacios en blanco ("CABbb" se compara con "CABBY"). Como el espacio en blanco es menor que "B" en la secuencia de orden, "CAB" es menor que "CABBY".

Se ve que las comparaciones no numéricas preservan el orden alfabético convencional.

**EJEMPLO 7.13** Como sucedía con la instrucción MOVE, se prohíbe mezclar elementos numéricos con elementos no numéricos en la condición relación. En particular, tome nota de lo siguiente:

- (1) Los datos numéricos que sean COMP o COMP-3 sólo pueden compararse con elementos numéricos (DISPLAY, COMP ó COMP-3).
- (2) No pueden compararse dos literales o constantes figurativas.
- (3) Los elementos numéricos cuyo formato USAGE sea DISPLAY se pueden comparar con elementos alfanuméricos (PIC X), pero los resultados pueden ser distintos de los que se esperan. Si necesita hacer una comparación de este tipo, consulte el manual del sistema.

## 7.7 ESTRUCTURA DE SELECCION: CONDICION SIGNO

La condición signo sólo se puede utilizar con operando numéricos; su sintaxis es

operando IS [NOT] {POSITIVE  
ZERO  
NEGATIVE}

### EJEMPLO 7.14

- IF SALDO-ACTUAL IS POSITIVE

La condición es verdadera sólo cuando el saldo es mayor que cero. Una condición equivalente sería:

IF SALDO-ACTUAL IS GREATER THAN ZERO

- IF SALDO-ACTUAL - IMPORTE-PAGADO IS NOT NEGATIVE

Se está utilizando una expresión aritmética en una condición signo, su equivalente sería:

IF SALDO-ACTUAL - IMPORTE-PAGADO IS NOT LESS THAN ZERO

- IF NUM-CUENTAS-DESCUBIERTO IS ZERO

Equivalente a:

IF NUM-CUENTAS-DESCUBIERTO IS EQUAL TO ZERO

- IF 12.34 IS POSITIVE

Inválido: no se pueden utilizar ni literales ni constantes figurativas como operando único de la condición signo.

- IF TALLA - 12.54 IS NOT NEGATIVE

Se pueden utilizar literales y constantes figurativas en una expresión aritmética en la condición signo.

## 7.8 ESTRUCTURA DE SELECCION: CONDICION NOMBRE-DE-CONDICION

La condición nombre-de-condición sólo puede utilizarse en la división de procedimientos en sustitución de una condición relación. Un elemento de datos puede tener asociados con él uno o más nombre-de-condición; cada uno se define en la división de datos *inmediatamente después* del referido elemento por medio de una entrada especial al nivel 88 con el siguiente formato:

88 nombre-de-condición {VALUE IS  
VALUES ARE} literal-1 [ {THROUGH  
THRU} literal-2 ]  
[ literal-3 [ {THROUGH  
THRU} literal-4 ] ] ...

Entonces la condición "nombre-de-condición" es *verdadera si y sólo si el elemento de datos en cuestión contiene el valor especificado por la cláusula VALUE IS...*

**EJEMPLO 7.15** Vamos a considerar una determinada tarea (a) sin, y (b) con el uso de condiciones nombre-de-condición.

(a) 01 REGISTRO-CLIENTE-ACTUALIZADO.

05 CODIGO-CLIENTE	PIC X.
05 NOMBRE-CLIENTE	PIC X(20).
05 DOMICILIO-CLIENTE	PIC X(30).

.....  
01 INTER-FIN-FICHERO PIC X(3).

.....  
READ FICHERO-CLIENTES-ACTUALIZADO  
AT END  
MOVE "SI" TO INTER-FIN-FICHERO

```

IF INTER-FIN-FICHERO NOT EQUAL "SI"
  IF CODIGO-CLIENTE EQUAL "A"
    PERFORM RUTINA-NUEVO-CLIENTE
  ELSE IF CODIGO-CLIENTE EQUAL "D"
    PERFORM RUTINA-BORRADO-CLIENTE
  ELSE IF CODIGO-CLIENTE EQUAL "C"
    PERFORM RUTINA-CAMBIO-DOMICILIO
  ELSE
    PERFORM RUTINA-CODIGO-INVALIDO

```

El primer byte de REGISTRO-CLIENTE-ACTUALIZADO contiene un código que indica si el registro contiene información sobre: un nuevo cliente para el FICHERO-MAESTRO-CLIENTES, una baja de un cliente que debe borrarse del fichero o un cliente cuyo nombre o domicilio debe modificarse en el fichero. Este código se comprueba con una serie de instrucciones IF que determinan el párrafo apropiado que debe ejecutarse con PERFORM para procesar el registro. Se hacen las siguientes observaciones:

- (1) El valor de código aparece como literales en la instrucción IF; si estos valores tuvieran que modificarse, el programador tendría que hacer a través de toda la división de procedimientos para localizar todas las ocurrencias de los literales.
- (2) 'CODIGO-CLIENTE EQUAL "C"' no proporciona mucha información sobre el *significado* de que el código sea igual a "C".

(b)	01 REGISTRO-CLIENTE-ACTUALIZADO.	
	05 CODIGO-CLIENTE	PIC X.
	88 TRANS-NUEVO-CLIENTE	VALUE "A".
	88 TRANS-BAJA-CLIENTE	VALUE "D".
	88 TRANS-CAMBIO-DOMICILIO	VALUE "C".
	05 NOMBRE-CLIENTE	PIC X(20).
	05 DOMICILIO-CLIENTE	PIC X(30).
<hr/>		
	01 INTER-FIN-FICHERO	PIC X(3).
	88 NO-MAS-REGISTROS	VALUE "SI".
	88 SI-MAS-REGISTROS	VALUE "NO".
<hr/>		
	READ FICHERO-CLIENTES-ACTUALIZADO	
	AT END	
	MOVE "SI" TO INTER-FIN-FICHERO	
	 IF SI-MAS-REGISTROS	
	IF TRANS-NUEVO-CLIENTE	
	PERFORM RUTINA-NUEVO-CLIENTE	
	ELSE IF TRANS-BAJA-CLIENTE	
	PERFORM RUTINA-BORRADO-CLIENTE	
	ELSE IF TRANS-CAMBIO-DOMICILIO	
	PERFORM RUTINA-CAMBIO-DOMICILIO	
	ELSE	
	PERFORM RUTINA-CODIGO-INVALIDO	

La cláusula VALUE es necesaria para los datos al nivel 88, pero *no se permite ninguna otra cláusula*. Esto tiene sentido, porque un nombre-de-condición se asocia con un *valor* particular del elemento de datos bajo el cual se define.

Observe cómo las condiciones relación de (a) se reemplazan por condiciones nombre-de-condición; por ejemplo,

```

IF CODIGO-CLIENTE EQUAL "A"
se reemplaza por su equivalente exacto
IF TRANS-NUEVO-CLIENTE

```

Se obtienen tres beneficios del uso de los nombres-de-condición:

- (1) Sin que haya literales en la división de procedimientos, el programa es más fácil de modificar en caso necesario. Todo lo que se necesita es cambiar la entrada al nivel 88 en la división de datos.
- (2) Los nombres-de-condición pueden ser descriptivos de la condición que representan. Así, "TRANS-CAMBIO-DOMICILIO" le dice más al lector que 'CODIGO-CLIENTE EQUAL "C"'.
- (3) Un nombre-de-condición puede representar a una condición relación bastante complicada.

#### EJEMPLO 7.16

01	REGISTRO-EXPEDIENTE.		
05	EXP-ID-ESTUDIANTE	PIC X(9).	
05	AÑOS-INGRESO	PIC 9.	
88	NOVATO	VALUE 1.	
88	NUEVO	VALUE 2.	
88	JUNIOR	VALUE 3.	
88	SENIOR	VALUE 4.	
88	GRADUADO	VALUE 1 THRU 4.	
88	LICENCIADO	VALUE 5 THRU 9.	
05	EXP-ASIGN-APROBADAS	PIC 99.	
05	EXP-NOTA-MEDIA	PIC 9V9.	
88	NOTABLE	VALUE 3.8 THRU 4.0.	
88	APROBADO	VALUE 3.5 THRU 3.7.	
88	SUSPENSO	VALUE 0.0 THRU 1.9.	
05	EXP-NOMBRE-ESTUDIANTE	PIC X(30).	

El uso de la opción THRU permite especificar un *rango* de valores en un nombre-de-condición. La condición GRADUADO es verdadera si AÑOS-INGRESO tiene un valor entre 1 y 4 inclusive, etc.

Algunos programadores prefieren escribir el nivel 88 en el margen A para hacerlo visible; otros le consideran como otro nivel normal [véase el Ejemplo 7.15(b)].

Observe que se entiende mucho mejor

IF SENIOR AND APROBADO

que la condición relación equivalente

IF AÑOS-INGRESO EQUAL 4 AND  
EXP-NOTA-MEDIA NOT LESS THAN 3.5 AND  
EXP-NOTA-MEDIA NOT GREATER THAN 3.7

**EJEMPLO 7.17** Para un nombre-de-condición se puede especificar más de un rango de valores:

05	CODIGO-LOCALIDAD	PIC XX.	
88	PHILADELPHIA	VALUE "AA" THRU "AC" "B7" "D5" "E3" THRU "E5".	

Dada la definición anterior se puede utilizar

IF PHILADELPHIA

en lugar de su complicado equivalente

IF (CODIGO-LOCALIDAD NOT LESS THAN "AA"  
AND NOT GREATER THAN "AC")  
OR (CODIGO-LOCALIDAD EQUAL "B7" OR "D5")  
OR (CODIGO-LOCALIDAD NOT LESS THAN "E3"  
AND NOT GREATER THAN "E5")

**EJEMPLO 7.18** Dadas las definiciones

01	INTER-FIN-FICHERO	PIC XXX.	
88	NO-MAS-REGISTROS	VALUE "SI".	
88	MAS-REGISTROS	VALUE "NO".	

se resaltan tres errores de principiante.

- READ FICHERO-EJEMPLO  
AT END  
MOVE "SI" TO NO-MAS-REGISTROS

(en lugar de lo correcto que es MOVE "SI" TO INTER-FIN-FICHERO). Los nombres-de-condición no son elementos de datos, sino la representación de una condición relación. La instrucción

MOVE "SI" TO NO-MAS-REGISTROS

tiene tan poco sentido como

MOVE "SI" TO (INTER-FIN-FICHERO IS EQUAL TO "SI")

- IF NO-MAS-REGISTROS IS EQUAL TO "SI"

NO-MAS-REGISTROS es ya una condición relación (en forma condensada); lo que se pretende es

IF NO-MAS-REGISTROS

o

IF INTER-FIN-FICHERO IS EQUAL TO "SI"

## 7.9 OPERADORES LOGICOS Y CONDICIONES COMPUESTAS

El operador lógico "NOT" se puede utilizar para negar una *condición simple* (de clase, relación o nombre-de-condición) que no contenga ella misma un "NOT".

### EJEMPLO 7.19

- IF NOT A IS LESS THAN B

La condición que se está negando es "A IS LESS THAN B"; por tanto, sería más claro escribir la instrucción IF así:

IF NOT (A IS LESS THAN B)

que es equivalente desde un punto de vista lógico a

IF A IS NOT LESS THAN B

Esta última versión, con el "NOT" en la condición simple, es preferible en las condiciones de clase, signo y relación.

- IF NOT REGISTRO-EN-BUFFER

"REGISTRO-EN-BUFFER" debe ser un nombre-de-condición. Si REGISTRO-EN-BUFFER es verdadera, NOT REGISTRO-EN-BUFFER es falsa; si REGISTRO-EN-BUFFER es falsa, NOT REGISTRO-EN-BUFFER es verdadera.

Además de con la negación, se pueden combinar las condiciones simples mediante el uso de los operadores lógicos AND y OR para formar así *condiciones compuestas*. El valor de "condición-1 AND condición-2" depende de los valores de condición-1 y condición-2 como se ve en la Figura 7-8.

<i>condición-1</i>	<i>condición-2</i>	<i>condición-1 AND condición-2</i>
verdadera	verdadera	verdadera
verdadera	falsa	falsa
falsa	verdadera	falsa
falsa	falsa	falsa

Fig. 7-8

Observe que el resultado es una única condición, que es verdadera sólo si las dos condiciones son verdaderas.

Cuando se combinan dos condiciones mediante OR, la condición compuesta sigue la tabla de verdad de la Figura 7-9.

<i>condición-1</i>	<i>condición-2</i>	<i>condición-1 OR condición-2</i>
verdadera	verdadera	verdadera
verdadera	falsa	verdadera
falsa	verdadera	verdadera
falsa	falsa	falsa

Fig. 7-9

Observe que la condición compuesta es falsa sólo si las dos condiciones son falsas.

#### EJEMPLO 7.20

- IF HORAS-TRABAJADAS NUMERIC AND HORAS-TRABAJADAS GREATER THAN 40.0  
La condición compuesta es verdadera si y sólo si las condiciones simples lo son. Esta condición particular podría originar problemas. En el COBOL del sistema IBM OS/VS, las condiciones de relación se evalúan antes que las condiciones de clase. Por tanto, si HORAS-TRABAJADAS no es numérico, la evaluación de "HORAS-TRABAJADAS GREATER THAN 40.0" podría hacer abortar al programa. La dificultad se elimina utilizando dos instrucciones IF:

```
IF HORAS-TRABAJADAS NUMERIC
    IF HORAS-TRABAJADAS GREATER THAN 40.0
```

Ahora la condición de relación sólo se evalúa si HORAS-TRABAJADAS es numérico.

- IF HORAS-TRABAJADAS NOT NUMERIC OR SALARIO-HORA NOT NUMERIC
 

```
PERFORM RUTINA-ERROR
ELSE
    PERFORM CALCULO-SALARIO
```

Si una de las condiciones simples es verdadera, la condición compuesta es verdadera. Por tanto, la RUTINA-ERROR se ejecuta si HORAS-TRABAJADAS o SALARIO-HORA (cualquiera de las dos) es no numérico; si ambos campos son numéricos, se ejecuta CALCULO-SALARIO.

- IF HORAS-TRABAJADAS NUMERIC AND SALARIO-HORA NUMERIC
 

```
PERFORM CALCULO-SALARIO
ELSE
    PERFORM RUTINA-ERROR
```

Equivalente a la anterior. Debe verificar mediante tablas de verdad que lo opuesto (negación) de "A OR B" es "(NOT A) AND (NOT B)" y que lo opuesto de "A AND B" es "(NOT A) OR (NOT B)".

- IF ENTRADA-REGISTRO AND SALDO-DEUDA IS POSITIVE  
En este caso se combinan con AND un nombre-de-condición y una condición signo.
- IF EXISTENCIAS NOT LESS THAN 1 OR EXISTENCIAS NOT GREATER THAN 500
 

```
PERFORM AJUSTE-EXISTENCIAS
```

Error. Se adivina que se pretendía ejecutar PERFORM AJUSTE-EXISTENCIAS cuando el contenido de EXISTENCIAS no esté comprendido entre 1 y 500 inclusive. Sin embargo, un poco de reflexión lleva a la conclusión de que la condición es *siempre* verdadera (la única forma de que fuera falsa es que EXISTENCIAS fuera a la vez menor que 1 y mayor que 500, lo que es imposible); por tanto, *siempre* se ejecutaría PERFORM. La formulación correcta es:

```
IF EXISTENCIAS LESS THAN 1 OR EXISTENCIAS GREATER THAN 500
    PERFORM AJUSTE-EXISTENCIAS
```

Se puede utilizar más de un operador lógico para formar una condición compuesta. Al repetir *el mismo* operador no se necesitan paréntesis en la condición compuesta (puesto que el resultado no depende del orden de las operaciones). Ahora bien, cuando se mezclan operadores *diferentes*, el orden de operaciones debe fijarse mediante paréntesis.

**EJEMPLO 7.21**

- IF     AÑO-ENTRADA IS GREATER THAN 80  
 AND AÑO-ENTRADA IS LESS THAN 84  
 AND MES-ENTRADA IS NOT LESS THAN 1  
 AND MES-ENTRADA IS NOT GREATER THAN 12  
 AND DIA-ENTRADA IS NOT LESS THAN 1  
 AND DIA-ENTRADA IS NOT GREATER THAN 31  
 PERFORM PROCESO-FECHA-VALIDA  
 ELSE  
 PERFORM PROCESO-FECHA-INVALIDA

Las condiciones simples se escriben cada una en una línea y se alinean los operadores. Las seis condiciones simples anteriores tienen que ser verdaderas para que la condición compuesta sea verdadera. Así pues, para que se ejecute PERFORM PROCESA-FECHA-VALIDA, el año debe estar comprendido entre 81 y 83, el mes debe estar comprendido entre 1 y 12 y el dia debe estar comprendido entre 1 y 31. Aunque en esta comprobación se detectan la mayor parte de los errores, otros se pasan por alto (por ejemplo, 31/09/83).

- IF     NUM-EMPLEADO-TARJETA-HORA NOT NUMERIC  
 OR HORAS-NORMALES-TARJETA-HORA NOT NUMERIC  
 OR HORAS-EXTRA-TARJETA-HORA  
 MOVE "SI" TO INTER-CAMPOS-INVALIDOS

Esta rutina valida los campos numéricos del REGISTRO-TARJETA-HORA. Observe cómo cada condición simple se escribe en una línea y que los operadores quedan alineados. Como se utiliza "OR", la condición compuesta será verdadera si una de las condiciones simples es verdadera.

- IF     (NOT TERMINOS-ESPECIALES-EXIGIDOS)  
 AND (MAL-CREDITO  
 OR IMPORTE-COMPRADO GREATER THAN LIMITE-CREDITO)

"TERMINOS-ESPECIALES-EXIGIDOS" y "MAL-CREDITO" deben ser nombres-de-condición. La secuencia de evaluaciones es: (i) se evalúa TERMINOS-ESPECIALES-EXIGIDOS; (ii) se niega el resultado de (i); (iii) se evalúa IMPORTE-COMPRADO GREATER THAN LIMITE-CREDITO; (iv) se evalúa MAL-CREDITO; (v) los resultados de (iii) y (iv) se concatenan mediante OR; (vi) los resultados de (ii) y (v) se concatenan mediante AND.

- IF     (NUEVO-CLIENTE  
 AND (FICHERO-ID-CLIENTE OR  
 ID-CLIENTE NOT NUMERIC))  
 OR  
 (VIEJO-CLIENTE  
 AND (CLIENTE-NO-EN-FICHERO  
 OR DIAS-RETRASO-PAGO GREATER THAN 90))

Si las condiciones compuestas llegan a ser tan complicadas, *muy probablemente convenga rediseñar el programa total o parcialmente.*

Si no se utilizan los paréntesis (o si las condiciones están en el mismo nivel de paréntesis), las expresiones lógicas se efectúan en el siguiente orden: (1) operaciones aritméticas que aparezcan en las condiciones de relación; (2) condiciones simples en el orden: relación, clase, nombre-de-condición, signo; (3) negación de condiciones simples en el orden anterior; (4) AND; (5) OR. Si dos condiciones son equivalentes en este orden de prelación, se evalúan de izquierda a derecha.

Si las reglas anteriores parecen complicadas, hay una forma de evitar la más remota posibilidad de error: *use paréntesis.*

**EJEMPLO 7.22** Sin paréntesis, la tercera condición compuesta del Ejemplo 7.21 sería

```
IF NOT TERMINOS-ESPECIALES-EXIGIDOS
  AND MAL-CREDITO
  OR IMPORTE-COMPRADO GREATER THAN LIMITE-CREDITO
```

El orden de evaluación es el siguiente: (i) IMPORTE-COMPRADO GREATER THAN LIMITE-CREDITO (condición de relación); (ii) TERMINOS-ESPECIALES-EXIGIDOS (nombre-de-condición); (iii) MAL-CREDITO (nombre-de-condición); (iv) NOT (ii) (negación de la condición simple); (v) (iv) AND (iii) (AND antes que OR); (vi) (v) OR (i). El resultado es el mismo que si se usaran los siguientes paréntesis:

```
IF ((NOT TERMINOS-ESPECIALES-EXIGIDOS)
    AND MAL-CREDITO)
    OR (IMPORTE-COMPRADO GREATER THAN LIMITE-CREDITO)
```

Los paréntesis del Ejemplo 7.21 son necesarios para lograr el efecto deseado (que es distinto del anterior).

## 7.10 ABREVIATURAS DE LAS CONDICIONES COMPUESTAS

La forma general de una condición de relación simple es:

sujeto operador-relacional objeto

Una condición compuesta combina dos o más condiciones de relación simples por medio de los operadores lógicos AND y OR:

sujeto operador-relacional objeto { AND } sujeto operador-relacional objeto  
OR

Si el *operador-relacional* es el mismo en dos condiciones simples de una condición compuesta, no es preciso repetirlo. Del mismo modo, si el sujeto es el mismo en varias relaciones simples de una relación compuesta, no es necesario repetirlo. El objeto debe escribirse siempre.

### EJEMPLO 7.23

- IF A EQUAL B OR A EQUAL C OR A EQUAL D

En este ejemplo se repiten el sujeto (elemento "A") y el operador relacional ("EQUAL"). Ambas repeticiones pueden omitirse consiguiéndose el mismo resultado.

IF A EQUAL B OR C OR D

Por supuesto en el primer caso deben detallarse el sujeto y el operador relacional.

- IF A EQUAL B AND A EQUAL C OR A GREATER THAN D  
se puede abreviar quedando

IF A EQUAL B AND C OR GREATER THAN D

Si en una condición simple falta el sujeto o el operador relacional, se asume el *anterior* sujeto u operador relacional.

- IF ELEMENTO-EJEMPLO NOT EQUAL 10.5 OR OTRO-ELEMENTO OR 3.0  
es una abreviatura de

IF ELEMENTO-EJEMPLO NOT EQUAL 10.5  
OR ELEMENTO-EJEMPLO NOT EQUAL OTRO-ELEMENTO  
OR ELEMENTO-EJEMPLO NOT EQUAL 3.0

Las abreviaturas en las condiciones compuestas sólo deben usarse cuando no se oscurece el propósito. Cuando puedan existir dudas, *utilice paréntesis y no use abreviaturas*.

## 7.11 INSTRUCCIONES IF ANIDADAS Y LINEALES

Los algoritmos estructurados se componen de combinaciones de las estructuras secuencial, de selección e iterativa; a menudo tienen estructuras de selección *dentro de* otras estructuras de selección. A la implementación de esto en COBOL se la llama instrucciones IF *anidadas*. Cuando se utilizan los IF anidados, deben desplazarse las líneas adecuadamente para mostrar claramente las relaciones entre las diversas instrucciones IF.

**EJEMPLO 7.24**

```

IF NUEVO-REGISTRO-MAESTRO
  IF YA-EXISTE
    PERFORM ERROR-REG-DUPPLICADO
  ELSE
    PERFORM CREA-NUEVO-REGISTRO
ELSE
  IF CAMBIO-REGISTRO-MAESTRO
    IF YA-EXISTE
      PERFORM MODIFICACION-EN-MAESTRO
    ELSE
      PERFORM ERROR-REG-NO-ENCONTRADO

```

Observe el desplazamiento de cada instrucción IF y cómo las cláusulas ELSE están siempre alineadas con su IF correspondiente. No suele haber problemas al interpretar los IF anidados cuando *cada uno* tiene su cláusula ELSE asociada. Sin embargo, esto no es necesario: "IF CAMBIO-REGISTRO-MAESTRO" no tiene cláusula ELSE asociada.

**EJEMPLO 7.25** Al interpretar el compilador COBOL instrucciones IF anidadas empareja cada cláusula ELSE con la instrucción IF precedente que no haya sido emparejada ya con otro ELSE (si no puede realizar el emparejamiento, debe haber un error sintáctico). Esta regla puede ocasionar problemas cuando las instrucciones IF se escriben omitiendo la cláusula ELSE:

```

IF LECTURA-TRANS-HECHA
  IF IMPORTE-PAGO NUMERIC
    SUBTRACT IMPORTE-PAGO FROM SALDO-ACTUAL
ELSE
  MOVE "FINAL" TO CABECERA-LINEA-TOTAL
  PERFORM PROD-LINEA-TOTAL-FINAL

```

En este caso el compilador empareja "ELSE..." con "IF IMPORTE-PAGO NUMERIC" (en vez de con "IF LECTURA-TRANS-HECHA", como parece haber querido el programador). Por tanto, las instrucciones se ejecutarán como sigue:

```

IF LECTURA-TRANS-HECHA
  IF IMPORTE-PAGO NUMERIC
    SUBTRACT IMPORTE-PAGO FROM SALDO-ACTUAL
ELSE
  MOVE "FINAL" TO CABECERA-LINEA-TOTAL
  PERFORM PROD-LINEA-TOTAL-FINAL

```

Las instrucciones IF anidadas son incorrectas porque la línea total final se produce siempre que se dé entrada a un pago no numérico como parte de un registro de transacciones. El intento original era imprimir la línea total final cuando no se puede dar entrada a más registros de transacciones (fin de fichero).

Para evitar estos errores, algunas instalaciones requieren que cada instrucción IF se escriba con la cláusula ELSE asociada (aunque sea "ELSE NEXT SENTENCE"):

```

IF LECTURA-TRANS-HECHA
  IF IMPORTE-PAGO NUMERIC
    SUBTRACT IMPORTE-PAGO FROM SALDO-ACTUAL
  ELSE
    NEXT SENTENCE
ELSE
  MOVE "FINAL" TO CABECERA-LINEA-TOTAL
  PERFORM PROD-LINEAL-TOTAL-FINAL

```

Si se ejecuta "ELSE NEXT SENTENCE", la computadora pasa a la instrucción que sigue al punto.

A veces un programa tiene que comprobar el mismo campo para determinar si es igual a alguno de una serie. En tales casos se puede utilizar una forma especial de los IF anidados que se llaman *IF lineales*.

**EJEMPLO 7.26**

```

IF CODIGO-TRANSACCION EQUAL 1
    PERFORM CREA-NUEVO-REGISTRO
ELSE
    IF CODIGO-TRANSACCION EQUAL 2
        PERFORM BORRA-REGISTRO
    ELSE
        IF CODIGO-TRANSACCION EQUAL 3
            PERFORM MODIFICA-REGISTRO
        ELSE
            PERFORM RUTINA-CODIGO-TRANS-INVALIDO

```

Esta serie de comprobaciones para CODIGO-TRANSACCION quedan mejor en unos IF lineales:

```

IF CODIGO-TRANSACCION EQUAL 1
    PERFORM CREA-NUEVO-REGISTRO
ELSE IF CODIGO-TRANSACCION EQUAL 2
    PERFORM BORRA-REGISTRO
ELSE IF CODIGO-TRANSACCION EQUAL 3
    PERFORM MODIFICA-REGISTRO
ELSE
    PERFORM RUTINA-CODIGO-TRANS-INVALIDO

```

En los IF lineales cada cláusula ELSE se escribe alineada bajo el IF inicial y cada IF posterior se escribe *en la misma línea* que el ELSE que le precede. Las acciones que tiene que llevar a cabo cada IF figuran en líneas aparte y desplazadas. Observe que la última cláusula ELSE no va seguida de una instrucción IF. Las instrucciones asociadas con esta última cláusula ELSE sólo se ejecutan si el elemento que está siendo comprobado no contiene *ninguno* de los valores chequeados en las condiciones precedentes. Observe también que sólo se ejecuta un párrafo mediante PERFORM —el que haya de ser depende exclusivamente del valor de CODIGO-TRANSACCION.

**EJEMPLO 7.27** Las instrucciones IF lineales pueden usarse para crear una cuarta estructura lógica, la *estructura caso*. La estructura caso no es *imprescindible*, pero algunos lenguajes disponen de la misma. La estructura caso provoca la ejecución de *una* rutina de entre un conjunto de rutinas, dependiendo del valor de un elemento de control que se comprueba dentro de la estructura. La estructura caso puede aparecer en los diagramas de flujo de diversas formas, una de las cuales se ilustra en la Figura 7-10, relativa al Ejemplo 7.26. (Véase el Apéndice C para estudiar la estructura caso en el COBOL de 1980.)

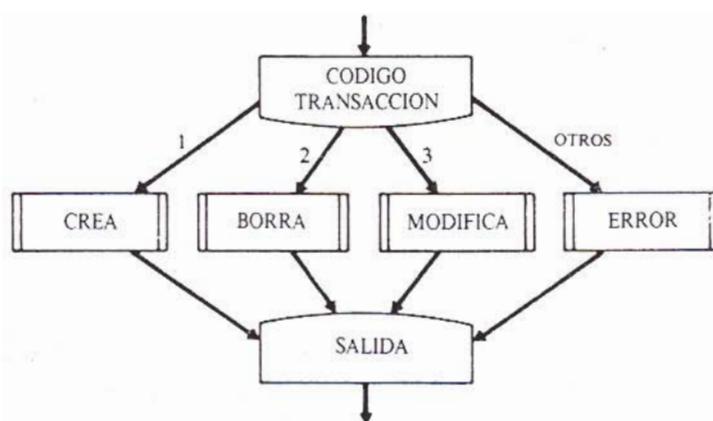


Fig. 7-10

## 7.12 ESTRUCTURA ITERATIVA: INSTRUCCION PERFORM

La instrucción PERFORM hace que un párrafo o sección de la división de procedimientos se ejecute un cierto número de veces. Esta instrucción tiene cuatro formas.

### PERFORM simple

La versión más simple de la instrucción PERFORM es:

PERFORM nombre-procedimiento-1  $\left[ \begin{array}{c} \text{THROUGH} \\ \text{THRU} \end{array} \right]$  nombre-procedimiento-2

Cada "nombre-procedimiento" debe corresponder a un párrafo o una sección.

#### EJEMPLO 7.28

PERFORM PARRAFO-EJEMPLO  
COMPUTE...

.....  
PARRAFO-EJEMPLO.

ADD...  
MOVE...  
SUBTRACT...

PARRAFO-SIGUIENTE

OPEN...  
MOVE...

La instrucción PERFORM hace que se ejecuten *todas* las instrucciones de PARRAFO-EJEMPLO de forma secuencial. Después de la ejecución de SUBTRACT..., la computadora pasa a la instrucción que sigue a PERFORM (COMPUTE...).

Cuando se ejecuta una sección (que puede contener varios párrafos) se comienza con la primera instrucción del primer párrafo y se acaba con la última instrucción del último párrafo.

#### EJEMPLO 7.29 Dados los párrafos definidos en el Ejemplo 7.28,

PERFORM PARRAFO-EJEMPLO THRU PARRAFO-SIGUIENTE  
COMPUTE...

.....

obliga a la computadora a empezar ejecutando las instrucciones del PARRAFO-EJEMPLO de forma secuencial, comenzando con ADD... La ejecución continúa con una instrucción tras otra y un párrafo tras otro hasta que se ejecuta la última instrucción del párrafo que sigue a "THRU", en ese momento la ejecución continuará con la instrucción siguiente a PERFORM (es decir, COMPUTE...).

Cuando se utiliza la opción THRU con secciones, la ejecución comienza con la primera instrucción del primer párrafo de la primera sección y finaliza con la última instrucción del último párrafo de la última sección.

En general es preferible evitar el uso de la opción THRU en *cualquier versión* de la instrucción PERFORM. Como la ejecución de las instrucciones con PERFORM...THRU... depende de su posición física en el programa fuente, existe la posibilidad de que un programador que se ocupe del mantenimiento inserte inadvertidamente un párrafo o sección dentro del rango de una instrucción PERFORM...THRU... El uso de PERFORM...THRU... también dificulta la comprensión del programa fuente, puesto que es necesario recordar dónde empieza y acaba el rango de PERFORM. Hay pocas situaciones en COBOL en las que es necesario recordar dónde empieza y acaba el rango de PERFORM...THRU...; por tanto, tratar de evitarlo no es difícil.

### Versión TIMES

La siguiente versión de la instrucción PERFORM es:

```
PERFORM nombre-procedimiento-1 [ { THROUGH } nombre-procedimiento-2 ]
      { identificador-1 } TIMES
      { entero-1 }
```

La opción THRU juega el mismo papel en la definición del rango de instrucciones a ser ejecutadas que el que jugaba en la instrucción PERFORM simple. De nuevo insistimos en que no se recomienda su uso. Si se utiliza el identificador-1, debe ser un elemento numérico entero (es decir, PIC S9 sin punto decimal). El identificador-1 o el entero-1 indican el *número de veces* que el rango de instrucciones especificado debe ejecutarse antes de que la computadora vaya a la instrucción que sigue a PERFORM. Si el identificador-1 o entero-1 es negativo o cero, *no se ejecuta* el rango de instrucciones y la computadora pasa a la siguiente instrucción.

### EJEMPLO 7.30

```
05 WS-NUMERO-ITERACIONES    PIC 99.
```

```
.....
```

```
ACCEPT WS-NUMERO-ITERACIONES FROM DISPOSITIVO-Opciones-OPERADOR
```

```
.....
```

```
PERFORM IMPRIME-CHEQUE
      WS-NUMERO-ITERACIONES TIMES
```

Aquí se utiliza una variable numérica para indicar el número de iteraciones. DISPOSITIVO-Opciones-OPERADOR debe definirse en el párrafo SPECIAL-NAMES de la división del entorno y configuración. Observe que si el operador introdujera cero en WS-NUMERO-ITERACIONES, la instrucción PERFORM originaría cero ejecuciones de IMPRIME-CHEQUE.

A pesar de su simplicidad, PERFORM...TIMES no se utiliza con frecuencia en los programas en COBOL porque rara vez se conoce de antemano el número de iteraciones a realizar. Más frecuentemente se necesita repetir la ejecución de una rutina hasta que tiene lugar determinada condición que indica el cese de las iteraciones (por ejemplo, el fin de fichero).

### Versión UNTIL

La sintaxis es:

```
PERFORM nombre-procedimiento-1 [ { THROUGH } ] nombre-procedimiento-2
      { THRU }
      UNTIL condición-1
```

El efecto de PERFORM...UNTIL... se describe en el diagrama de flujo estructurado de la Figura 7-11.

Al comparar con la Figura 7-5, se reconoce la estructura DO WHILE (con condición de negación). Todas las características apuntadas en el Ejemplo 7.3 sirven para PERFORM...UNTIL... Una vez más, desaconsejamos el uso de la opción THRU.

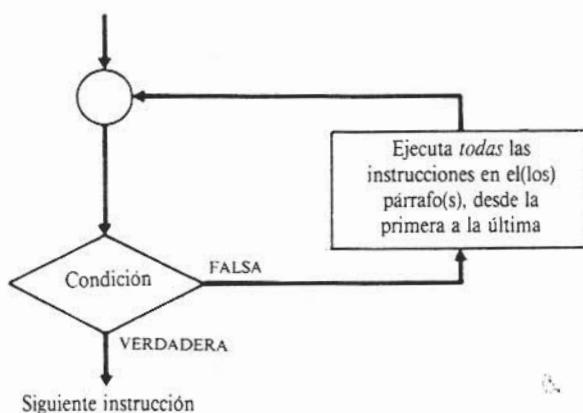


Fig. 7-11

**EJEMPLO 7.31**

- MOVE "NO" TO INTERRUPTOR  
PERFORM IMPRIME-MENSAJE  
UNTIL INTERRUPTOR EQUAL "SI"
- .....

```

IMPRIME-MENSAJE
  WRITE LINEA-IMPRESION
    FROM WS-LINEA-MENSAJE
  
```

Un "bucle infinito": después de tomar el valor "NO", INTERRUPTOR nunca valdrá "SI" y por tanto la instrucción WRITE de IMPRIME-MENSAJE se ejecutará repetida e incesantemente. En la práctica, la mayor parte de los sistemas operativos establecen límites en el tiempo que un programa puede hacer uso de la CPU y también en el número de líneas impresas que puede producir: si se rebasa cualquiera de estos límites, el programa abortará produciéndose un error ABEND.

- PERFORM DETERMINA-NUEVO-SALDO  
UNTIL PROCESADAS-TODAS-TRANS

"PROCESADAS-TODAS-TRANS" debe ser un nombre-de-condición (y el nombre-de-condición debe hacerse *verdadero* en algún punto durante la iteración de DETERMINA-NUEVO-SALDO).

- PERFORM LISTA-ARTICULOS  
UNTIL LISTADOS-TODOS-ELEMENTOS  
OR LINEAS-RESTANTES IS NEGATIVE  
OR TIEMPO-TOTAL IS GREATER THAN 20.0

Cualquier condición compuesta que pueda usarse en la instrucción IF puede también utilizarse en PERFORM... UNTIL... En este caso, mediante el operador lógico OR, se conectan un nombre de condición (LISTADOS-TODOS-ELEMENTOS), una condición signo y una condición de relación. Fíjese en cómo se escribe este ejemplo.

**Versión VARYING**

Esta versión de la instrucción PERFORM es la más complicada sintácticamente, pero es útil muchas veces al trabajador con tablas (véase el Capítulo 10) o al escribir bucles en los que se repite la ejecución de un párrafo con un elemento cuyo contenido varía de forma controlada en cada repetición. El formato general de PERFORM...VARYING... aparece en la Figura 7-12 (los nombres-de-índice se comentan en el Capítulo 10; no se preocupe por ellos por ahora).

$\underline{\text{PERFORM}} \text{ nombre-procedimiento-1} \left[ \begin{cases} \text{THROUGH} \\ \underline{\text{THRU}} \end{cases} \right] \text{ nombre-procedimiento-2 } \right]$   
 $\underline{\text{VARYING}} \left\{ \begin{array}{l} \text{identificador-1} \\ \text{nombre-\'indice-1} \end{array} \right\} \underline{\text{FROM}} \left\{ \begin{array}{l} \text{literal-2} \\ \text{identificador-2} \\ \text{nombre-\'indice-2} \end{array} \right\}$   
 $\underline{\text{BY}} \left\{ \begin{array}{l} \text{literal-3} \\ \text{identificador-3} \end{array} \right\} \underline{\text{UNTIL}} \text{ condición-1}$   
 $\left[ \underline{\text{AFTER}} \left\{ \begin{array}{l} \text{identificador-4} \\ \text{nombre-\'indice-4} \end{array} \right\} \underline{\text{FROM}} \left\{ \begin{array}{l} \text{literal-5} \\ \text{identificador-5} \\ \text{nombre-\'indice-5} \end{array} \right\} \right.$   
 $\underline{\text{BY}} \left\{ \begin{array}{l} \text{literal-6} \\ \text{identificador-6} \end{array} \right\} \underline{\text{UNTIL}} \text{ condición-2}]$   
 $\left[ \underline{\text{AFTER}} \left\{ \begin{array}{l} \text{identificador-7} \\ \text{nombre-\'indice-7} \end{array} \right\} \underline{\text{FROM}} \left\{ \begin{array}{l} \text{literal-8} \\ \text{identificador-8} \\ \text{nombre-\'indice-8} \end{array} \right\} \right.$   
 $\underline{\text{BY}} \left\{ \begin{array}{l} \text{literal-9} \\ \text{identificador-9} \end{array} \right\} \underline{\text{UNTIL}} \text{ condición-3}]$

Fig. 7-12

**EJEMPLO 7.32**

```

PERFORM ASIGNA-BUZON
  VARYING NUMERO-BUZON FROM 1 BY 1
  UNTIL PROCESADOS-TODOS-EMPLEADOS
  .....
ASIGNA-BUZON.
  READ FICHERO-MAESTRO-EMPLEADOS
    INTO WS-REG-EMPLEADO
    AT END
      MOVE "SI" TO INTER-TODOS-EMPLEADOS

IF ENTRADA-OTRO-EMPLEADO
  MOVE NUMERO-BUZON TO WS-NUM-BUZON-SALIDA
  MOVE WS-NOMBRE-EMPLEADO TO WS-NOMBRE-SALIDA
  WRITE LINEA-LISTADO-BUZON
    FROM WS-LINEA-SALIDA

```

Esta rutina está diseñada para asignar buzones de la compañía (numerados 1, 2, 3, etc.) a los empleados. La versión VARYING de PERFORM se utiliza para controlar el NUMERO-BUZON que se asigna a cada empleado. NUMERO-BUZON se fija inicialmente con el valor de "FROM", 1. Después se evalúa la condición de UNTIL, y si es falsa, se ejecuta el párrafo ASIGNA-BUZON. Este párrafo lee un registro de empleado y, si no se encuentra al final del fichero, imprime una línea con el nombre de empleado y el contenido del NUMERO-BUZON. Después de ejecutarse la última instrucción de ASIGNA-BUZON, al elemento NUMERO-BUZON se le suma el valor de "BY", es decir, 1, tomando el valor 2. A continuación se vuelve a evaluar la condición de UNTIL, etc. Después de que se haya procesado el último empleado (al que se le habrá asignado el número de buzon  $n$  si hay  $n$  empleados), PROCESADOS-TODOS-EMPLEADOS se hace verdadero ("SI") y la computadora salta a la primera instrucción siguiente a PERFORM.

En este ejemplo, los valores de FROM y BY son iguales. La Figura 7-13 es el diagrama de flujo estructurado correspondiente al caso más general.

```
PERFORM PARRAFO-EJEMPLO
  VARYING ELEMENTO-EJEMPLO FROM VALOR-A BY VALOR-B
  UNTIL CONDICION-EJEMPLO
```

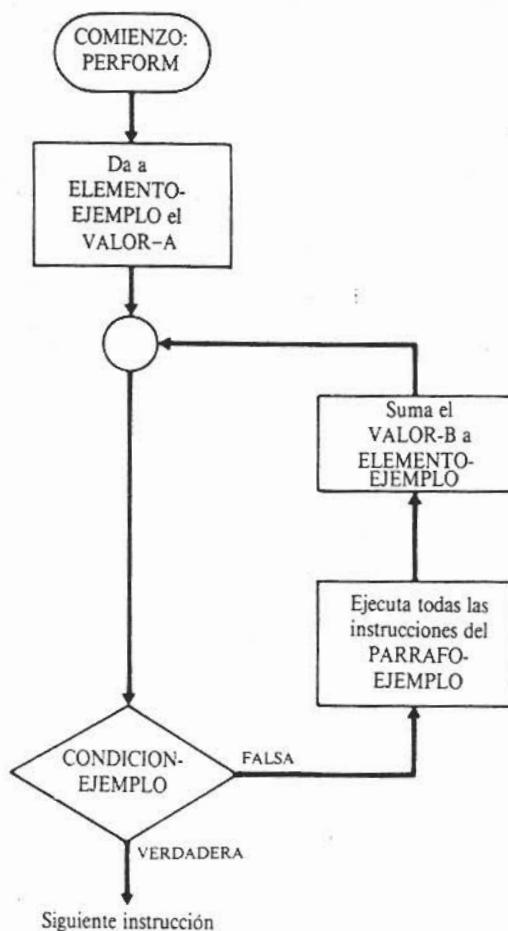


Fig. 7-13

**EJEMPLO 7.33** "PERFORM EJEMPLO-PARA 10 TIMES" se puede reescribir así:

```
PERFORM EJEMPLO-PARA
  VARYING NUMERO-DE-VECES FROM 1 BY 1
  UNTIL NUMERO-DE-VECES GREATER THAN 10
```

o bien

```
PERFORM EJEMPLO-PARA
  VARYING NUMERO-DE-VECES FROM 1 BY 1
  UNTIL NUMERO-DE-VECES EQUAL 11
```

Es muy importante comprender que

```
PERFORM EJEMPLO-PARA
  VARYING NUMERO-DE-VECES FROM 1 BY 1
  UNTIL NUMERO-DE-VECES IS EQUAL TO 10
```

hace que EJEMPLO-PARA se ejecute sólo *nueve* veces. La razón se ve fácilmente en la Figura 7-13. De modo similar,

```
PERFORM COMPARA-COD-LOCALIZACION
  VARYING NUMERO-LOCALIZACION FROM 1 BY 1
  UNTIL NUMERO-LOCALIZACION EQUAL TO MAX-LOCALIZACIONES
```

hace que se ejecute COMPARA-COD-LOCALIZACION un número de veces que es *una unidad inferior* al contenido de MAX-LOCALIZACIONES.

#### EJEMPLO 7.34

```
PERFORM COMPARA-CODIGOS
  VARYING NUMERO-COMPARACION FROM VALOR-A BY VALOR-B
  UNTIL NUMERO-COMPARACION GREATER THAN VALOR-C
```

Los cambios que se produzcan en NUMERO-COMPARACION, VALOR-B o VALOR-C dentro del párrafo COMPARA-CODIGOS tendrán el efecto esperado en el número de iteraciones. Sin embargo, una modificación en el valor de FROM (VALOR-A) durante la ejecución de PERFORM no afecta el número de iteraciones porque el valor de FROM sólo juega un papel al principio de la ejecución de PERFORM (donde inicializa el contenido del elemento de VARYING).

#### EJEMPLO 7.35

```
PERFORM EJEMPLO-PARA
  VARYING ITEM-A FROM ITEM-B BY ITEM-C
  UNTIL ITEM-A GREATER THAN ITEM-D
```

Los elementos utilizados con PERFORM...VARYING... deben ser numéricos. Así ITEM-A, ITEM-B, ITEM-C e ITEM-D deben disponer de formato PICTURE compuesto por 9 y opcionalmente S y V.

- PERFORM EJEMPLO-PARA
 

```
VARYING ITEM-A FROM .25 BY .03
      UNTIL ITEM-A GREATER THAN .40
```

EJEMPLO-PARA se ejecutará 6 veces, ITEM-A tomará los valores .25, .28, .31, .34, .37, .40. La ejecución de PERFORM concluirá cuando ITEM-A tenga el valor .43.

- PERFORM EJEMPLO-PARA
 

```
VARYING ITEM-A FROM .25 BY .03
      UNTIL ITEM-A EQUAL TO .41
```

En este caso, PERFORM produce un bucle infinito, puesto que la condición "ITEM-A EQUAL TO .41" no es nunca verdadera (.41-.25 no es divisible por .03). Véase el Ejemplo 7.31.

- PERFORM EJEMPLO-PARA
 

```
VARYING ITEM-A FROM 10 BY -1
      UNTIL ITEM-A IS ZERO
```

Con el valor de BY negativo, el ITEM-A tomará sucesivamente los valores 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 y EJEMPLO-PARA se ejecutará diez veces. Después de la décima ejecución, ITEM-A tomará el valor cero y la computadora pasará a la instrucción que sigue a PERFORM.

Cuando varían *dos* elementos en PERFORM...VARYING...,

```
PERFORM EJEMPLO-PARA
  VARYING ITEM-A FROM ITEM-B BY ITEM-C
  UNTIL ITEM-A GREATER THAN ITEM-D
  AFTER ITEM-E FROM ITEM-F BY ITEM-G
  UNTIL ITEM-E GREATER THAN ITEM-H
```

la ejecución se lleva a cabo como se indica en el diagrama de flujo de la Figura 7-14. Se observa que para cada valor de ITEM-A (en el rango ITEM-B ≤ ITEM-A ≤ ITEM-D), se ejecuta EJEM-

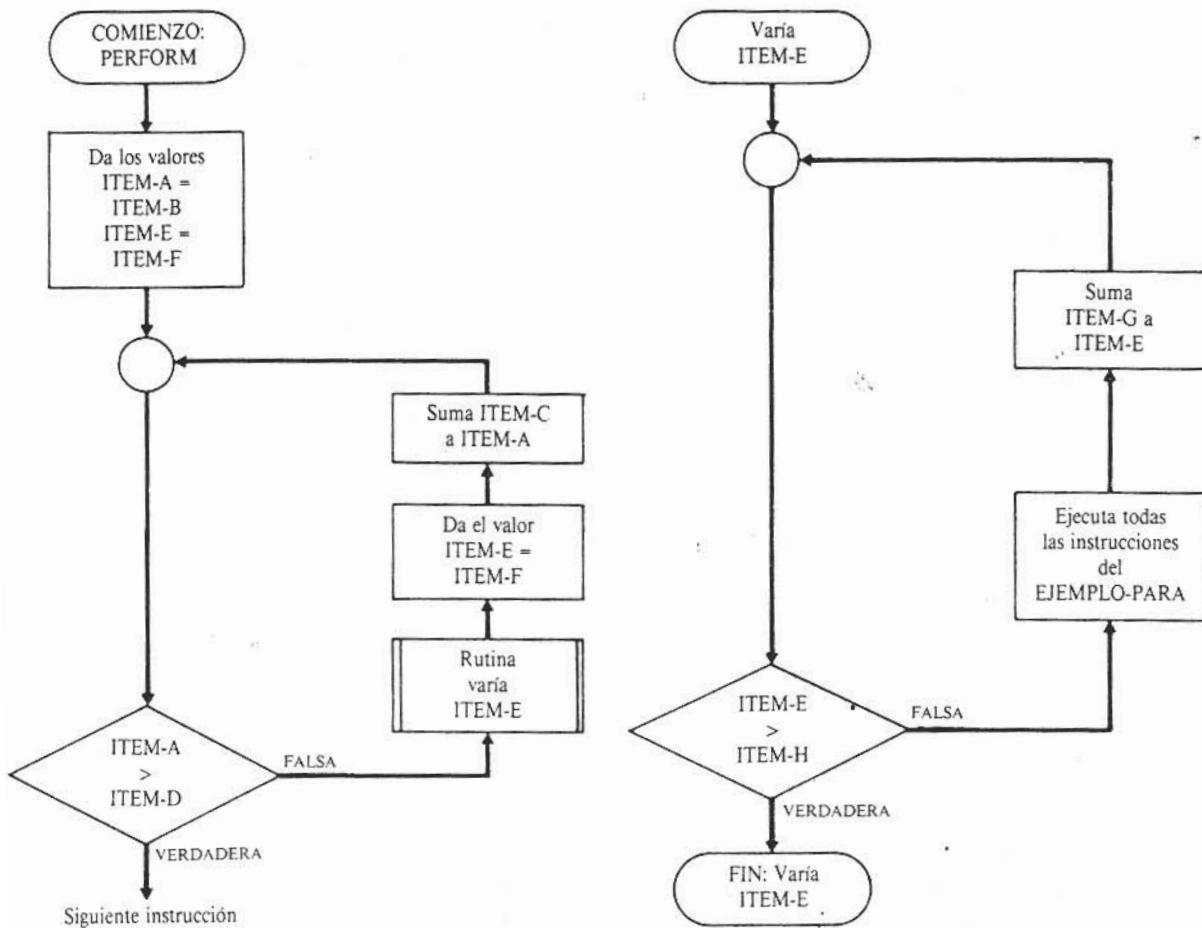


Fig. 7-14

PLO PARA para todos los valores de ITEM-E (en el rango  $ITEM-F \leq ITEM-E \leq ITEM-H$ ). De este modo, el efecto de `PERFORM... VARYING...` es ejecutar EJEMPLO-PARA:

$$\begin{aligned} \text{número total de iteraciones} \\ = (\text{número de valores en el rango de ITEM-A}) \\ \times (\text{número de valores en el rango de ITEM-E}) \end{aligned}$$

#### EJEMPLO 7.36

```

PERFORM CALCULO-PAGO-HIPOTECA
  VARYING PAGO-ANUAL FROM 83 BY 1
  UNTIL PAGO-ANUAL GREATER THAN 93
  AFTER PAGO-MENSUAL FROM 1 BY 1
  UNTIL PAGO-MENSUAL GREATER THAN 12
  
```

`CALCULO-PAGO-HIPOTECA` se ejecuta para cada valor de `PAGO-ANUAL` desde 83 a 93 ambos inclusive. Para cada uno de los 11 valores de `PAGO-ANUAL`, `PAGO-MENSUAL` variará de 1 a 12. La rutina calcula, por tanto,  $11 \times 12 = 132$  valores del pago de la hipoteca, que pueden representarse en forma tabular o lineal (Figura 7-15).

Mes \ Año	1	2	...	12
83	\$xxx			
84				
:				
93				

(a)

Año	Mes	Pago
83	1	\$xxx
83	2	
83	12	
84	1	
84	2	
84	12	
93	1	
93	2	
93	12	

(b)

Fig. 7-15

El número máximo de elementos que pueden variar en una instrucción PERFORM...VARYING... es tres (véase el Apéndice C para el COBOL de 1980):

```

PERFORM EJEMPLO-PARA
  VARYING A FROM B BY C
    UNTIL A GREATER THAN D
  AFTER E FROM F BY G
    UNTIL E GREATER THAN H
  AFTER I FROM J BY K
    UNTIL I GREATER THAN L

```

Para cada valor que toma A, E recorre su rango de valores completamente; y para cada valor que toma E, I recorre su rango de valores completo. Por consiguiente, el número total de iteraciones de EJEMPLO-PARA es

$$(\text{número de valores de A}) \times (\text{número de valores de E}) \times (\text{número de valores de I})$$

En la Figura 7-16 se da un diagrama de flujo estructurado de este caso.

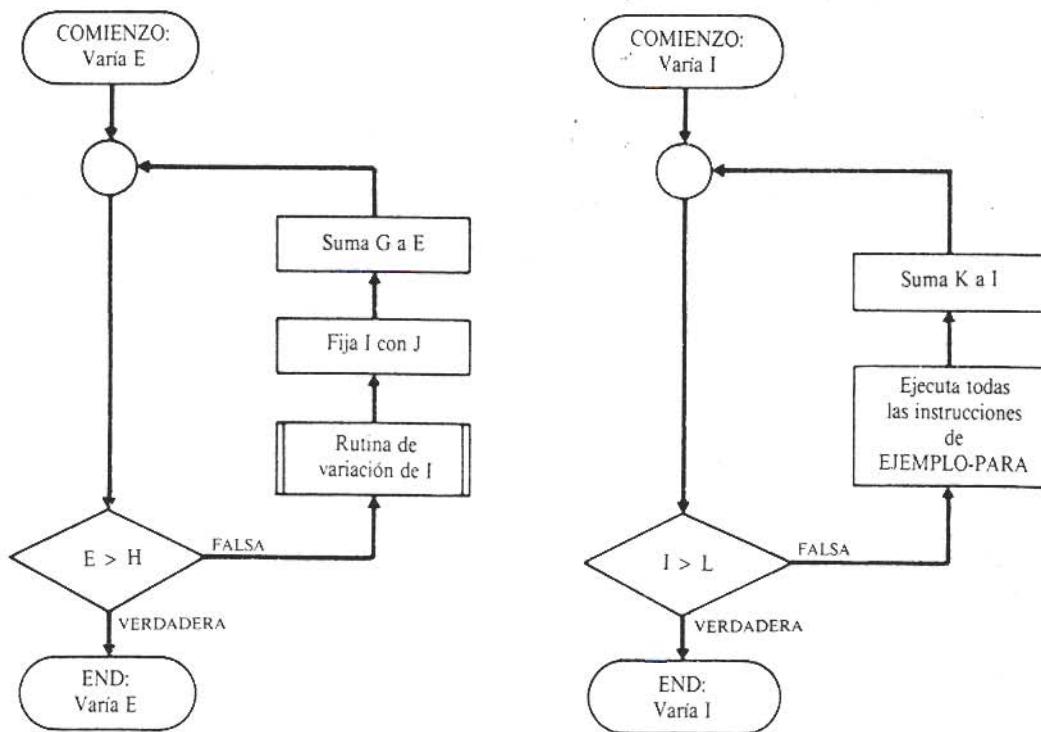
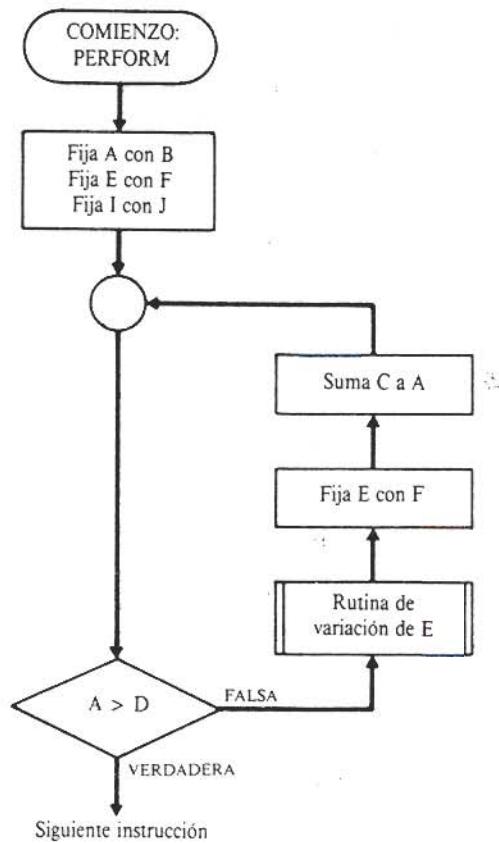
### EJEMPLO 7.37

```

PERFORM PROD-INTERES-TOTAL
  VARYING IMPORTE-HIPOTECA
    FROM 50000 BY 1000
    UNTIL IMPORTE-HIPOTECA GREATER THAN 100000
  AFTER PLAZO-REPAGO
    FROM 10 BY 1
    UNTIL PLAZO-REPAGO GREATER THAN 30
  AFTER TIPO-INTERES
    FROM .08 BY .01
    UNTIL TIPO-INTERES GREATER THAN .20

```

Esta instrucción PERFORM se puede utilizar para generar un informe con las cargas totales de intereses para varios importes de hipoteca a varios tipos de interés y con varios plazos. Los importes totales de la hipoteca varían de 50.000 a 100.000 de 1.000 en 1.000. Para cada valor de IMPORTE-HIPOTECA, PLAZO-REPAGO varía de 10 a 30 de uno en uno. Para cada valor de IMPORTE-HIPOTECA y PLAZO-REPAGO, TIPO-INTERES variará de 8 a 20% de uno en uno. El informe podría imprimirse en forma de lista lineal o quizás mejor en forma de 51 tablas, cada una con los intereses totales para todas las combinaciones de plazo y tipo de interés.



### 7.13 INSTRUCCIONES PERFORM ANIDADAS

Del mismo modo que puede haber instrucciones IF dentro de instrucciones IF (Sección 7.11), también es posible que haya instrucciones PERFORM dentro de un párrafo que se ejecuta a su vez con otra instrucción PERFORM.

#### EJEMPLO 7.38

```
PERFORM PARA-A
MOVE "SI" TO INTERRUPTOR-EJEMPLO
```

.....

PARA-A.

```
MOVE 1 TO L
MOVE 2 TO M
PERFORM PARA-B
MOVE 3 TO N
```

.....

PARA-B.

```
ADD 1 TO L
ADD 2 TO M
PERFORM PARA-C
VARYING UN-ELEMENTO FROM 1 BY 1
UNTIL UN-ELEMENTO GREATER THAN 3
ADD 3 TO P
```

.....

PARA-C.

```
MOVE SPACES TO X
MOVE LOW-VALUES TO Y
```

La ejecución de la instrucción PERFORM de más alto nivel hace que PARA-A se ejecute una vez. Sin embargo, PARA-A contiene una instrucción PERFORM, de forma que la ejecución de PARA-A implica la ejecución de PARA-B una vez. De forma análoga, PARA-B contiene una instrucción PERFORM que ejecuta el párrafo PARA-C *tres veces*. Recuerde que cuando se concluye la ejecución de una instrucción PERFORM, el control vuelve *automáticamente* a la instrucción que sigue a PERFORM. Así la computadora va desde la tercera iteración de la última instrucción PARA-C a la instrucción siguiente a PERFORM en el párrafo PARA-B; desde la última instrucción PARA-B hacia atrás hasta la instrucción que sigue a PERFORM en PARA-A; y desde la *última instrucción en PARA-A hacia atrás hasta la instrucción que sigue al PERFORM que inició la secuencia*. De este modo, las instrucciones se ejecutan en el siguiente orden:

1. PERFORM PARA-A
2. MOVE 1 TO L
3. MOVE 2 TO M
4. PERFORM PARA-B
5. ADD 1 TO L
6. ADD 2 TO M
7. PERFORM PARA-C VARYING...
8. MOVE SPACES TO X (1.<sup>a</sup> iteración; UN-ELEMENTO = 1)
9. MOVE LOW-VALUES TO Y (1.<sup>a</sup> iteración; UN-ELEMENTO = 1)
10. MOVE SPACES TO X (2.<sup>a</sup> iteración; UN-ELEMENTO = 2)
11. MOVE LOW-VALUES TO Y (2.<sup>a</sup> iteración; UN-ELEMENTO = 2)
12. MOVE SPACES TO X (3.<sup>a</sup> iteración; UN-ELEMENTO = 3)
13. MOVE LOW-VALUES TO Y (3.<sup>a</sup> iteración; UN-ELEMENTO = 3)
14. ADD 3 TO P
15. MOVE 3 TO N
16. MOVE "SI" TO INTERRUPTOR-EJEMPLO

La primera causa de errores al utilizar instrucciones PERFORM anidadas es escribir algún PERFORM que en algún punto se ejecute a sí mismo. Cuando una rutina *se llama a sí misma*, se dice que es *recursiva*. Las instrucciones PERFORM recursivas no están permitidas en COBOL (aunque la recursividad es posible en otros lenguajes de programación).

#### EJEMPLO 7.39

```

PROCEDURE DIVISION.
OPEN INPUT FICHERO-COMPROBACION
MOVE "NO" TO INTERRUPTOR-FIN-FICHERO
PERFORM LEE-Y-VALIDA
    UNTIL INTERRUPTOR-FIN-FICHERO EQUAL "SI"
CLOSE FICHERO-COMPROBACION
STOP RUN

LEE-Y-VALIDA.
READ FICHERO-COMPROBACION
    AT END
        MOVE "SI" TO INTERRUPTOR-FIN-FICHERO

IF INTERRUPTOR-FIN-FICHERO EQUAL "NO"
    IF COMP-CAMPO-REGISTRO NUMERIC
        PERFORM PROCESO-REG-VALIDO
    ELSE
        PERFORM PROD-MENSAJE-ERROR
        PERFORM LEE-Y-VALIDA

```

La última instrucción del párrafo LEE-Y-VALIDA es "PERFORM LEE-Y-VALIDA". Por tanto, bajo ciertas circunstancias (COMP-CAMPO-REGISTRO no es numérico), PERFORM se ejecutará *a sí mismo*. Esto constituye un error de programación, que, en el COBOL del sistema IBM OS/VS, daría lugar a un bucle infinito.

El programador evidentemente piensa que si se encuentra un registro inválido, lo apropiado es imprimir un mensaje de error y proseguir con la lectura y procesamiento del siguiente registro. La estrategia es correcta —lo que él o ella olvidó es que la rutina LEE-Y-VALIDA está *ya* bajo el control de una instrucción PERFORM...UNTIL... anterior. Se puede ejecutar una *segunda* instrucción PERFORM para el párrafo, *antes de que el primer PERFORM haya concluido su ejecución*. En este caso, PERFORM...UNTIL... hace que la instrucción PERFORM... del final sea innecesaria.

#### EJEMPLO 7.40

```

PROCEDURE DIVISION.

PARA-A.
    MOVE 1 TO L
    MOVE 2 TO M
    PERFORM PARA-B
    MOVE 3 TO N

PARA-B.
    ADD 1 TO L
    ADD 2 TO M
    PERFORM PARA-C
    ADD 3 TO P

PARA-C.
    MOVE SPACES TO X
    PERFORM PARA-A
    MOVE LOW-VALUES TO Y

```

En esta ocasión, la instrucción "PERFORM PARA-B" acaba ejecutándose a sí mismo, pero indirectamente después de una serie de PERFORM anidados. Este error se manifestaría como si se tratase de un bucle infinito. PARA-A contiene una instrucción PERFORM que ejecuta PARA-B (hasta el momento todo es correcto). PARA-B contiene a su vez una instrucción PERFORM que ejecuta PARA-C (todavía correcto). PARA-C, sin embargo, contiene una instrucción PERFORM que ejecuta PARA-A. *Esto es inválido* puesto que, durante la segunda ejecución de PARA-A, la instrucción "PERFORM PARA-B" se ejecutará *mientras que PARA-B está todavía siendo ejecutado*.

Para evitar situaciones como la del Ejemplo 7.40 debe observarse lo siguiente:

**Regla para el uso de instrucciones PERFORM anidadas.** Un párrafo o sección que esté siendo ejecutado bajo el control de una instrucción PERFORM no puede ejecutarse de nuevo mediante otra instrucción PERFORM hasta que el PERFORM de control concluya su ejecución.

Cuando retorne el control desde un procedimiento, se puede, por supuesto, volver a ejecutar mediante PERFORM el procedimiento de nuevo.

## 7.14 DISEÑO LOGICO: PSEUDOCODIGO

Frecuentemente se utiliza *pseudocódigo* en lugar de diagramas de flujo para diseñar y documentar la lógica de un programa. Como es muy similar al castellano, algunas veces se llama al pseudocódigo *castellano estructurado*. Como el pseudocódigo no está sujeto a estrictas reglas gramaticales, es mucho más fácil de producir y modificar que los diagramas de flujo estructurados (que deben dibujarse con sumo cuidado).

No existe una versión universal del pseudocódigo; lo único que se necesita es que su versión sea capaz de representar las tres estructuras lógicas fundamentales: secuencial, de selección e iterativa. La entrada, salida y procesamiento de datos se describen en frases condensadas en castellano y se utiliza el desplazamiento de las líneas para clarificar las relaciones entre las instrucciones en el pseudocódigo.

**EJEMPLO 7.41** Ejemplo de pseudocódigo para las estructuras lógicas básicas:

### *Secuencial*

```
incrementa contador de días excedidos en 1
calcula exceso coste como el 5% del saldo debido
construir la instrucción de impresión
```

### *Selección*

```
si horas trabajadas es mayor que 40
.....
en caso contrario
.....
fin-si
```

### *Iterativa*

```
ejecuta hasta que identificador-cliente-transacción no sea igual que el identificador del cliente en el
maestro
.....
fin-ejecuta
```

Caben otras posibilidades, por supuesto, siempre que sean (1) fáciles de leer y entender y (2) consistentes (es decir, si utiliza ejecuta fin-ejecuta para la estructura iterativa debe usarse siempre).

**EJEMPLO 7.42** Dé el pseudocódigo equivalente para (a) Figura 7-4, (b) Figura 7-7.

- (a) lee fecha de pago, importe, fecha de vencimiento, saldo anterior  
 si fecha de pago es mayor que fecha de vencimiento,  
     llama a la rutina de pago atrasado  
 en caso contrario  
     si importe es mayor que saldo,  
         calcula resto  
         formatea cheque  
         imprime cheque  
 en caso contrario  
     calcula saldo nuevo como saldo anterior-importe  
     fin-si  
 fin-si  
 construye línea de impresión

Observe que este pseudocódigo llama a un proceso predefinido en la instrucción "llama a la rutina de pago atrasado". También sería aceptable utilizar "ejecuta la rutina de pago atrasado" o cualquier otra frase que indique claramente lo que tiene que hacerse en ese instante.

- (b) da a fin-de-tarjetas el valor "no"  
 da a núm-empleados el valor cero  
 lee fichero-tarjetas y al final fija fin-de-tarjetas con "sí"  
 ejecuta hasta que fin-de-tarjetas sea igual a "sí"  
     incrementa el núm-empleados en 1  
     mueve nombre, horas normales y horas extras a las áreas de salida  
     cálculo de horas totales como horas normales + horas extras  
     imprime línea del listado con nombre, horas normales, horas  
         extras y horas totales  
     lee fichero-tarjetas y al final fija fin-de-tarjetas con "sí"  
 fin-ejecuta  
 imprime núm-empleados  
 parada

La versión en pseudocódigo es mucho más compacta que su diagrama de flujo equivalente. Observe que las instrucciones del pseudocódigo se repiten en una estructura iterativa comprendida entre "ejecuta hasta..." y "fin-ejecuta". En la traducción al COBOL, las instrucciones que tienen que repetirse deben escribirse como un párrafo aparte que se ejecutara mediante "PERFORM...UNTIL...". (Véase el Apéndice C para el COBOL de 1980.)

La mayor parte de los programadores prefieren el pseudocódigo a los diagramas de flujo estructurados. En cualquier caso, la lógica del programa antes de escribirlo en COBOL. La división de procedimientos debe diseñarse con pseudocódigo o diagramas de flujo.

Se puede ahorrar mucho tiempo de depuración comprobando el pseudocódigo o diagrama de flujo antes de codificar en COBOL. Esta comprobación consiste en suponer que uno mismo es la computadora y seguir el algoritmo paso a paso, procesando algunos datos y produciendo resultados. Si dichos resultados son correctos, el algoritmo debe ser correcto. Si se traduce correctamente a COBOL, el programa resultante también debe ser correcto.

## 7.15 DISEÑO LOGICO: TABLAS DE DECISION

Los diagramas de flujo y el pseudocódigo pueden llegar a ser en ocasiones complicados (por ejemplo, cuando se producen profundas anidaciones). Las estructuras de selección son necesarias en un algoritmo. Una *tabla de decisión* es un diagrama que sumariza *complicados procedimientos de decisión*; más exactamente, es la tabla de verdad (Sección 7.9) de un conjunto complicado de instrucciones lógicas (llamadas *acciones*) que se componen de instrucciones más simples (llamadas *condiciones*). En la Figura 7-17 se detalla la forma de una tabla de decisión.

<i>paquete de condiciones</i>	<i>condiciones</i>
<i>paquete de acciones</i>	<i>acciones</i>

Fig. 7-17

**EJEMPLO 7.43** La Figura 7-18 es una tabla de decisión relativa a una compra a crédito en un almacén. Hay tres condiciones que pueden ser verdaderas (sí = "Y") o falsas (no = "N"). Observe que todas las posibles combinaciones de veracidad o falsedad de las tres condiciones se representan en  $2^3 = 8$  columnas de condición. (Con  $n$  condiciones se precisarían  $2^n$  columnas.) Hay tres acciones posibles al efectuar una compra. (Es accidental que el número de acciones sea igual al número de condiciones.) En cada columna se pone "x" para indicar que la acción de la línea correspondiente será tomado en dicho supuesto. Véase el Problema 7.91 para conocer el pseudocódigo equivalente a esta tabla de decisión; la tabla de decisión es mucho más fácil de entender que el pseudocódigo.

Cuenta activa	Y	Y	Y	Y	N	N	N	N
Recibido pago del último mes	Y	Y	N	N	Y	Y	N	N
Compra dentro del límite	Y	N	Y	N	Y	N	Y	N
Compra aceptada	x	x	x					
Referencia a la Oficina de Créditos		x	x					
Compra no aceptada				x	x	x	x	x

Fig. 7-18

### Preguntas de repaso

- 7.1 ¿Qué se entiende por "lógica de un programa"?
- 7.2 ¿Cuáles son las tres estructuras lógicas básicas?
- 7.3 ¿Qué es un diagrama de flujo estructurado? ¿Cómo se utiliza?
- 7.4 ¿Cuál es el propósito de cada uno de los símbolos para los diagramas de flujo que se han visto en este capítulo?
- 7.5 Muestre cómo es el diagrama de flujo de la estructura secuencial.
- 7.6 Muestre cómo es el diagrama de flujo de la estructura de selección.
- 7.7 Muestre cómo es el diagrama de flujo de la estructura iterativa.
- 7.8 Muestre cómo pueden combinarse las tres estructuras lógicas básicas en un diagrama de flujo.

- 7.9 ¿Qué es la estructura "DO WHILE"?
- 7.10 ¿Cómo se implementa la estructura de selección en COBOL?
- 7.11 ¿Cómo puede utilizarse "NEXT SENTENCE"?
- 7.12 Explique la importancia del punto en la instrucción IF.
- 7.13 Explique la sintaxis y el uso de la condición de clase.
- 7.14 ¿Cuándo un dato es NUMERIC? ¿Y cuándo es ALPHABETIC?
- 7.15 ¿Qué elementos debe validar un programa COBOL?
- 7.16 ¿Qué sucede si se utilizan en un cálculo o condición de relación un dato numérico inválido?
- 7.17 ¿Cuándo *no* debe validarse la entrada de datos?
- 7.18 Explique la sintaxis y el uso de la condición de relación.
- 7.19 ¿Cómo son las relaciones "menor o igual que" y "mayor o igual que" en COBOL?
- 7.20 Explique las reglas por las cuales se comparan dos elementos numéricos en COBOL.
- 7.21 Explique cómo se comparan dos elementos no numéricos en COBOL.
- 7.22 ¿Qué combinaciones de datos son inválidas en las condiciones de relación?
- 7.23 Explique la sintaxis y el uso de la condición de signo.
- 7.24 Indique tres ventajas del uso de los nombres-de-condición en lugar de las condiciones de relación.
- 7.25 ¿Cómo se definen los nombres-de-condición?
- 7.26 ¿Cómo se evalúa una condición compuesta formada con los operadores lógicos AND, OR y NOT?
- 7.27 ¿Cuándo deben utilizarse paréntesis en las condiciones compuestas?
- 7.28 ¿Qué reglas se usan cuando no hay paréntesis en las condiciones compuestas?
- 7.29 Indique las reglas para las abreviaturas con condiciones compuestas. ¿Cuándo deben utilizarse abreviaturas? ¿Cuándo no deben utilizarse?
- 7.30 ¿Qué se entiende por IF anidados?
- 7.31 Explique la regla que debe seguirse en el desplazamiento de líneas al escribir instrucciones IF anidadas.
- 7.32 Explique cómo se emparejan IF y ELSE en los IF anidados.
- 7.33 Explique el uso de NEXT SENTENCE en las instrucciones IF anidadas.
- 7.34 Explique la estructura caso.
- 7.35 ¿Qué son las instrucciones IF lineales? Explique cómo se implementa la estructura caso.
- 7.36 Comente la sintaxis de las cuatro versiones de la instrucción PERFORM.
- 7.37 Explique la opción THRU de PERFORM. ¿Por qué debe evitarse?
- 7.38 ¿Cómo se utiliza PERFORM...UNTIL... para implementar la estructura iterativa?
- 7.39 ¿Qué sucede si la condición UNTIL llega a ser verdadera en medio de la ejecución con PERFORM de un párrafo?

- 7.40 ¿Qué es un bucle infinito?
- 7.41 ¿Es posible que un párrafo se ejecute cero veces con PERFORM? ¿Cómo?
- 7.42 Comente las diferencias entre PERFORM...VARYING..., PERFORM...VARYING...AFTER..., PERFORM...VARYING...AFTER...AFTER...
- 7.43 Comente las diferencias entre GREATER THAN y EQUAL con respecto al número de iteraciones llevadas a cabo por una instrucción PERFORM...UNTIL...
- 7.44 ¿Qué se entiende por PERFORM anidados?
- 7.45 Dé un "Thou Shalt Not" para instrucciones PERFORM anidadas.
- 7.46 ¿Qué es el "pseudocódigo" y por qué se prefiere a los diagramas de flujo?
- 7.47 Escriba el pseudocódigo y el diagrama de flujo estructurado de cada instrucción PERFORM del Problema 7.42.
- 7.48 ¿Qué es un "desk check"?
- 7.49 Explique el uso de tablas de decisión. ¿Cómo se escriben?

### Problemas resueltos

- 7.50 Escriba el contenido de la Figura 7-19 en COBOL.

```

IF EDAD IS GREATER THAN 12
    MOVE 4.00 TO TARIFA
ELSE
    MOVE 3.00 TO TARIFA

ADD 1 TO NUMERO-ADMITIDOS

```

Observe el punto estructurado al final de la instrucción IF... ELSE...

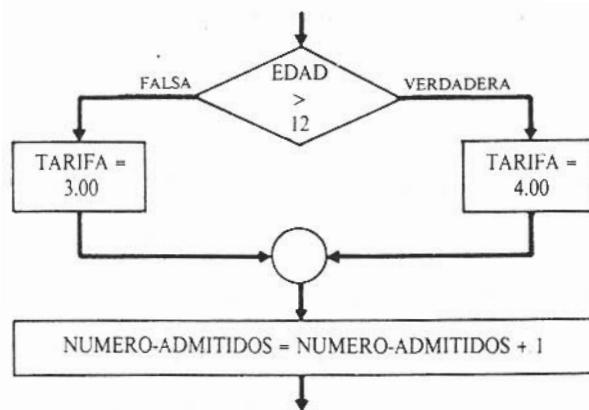


Fig. 7-19

7.51 Escriba el contenido de la Figura 7-20 en COBOL.

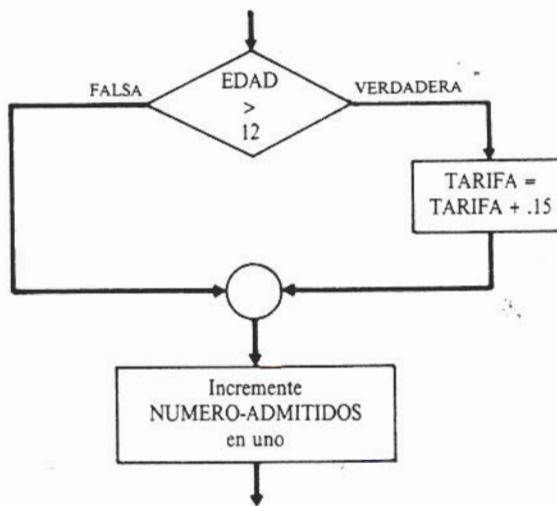


Fig. 7-20

La mayor parte de los programadores en COBOL escribirían:

```

IF EDAD GREATER THAN 12
  ADD .15 TO TARIFA
  
```

```

  ADD 1 TO NUMERO-ADMITIDOS
  
```

Algunos, sin embargo, preferirían que cada IF tuviera su ELSE:

```

IF EDAD GREATER THAN 12
  ADD .15 TO TARIFA
ELSE
  NEXT SENTENCE
  
```

```

  ADD 1 TO NUMERO-ADMITIDOS
  
```

7.52 Escriba en COBOL el contenido de la Figura 7-21.

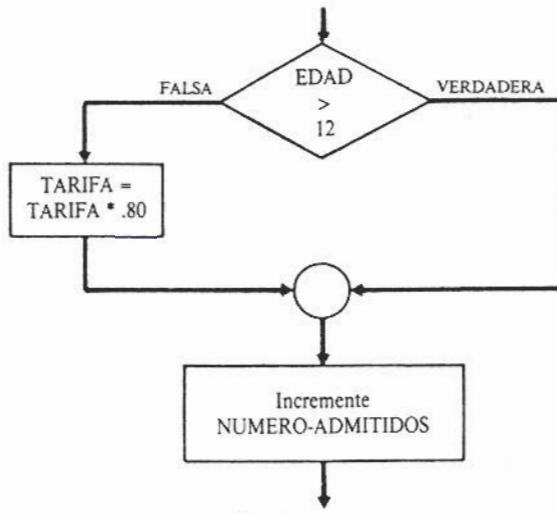


Fig. 7-21

Utilizando NEXT SENTENCE, la solución es muy directa:

```
IF EDAD GREATER THAN 12
    NEXT SENTENCE
ELSE
    COMPUTE TARIFA = TARIFA * .80

ADD 1 TO NUMERO-ADMITIDOS
```

Algunos programadores encuentran más elegante eliminar "NEXT SENTENCE" negando la condición de la Figura 7-21 e intercambiando TRUE y FALSE. Esto implicaría el siguiente programa:

```
IF EDAD NOT GREATER THAN 12
    COMPUTE TARIFA = TARIFA * .80

ADD 1 TO NUMERO-ADMITIDOS
```

- 7.53 ¿Qué está equivocado en lo siguiente?

```
IF TIPO-CLIENTE EQUAL "3"
    MOVE ZERO TO DESCUENTO-ENVIO
ELSE
    PERFORM DETERMINA-DESCUENTO-ENVIO
ADD DESCUENTO-ENVIO TO DESCUENTO-TOTAL
PERFORM FORMATEO-FACTURA
```

Si el desplazamiento indica el procesamiento deseado, se necesita un punto estructurado que marque el fin de la instrucción IF, como sigue:

```
IF TIPO-CLIENTE EQUAL "3"
    MOVE ZERO TO DESCUENTO-ENVIO
ELSE
    PERFORM DETERMINA-DESCUENTO-ENVIO

    ADD DESCUENTO-ENVIO TO DESCUENTO-TOTAL
    PERFORM FORMATEO-FACTURA
```

Sin el punto, las instrucciones ADD y PERFORM se considera que forman parte de la rutina ELSE (véase el Apéndice C en el COBOL de 1980).

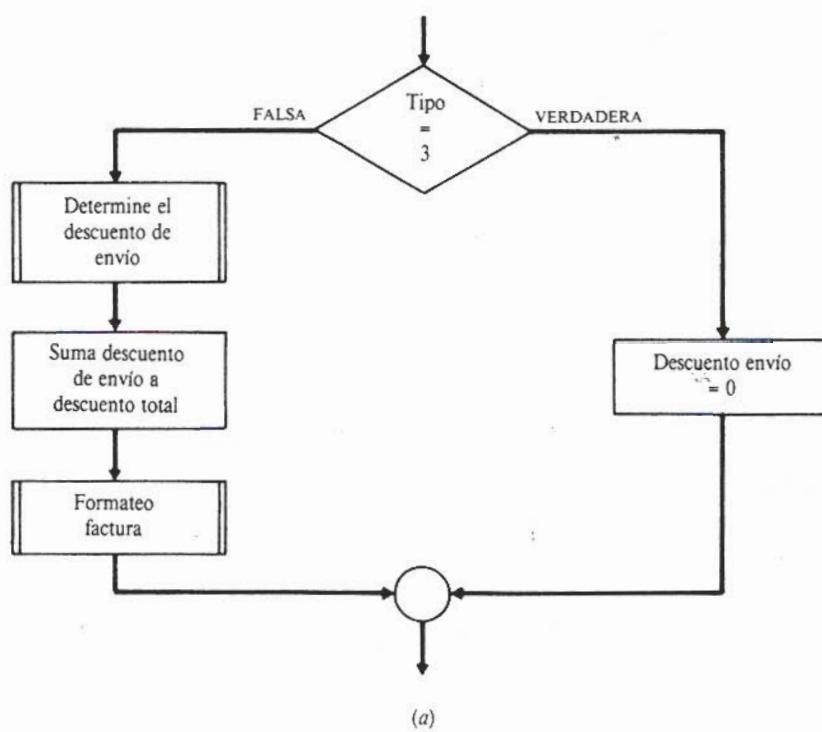
- 7.54 Dibuje el diagrama de flujo del segmento de programa del Problema 7.53 (a) sin y (b) con el punto.

Véase la Figura 7-22.

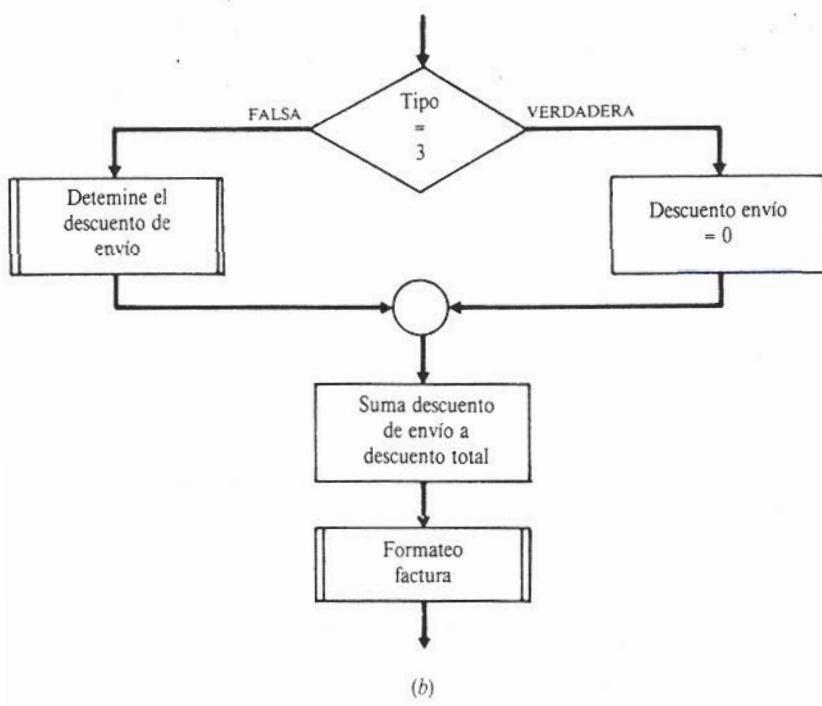
- 7.55 Escriba la siguiente instrucción IF sin utilizar NEXT SENTENCE.

```
IF NO-HAY-DESCUENTOS
    NEXT SENTENCE
ELSE
    ADD 1 TO NUMERO-CASOS-ESPECIALES
    PERFORM DETERMINA-DESCUENTO-ESPECIAL

IF NOT NO-HAY-DESCUENTOS
    ADD 1 TO NUMERO-CASOS-ESPECIALES
    PERFORM DETERMINA-DESCUENTO-ESPECIAL
```



(a)



(b)

Fig. 7-22

Para una solución mejor habría que definir un nombre de condición DESCUENTO-ESPECIAL:

05 INTER-DESCUENTOS-ESPECIALES	PIC X.
88 NO-HAY-DESCUENTOS	VALUE "A".
88 DESCUENTO-ESPECIAL	VALUE "B" THRU "G".

Ahora se puede eliminar "NOT" del siguiente modo:

```
IF DESCUENTO-ESPECIAL
    ADD 1 TO NUMERO-CASOS-ESPECIALES
    PERFORM DETERMINA-DESCUENTO-ESPECIAL
```

- 7.56** Escriba una rutina en COBOL que dé a INTER-REGISTRO-INVALIDO el valor "SI" si cualquiera de los campos NUMERO-DE-CUENTA, CODIGO-DE-TRANSACCION e IMPORTE fuese no numérico.

```
IF      NUMERO-DE-CUENTA NOT NUMERIC
OR      CODIGO-DE-TRANSACCION NOT NUMERIC
OR      IMPORTE NOT NUMERIC
        MOVE "SI" TO INTER-REGISTRO-INVALIDO
```

- 7.57** Corrija lo siguiente (Problema 7.56):

```
IF NUMERO-DE-CUENTA NOT NUMERIC
    PERFORM CUENTA-INVALIDA
ELSE IF CODIGO-TRANSACCION NOT NUMERIC
    PERFORM TRANSACCION-INVALIDA
ELSE IF IMPORTE NOT NUMERIC
    PERFORM IMPORTE-INVALIDO
```

El fallo aquí consiste en que el IF lineal no es apropiado para este tipo de validación. Cuando una condición es verdadera, se ignora el resto de la instrucción IF lineal. Obviamente no es esto lo que pretende, porque cuando se detecta un error se desea continuar con las comprobaciones para poder encontrar los otros errores que pudieran existir en el mismo registro. La manera correcta de hacer esto es con una serie de instrucciones IF separadas:

```
IF NUMERO-DE-CUENTA NOT NUMERIC
    PERFORM CUENTA-INVALIDA

IF CODIGO-DE-TRANSACCION NOT NUMERIC
    PERFORM TRANSACCION-INVALIDA

IF IMPORTE NOT NUMERIC
    PERFORM IMPORTE-INVALIDO
```

- 7.58** Dados los siguientes formatos PICTURE y contenidos, indique cuáles son NUMERIC, cuáles ALPHABETIC y cuáles ninguno de los dos:

	<i>PICTURE</i>	<i>Contenido</i>
(a)	S99	-82
(b)	XXX	"E T"
(c)	S99	82
(d)	99	82
(e)	99	+82
(f)	XXX	"C3B"
(g)	XXX	" "

(a) NUMERIC; (b) ALPHABETIC; (c) ni uno ni otro (cuando el formato PICTURE prevé un signo, el contenido siempre tiene dicho signo); (d) NUMERIC; (e) ni uno ni otro (cuando el formato PICTURE no prevé signo, el contenido no debe tener signo); (f) ni uno ni otro; (g) ALPHABETIC (puede haber espacios).

- 7.59 Indique si los siguientes datos deben validarse antes de su procesamiento: (a) el importe de un depósito bancario tecleado por el operador, (b) el nombre de un suscriptor a una revista que se teclea por consola, (c) el importe de una compra a crédito tecleado por el empleado del comercio, (d) el saldo anterior de un cliente en una tarjeta de crédito obtenido del fichero maestro de clientes almacenado en disco, (e) el saldo anterior de un cliente en una cuenta corriente obtenido del fichero maestro de clientes almacenado en disco, (f) la cantidad de un artículo ordenada por un cliente y tecleada a partir del documento de pedido original, (g) el precio del artículo ordenado en (f) leído en el fichero maestro del inventario almacenado en disco.

- (a) Este campo es importante y se introduce manualmente; por tanto, el primer programa que lo procese debe validar la entrada.
- (b) Este campo se introduce manualmente, pero su importancia es relativa. La revista probablemente llegará a su destino incluso aunque el nombre no sea totalmente correcto. Además, una comprobación de clase para datos alfabéticos sólo detectaría los caracteres numéricos, pero no las faltas ortográficas. Probablemente no convenga validar este campo.
- (c) Debe validarse la primera vez que se procesa, puesto que se introduce manualmente y es muy importante.
- (d) Presumiblemente todos los datos que figuran en el fichero maestro ya han sido validados y sería una pérdida de tiempo volver a hacerlo.
- (e) Igual que (d).
- (f) Deben validarse la primera vez que se procesan.
- (g) Igual que (d).

- 7.60 Señale cómo aparecerían las siguientes relaciones algebraicas en la instrucción IF de COBOL.

$$(a) a + b \leq 10 \quad (b) 5a \geq \frac{b}{3} \quad (c) a \neq b \quad (d) \frac{a}{b} \leq c \quad (e) a \geq b$$

- (a) IF A + B NOT GREATER THAN 10
- (b) IF 5 \* A NOT LESS THAN B / 3
- (c) IF A NOT EQUAL TO B
- (d) IF A / B NOT GREATER THAN C
- (e) IF A NOT LESS THAN B

- 7.61 Indique si el elemento A es menor que, igual o mayor que el elemento B:

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
A	CAD	+72	HI	-003	-4	+34.5	+34.00
B	COB	0072	HIGH	-3	+4	+34	+34

(a) A < B (orden alfabético); (b) A = B (comparación algebraica); (c) A < B (HI = HIGH); (d) A = B (comparación algebraica); (e) A < B (comparación algebraica); (f) A > B (comparación algebraica); (g) A = B (comparación algebraica).

- 7.62 Vuelva a escribir las siguientes condiciones utilizando condiciones de signo en lugar de condiciones relación; (a) IF IMPORTE-DEBIDO IS GREATER THAN ZERO...; (b) PERFORM ENVIO-LIBRO-MAYOR UNTIL CUENTA-MAYOR IS EQUAL TO ZERO; (c) IF EXISTENCIAS IS LESS THAN ZERO...

(a) IF IMPORTE-DEBIDO IS POSITIVE...; (b) PERFORM ENVIO-LIBRO-MAYOR UNTIL CUENTA-MAYOR IS ZERO; (c) IF EXISTENCIAS NEGATIVE... Observe la omisión de la palabra opcional "IS" en (c).

- 7.63 Vuelva a escribir las siguientes definiciones de datos e instrucciones IF... PERFORM... utilizando nombres de condición en lugar de condiciones relación:

(a) 05 ID-CLIENTE PIC X (5).

```
IF ID-CLIENTE NOT LESS THAN ZERO AND NOT GREATER
THAN "1111"
PERFORM PROCESO-CLIENTE-PREFERENTE
```

(b) 05 CODIGO-DE-TRANSACCION PIC X.

```
IF CODIGO-DE-TRANSACCION EQUAL "1"
    PERFORM PROCESO-DEPOSITO
ELSE IF CODIGO-DE-TRANSACCION EQUAL "2"
    PERFORM PROCESO-REINTEGRO
```

(c) 05 CODIGO-REGION PIC X.

```
IF CODIGO-REGION EQUAL "A" OR "B" OR "C"
    PERFORM ENVIO-AL-NORESTE
ELSE IF CODIGO-REGION GREATER THAN "C"
    AND LESS THAN "L"
    PERFORM ENVIO-AL-SURESTE
ELSE IF CODIGO-REGION EQUAL "M" OR "P"
    PERFORM ENVIO-AL-OESTE
```

(d) 05 AÑOS-EN-EL-COLEGIO PIC 99.

```
IF AÑOS-EN-EL-COLEGIO EQUAL 6
    PERFORM PROCESO-SEXTO-GRADO
ELSE IF AÑOS-EN-EL-COLEGIO NOT LESS THAN 9 AND NOT
    GREATER THAN 12
    PERFORM PROCESO-BACHILLERATO
```

(e) 05 TIPO-DEPRECIACION PIC 9.

```
IF TIPO-DEPRECIACION EQUAL 1 OR 2 OR 3
    PERFORM LINEAL-DECRIENTE
ELSE IF TIPO-DEPRECIACION EQUAL 4
    PERFORM PROGRESIVA-DOBLE
```

(a) 05 ID-CLIENTE PIC X(5).
 88 CLIENTE-PREFERENTE VALUE ZERO THRU "1111".

```
IF CLIENTE-PREFERENTE
    PERFORM PROCESO-CLIENTE-PREFERENTE
```

(b) 05 CODIGO-DE-TRANSACCION                   PIC X.  
       88 DEPOSITO                                  VALUE "1".  
       88 REINTEGRO                                  VALUE "2".

      IF DEPOSITO  
         PERFORM PROCESO-DEPOSITO  
       ELSE IF REINTEGRO  
         PERFORM PROCESO-REINTEGRO

(c) 05 CODIGO-REGION                           PIC X.  
       88 REGION NORESTE                          VALUE "A" THRU "C".  
       88 REGION SURESTE                          VALUE "D" THRU "K".  
       88 REGION OESTE                              VALUE "M" "P".

      IF REGION-NORESTE  
         PERFORM ENVIO-AL-NORESTE  
       ELSE IF REGION-SURESTE  
         PERFORM ENVIO-AL-SURESTE  
       ELSE IF REGION-OESTE  
         PERFORM ENVIO-AL-OESTE

(d) 05 AÑOS-EN-EL-COLEGIO                   PIC 99.  
       88 SEXTO-GRADO                              VALUE 6.  
       88 BACHILLERATO                             VALUE 9 THRU 12.

      IF SEXTO GRADO  
         PERFORM PROCESO-SEXTO-GRADO  
       ELSE IF BACHILLERATO  
         PERFORM PROCESO-BACHILLERATO

(e) 05 TIPO-DEPRECIACION                   PIC 9.  
       88 LINEAL                                    VALUE 1 THRU 3.  
       88 PROGRESIVA                                VALUE 4.

      IF LINEAL  
         PERFORM LINEAL-DECRECIENTE  
       ELSE IF PROGRESIVA  
         PERFORM PROGRESIVA-DOBLE

**7.64** Corrija las siguientes instrucciones en las que se hace mal uso de los nombres de condición del Problema 7.63.

(a) IF CLIENTE-PREFERENTE EQUAL "00111"

(b) IF DEPOSITO  
      MOVE SPACES TO REINTEGRO

(c) IF BACHILLERATO EQUAL 10

(a) IF ID-CLIENTE EQUAL "00111"

Un nombre de condición no puede utilizarse en lugar de un elemento de datos en el programa.  
 Un nombre de condición es sólo una abreviatura de una condición relación.

(b) IF DEPOSITO  
      MOVE SPACES TO CODIGO-DE-TRANSACCION

Al igual que en (a), en la instrucción MOVE debe usarse un elemento de datos y no un nombre de condición.

(c) IF AÑOS-EN-EL-COLEGIO EQUAL 10

No hay nombre de condición para la relación AÑOS-EN-EL-COLEGIO EQUAL 10; por tanto, la condición debe escribirse en la instrucción IF. También podría establecerse un nombre de condición así:

```
05 AÑOS-EN-EL-COLEGIO PIC 99.  
88 DECIMO-GRADO VALUE 10.
```

IF DECIMO-GRADO

- 7.65 Hay una regla en COBOL que establece que la cláusula VALUE no puede utilizarse en la FILE SECTION. ¿Se aplica esta regla también a los nombres de condición?

No, la regla se aplica sólo a los *elementos de datos* en la FILE SECTION. Se puede (se debe) utilizar en la FILE SECTION para definir nombres de condición (nivel 88).

- 7.66 Sean A, B, C y D nombres de condición. Supongamos que A y B son verdaderas y que C y D son falsas. Indique si las siguientes condiciones compuestas son verdaderas o falsas:

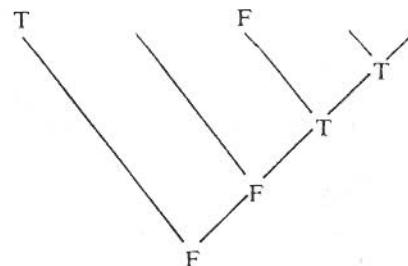
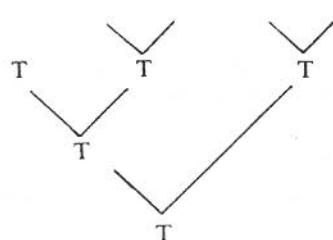
- (a) A AND B OR A AND C
- (c) A AND NOT C OR D
- (e) A AND NOT C OR NOT D
- (g) A AND NOT (C OR NOT D)
- (i) A AND B AND C
- (k) A OR C AND B OR D
- (m) A AND C OR B OR D

- (b) A AND (B OR A) AND C
- (d) A AND NOT (C OR D)
- (f) (A AND NOT C) OR NOT D
- (h) A AND C OR B AND D
- (j) A OR B OR C OR D
- (l) (A OR C) AND (B OR D)

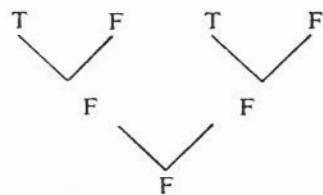
- (a) En ausencia de paréntesis, primero se ejecuta AND (de izquierda a derecha) seguida de OR. En este caso la expresión se evalúa como si fuera: (A AND B) OR (A AND C). (A AND B) es verdadera; por tanto, (A AND B) OR... es verdadera.
- (b) La expresión entre paréntesis se evalúa primero: (B OR A) es verdadera. Despues se llevan a cabo AND de izquierda a derecha: A AND (B OR A) es verdadera; luego se conjunta mediante AND este resultado con C y el resultado es falso.
- (c) En primer lugar se evalúa NOT C cuyo resultado es verdadero. A AND NOT C da como resultado verdadero. Al concatenar mediante OR el resultado anterior con D se obtiene verdadero.
- (d) (C OR D) es falsa; por tanto, NOT (C OR D) es verdadera. Al conjuntar esto con A mediante AND el resultado es verdadero.
- (e) NOT se ejecuta primero, de izquierda a derecha: NOT C es verdadera; NOT D es verdadera. A AND NOT C es verdadera. Este último resultado se concatena mediante OR con NOT D y el resultado es verdadero.

(f) (A AND NOT C) OR NOT D

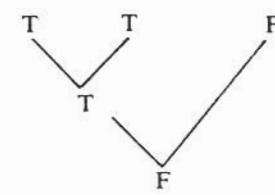
(g) A AND NOT (C OR NOT D)



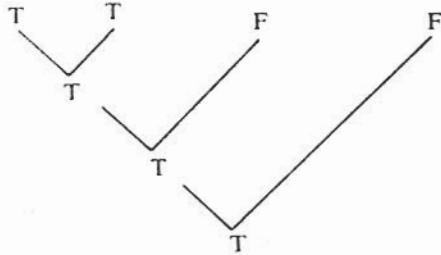
(h) A AND C OR B AND D



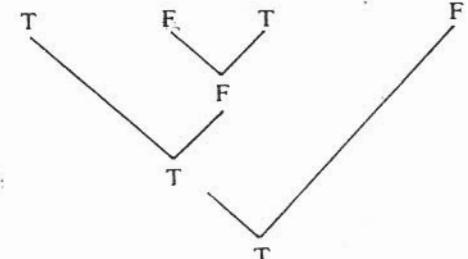
(i) A AND B AND C



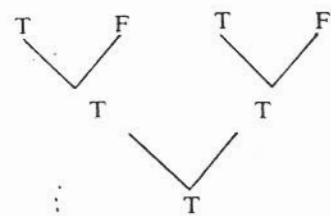
(j) A OR B OR C OR D



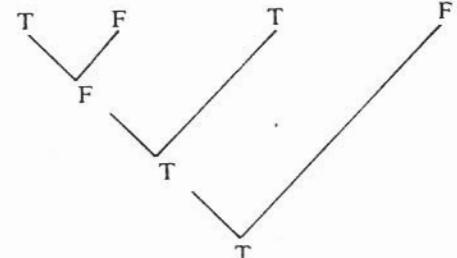
(k) A OR C AND B OR D



(l) (A OR C) AND (B OR D)



(m) A AND C OR B OR D



7.67 Sean A, B, C y D nombres de condición. Vuelva a escribir las siguientes condiciones compuestas con paréntesis que indiquen el orden de ejecución de las operaciones:

(a) NOT A OR B AND NOT C  
(c) A AND C OR B(b) NOT A OR NOT B AND NOT C  
(d) A OR B AND C(a) (NOT A) OR (B AND (NOT C))  
(c) (A AND C) OR B(b) (NOT A) OR ((NOT B) AND (NOT C))  
(d) A OR (B AND C)

7.68 Escriba una condición que sea verdadera si y sólo si ELEMENTO-A está comprendido entre 3 y 27 ambos inclusive.

La solución más simple es:

IF ELEMENTO-A NOT LESS THAN 3 AND NOT GREATER THAN 27

7.69 Escriba una condición que sea verdadera si y sólo si ELEMENTO-A no está comprendido entre 3 y 27 ambos inclusive.

IF ELEMENTO-A LESS THAN 3 OR GREATER THAN 27

Observe que se utiliza OR en lugar de AND: tan sólo con que *una* de las condiciones simples relativas a ELEMENTO-A sea verdadera el resultado será verdadero.

- 7.70 Escriba una condición que sea verdadera si y sólo si SALDO-ANTERIOR-PAGO y FECHA-DE-PAGO son numéricos.

```
IF      SALDO-ANTERIOR NOT NUMERIC
OR     PAGO NOT NUMERIC
OR     FECHA-PAGO NOT NUMERIC
```

- 7.71 Escriba una condición que sea verdadera si y sólo si A es 1, B es 2, C es 3 y D no es 4.

```
IF      A EQUAL 1
AND    B EQUAL 2
AND    C EQUAL 3
AND    D NOT EQUAL 4
```

- 7.72 Escriba una condición que sea verdadera si y sólo si END-OF-FILE es falsa y o bien SALDO es negativo o bien PAGO es cero.

```
IF      NOT FIN-DE-FICHERO
AND    (SALDO IS NEGATIVE OR PAGO IS ZERO)
```

Se precisa el paréntesis porque A AND B OR C se evaluaría como (A AND B) OR C.

- 7.73 Escriba una condición que sea verdadera si y sólo si AÑO está comprendido entre 82 y 85, MES entre 6 y 8 y DIA entre 5 y 20.

```
IF      (AÑO NOT LESS THAN 82 AND NOT GREATER THAN 85)
AND    (MES NOT LESS THAN 6 AND NOT GREATER THAN 8)
AND    (DIA NOT LESS THAN 5 AND NOT GREATER THAN 20)
```

Como sólo se necesita el operador lógico AND, los paréntesis son opcionales (aunque clarifican).

- 7.74 Abrevie en lo posible las siguientes condiciones compuestas:

- (a) A NOT EQUAL 1 AND A NOT EQUAL 2 AND A NOT EQUAL 3
- (b) A EQUAL 7 OR A EQUAL 8 AND B EQUAL 8
- (c) A EQUAL 1 OR A EQUAL 2 AND A EQUAL 3
- (d) A EQUAL 1 OR A EQUAL 2 OR A LESS THAN 9 OR B LESS THAN 10
- (e) A EQUAL 1 OR A LESS THAN 5 OR A GREATER THAN 8
- (f) A EQUAL B AND A LESS THAN C AND A LESS THAN D
  
- (a) A NOT EQUAL 1 AND 2 AND 3
- (b) A EQUAL 7 OR 8 AND B EQUAL 8
- (c) A EQUAL 1 OR 2 AND 3 (que puede reducirse lógicamente a A EQUAL 1)
- (d) A EQUAL 1 OR 2 OR LESS THAN 9 OR B LESS THAN 10
- (e) A EQUAL 1 OR LESS THAN 5 OR GREATER THAN 8
- (f) A EQUAL B AND LESS THAN C AND D

7.75 Escriba las siguientes condiciones compuestas de forma no abreviada:

- (a) A EQUAL B OR C AND LESS THAN D
  - (b) A GREATER THAN B OR D AND C OR LESS THAN B OR D
  - (c) A LESS THAN B AND D AND C
  - (d) A EQUAL B OR D AND C
  - (e) A EQUAL B OR C AND LESS THAN D OR GREATER THAN E
  - (f) A NOT EQUAL B OR D AND NOT LESS THAN C AND E
- 
- (a) A EQUAL B OR A EQUAL C AND A LESS THAN D
  - (b) A GREATER THAN B OR A GREATER THAN D AND A GREATER THAN C  
OR A LESS THAN B OR A LESS THAN D
  - (c) A LEES THAN B AND A LESS THAN D AND A LESS THAN C
  - (d) A EQUAL B OR A EQUAL D AND A EQUAL C
  - (e) A EQUAL B OR A EQUAL C AND A LESS THAN D OR A GREATER THAN E
  - (f) A NOT EQUAL B OR A NOT EQUAL D AND A NOT LESS THAN C AND A NOT  
LESS THAN E

7.76 Revise la separación entre líneas para que de forma correcta se ponga de manifiesto la jerarquía de las instrucciones IF anidadas:

- (a) IF NOT FIN-DE-FICHERO  
    IF LINEA-CREDITO-OK  
        PERFORM DETERMINA-LIMITE-CREDITO  
        PERFORM IMPRESION-TERMINOS  
    ELSE  
        PERFORM IMPRESION-TOTALES-FINAL
  
- (b) IF PAGO NOT NUMERIC  
        PERFORM PAGO-INVALIDO  
        IF FECHA2 NOT NUMERIC  
            PERFORM FECHA2-INVALIDA  
    ELSE  
        PERFORM APPLICACION-PAGO  
        IF SALDO NEGATIVE  
            PERFORM INGRESE-FONDOS  
    ELSE  
        PERFORM FICHERO-SALDO
  
- (c) IF A EQUAL B  
        MOVE 1 TO A  
        MOVE 2 TO B.  
        MOVE 3 TO C  
        IF B EQUAL C  
            MOVE 4 TO D  
            IF C EQUAL D  
                MOVE 5 TO E  
        ELSE  
            MOVE 6 TO F.

- (a) IF NOT FIN-DE-FICHERO  
     IF LINEA-CREDITO-OK  
         PERFORM DETERMINA-LIMITE-CREDITO  
         PERFORM IMPRESION-TERMINOS  
     ELSE  
         PERFORM IMPRESION-TOTALES-FINAL
- (b) IF PAGO NOT NUMERIC  
     PERFORM PAGO-INVALIDO  
     IF FECHA2 NOT NUMERIC  
         PERFORM FECHA2-INVALIDA  
     ELSE  
         PERFORM APPLICACION-PAGO  
         IF SALDO NEGATIVE  
             PERFORM INGRESE-FONDOS  
         ELSE  
             PERFORM FICHERO-SALDOS

En este caso como en (a), cada cláusula ELSE se empareja con el IF precedente que no haya sido emparejado.

- (c) IF A EQUAL B  
     MOVE 1 TO A  
     MOVE 2 TO B
- MOVE 3 TO C  
   IF B EQUAL C  
     MOVE 4 TO D  
     IF C EQUAL D  
       MOVE 5 TO E  
     ELSE  
       MOVE 6 TO F

Observe la importancia del punto para cambiar el significado de los IF anidados. Sin el punto después de "MOVE 2 TO B", la secuencia de ejecución hubiera sido la siguiente:

```
IF A EQUAL B
  MOVE 1 TO A
  MOVE 2 TO B
  MOVE 3 TO C
  IF B EQUAL C
    MOVE 4 TO D
    IF C EQUAL D
      MOVE 5 TO E
    ELSE
      MOVE 6 TO F
```

- 7.77 Vuelva a escribir lo siguiente sin utilizar NEXT SENTENCE y también corrija la separación entre líneas:

```
IF A EQUAL B
  NEXT SENTENCE
ELSE
  PERFORM PARA-1
  PERFORM PARA-2.
  PERFORM PARA-3
```

```

IF A NOT EQUAL B
  PERFORM PARA-1
  PERFORM PARA-2

  PERFORM PARA-3

```

- 7.78 Cree una estructura caso en un párrafo utilizando un IF lineal:

<i>Valor CODIGO-REGION</i>	<i>Acción</i>
1	PERFORM ENVIO-NORESTE
2	PERFORM ENVIO-SURESTE
3	PERFORM ENVIO-NORESTE
4	PERFORM ENVIO-OESTE

DETERMINA-ENVIO.

```

IF CODIGO-REGION EQUAL 1
  PERFORM ENVIO-NORESTE
ELSE IF CODIGO-REGION EQUAL 2
  PERFORM ENVIO-SURESTE
ELSE IF CODIGO-REGION EQUAL 3
  PERFORM ENVIO-NORESTE
ELSE IF CODIGO-REGION EQUAL 4
  PERFORM ENVIO-OESTE
ELSE
  PERFORM CODIGO-INVALIDO

```

o combinando los casos,

DETERMINA-ENVIO.

```

IF CODIGO-REGION EQUAL 1 OR 3
  PERFORM ENVIO-NORESTE
ELSE IF...

```

- 7.79 Muestre cómo ejecutar la estructura caso en el Problema 7.78.

```
PERFORM DETERMINA-ENVIO
```

- 7.80 Vuelva a escribir lo siguiente utilizando PERFORM...UNTIL...:

```

PERFORM CAPTURA-ENTRADA
PERFORM CAPTURA-ENTRADA
PERFORM CAPTURA-ENTRADA
PERFORM CAPTURA-ENTRADA

PERFORM CAPTURA-ENTRADA
  VARYING CONTADOR-VECES
    FROM 1 BY 1
  UNTIL CONTADOR-VECES GREATER THAN 4

```

- 7.81 ¿Cuántas veces se ejecuta CAPTURA-ENTRADA?

```
PERFORM CAPTURA-ENTRADA
  VARYING CONTADOR-VECES FROM 1 BY 1
  UNTIL CONTADOR-VECES EQUAL 10
```

Nueve veces.

7.82 ¿Cuántas veces se ejecuta PARA-A?

```
MOVE 10 TO CONTADOR-VECES
PERFORM PARA-A CONTADOR-VECES TIMES
```

.....  
PARA-A.

```
MOVE 5 TO CONTADOR-VECES
MOVE P TO Q
```

PARA-A se ejecuta diez veces. Cambiando el contenido de CONTADOR-VECES después de PERFORM...TIMES no se altera el número de iteraciones.

7.83 ¿Cuántas veces se ejecuta PARA-A?

```
PERFORM PARA-A
VARYING CONTADOR-VECES FROM 1 BY 1
UNTIL CONTADOR-VECES GREATER THAN 10
```

.....  
PARA-A.

```
MOVE P TO Q
MOVE 11 TO CONTADOR-VECES
```

PARA-A se ejecuta una vez. Al utilizar PERFORM...VARYING...UNTIL..., cualquier modificación de los datos que siguen a VARYING, BY y UNTIL no afecta al número de iteraciones.

7.84 ¿Cuántas veces se ejecuta PARA-S?

```
MOVE 11 TO CONTADOR-VECES
PERFORM PARA-A
UNTIL CONTADOR-VECES GREATER THAN 10
```

PARA-A no se ejecuta; *antes* de ejecutar PERFORM se evalúa la condición de UNTIL.

7.85 ¿Cuántas veces se ejecuta PARA-S?

```
MOVE ZERO TO CONTADOR-VECES
PERFORM PARA-S
UNTIL CONTADOR-VECES GREATER THAN 10
```

.....  
PARA-S.

```
ADD 1 TO CONTADOR-VECES
DISPLAY CONTADOR-VECES
```

Once veces: CONTADOR-VECES irá teniendo los valores 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

7.86 Dé los valores de ELEMENTO en cada ejecución de PARA-A:

- (a)      PERFORM PARA-A VARYING ELEMENTO FROM 1 BY 2  
              UNTIL ELEMENTO GREATER THAN 8
- (b)      PERFORM PARA-A  
              VARYING ELEMENTO FROM 10 BY -3  
              UNTIL ELEMENTO LESS THAN -4

- (c) PERFORM PARA-A  
     VARYING ELEMENTO FROM 2.7 BY -.4  
     UNTIL ELEMENTO NEGATIVE
- (d) PERFORM PARA-A  
     VARYING ELEMENTO FROM 6 BY 2  
     UNTIL ELEMENTO EQUAL 11
- (e) PERFORM PARA-A  
     VARYING ELEMENTO FROM 5 BY 5  
     UNTIL ELEMENTO GREATER THAN 25
- (a) ELEMENTO = 1, 3, 5, 7; la ejecución se detiene con ELEMENTO = 9.
- (b) ELEMENTO = 10, 7, 4, 1, -2; la ejecución se detiene con ELEMENTO = -5.
- (c) ELEMENTO = 2.7, 2.3, 1.9, 1.5, 1.1, .7, .3; la ejecución se detiene con ELEMENTO = -.1.
- (d) ELEMENTO = 6, 8, 10, 12, 14, 16,..., hasta que lo permita el sistema operativo, pues se trata de un bucle infinito.
- (e) ELEMENTO = 5, 10, 15, 20, 25; la ejecución se detiene con ELEMENTO = 30.

7.87 Dé los valores de A y B para los cuales se ejecuta PARA.

```

PERFORM PARA
    VARYING A FROM 2 BY 2
        UNTIL A GREATER THAN 8
    AFTER B FROM 5.5 BY -.5
        UNTIL B LESS THAN 4.0
  
```

A	2	2	2	2	4	4	4	4	6	6	6	6	8	8	8	8
B	5.5	5.0	4.5	4.0	5.5	5.0	4.5	4.0	5.5	5.0	4.5	4.0	5.5	5.0	4.5	4.0

La ejecución de la instrucción PERFORM finaliza con B = 5.5 y A = 10.

7.88 Dé los valores de A, B y C para los cuales se ejecuta PARA.

```

PERFORM PARA
    VARYING A FROM 2 BY 3
        UNTIL A GREATER THAN 7
    AFTER B FROM 5 BY -1
        UNTIL B LESS THAN 3
    AFTER C FROM 1 BY 1
        UNTIL C GREATER THAN 3
  
```

A	2	2	2	2	2	2	2	2	5	5	5	5	5	5	5	5
B	5	5	5	4	4	4	3	3	3	5	5	5	4	4	4	3
C	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1

La ejecución de la instrucción PERFORM finaliza con A = 8, B = 5 y C = 1.

7.89 Explique lo que está equivocado en las siguientes rutinas:

(a) PERFORM PARA-A  
 VARYING ELEMENTO-A FROM 1 BY 1 UNTIL ELEMENTO-A GREATER  
 THAN 10  
 VARYING ELEMENTO-B FROM 5 BY 2 UNTIL ELEMENTO-B GREATER  
 THAN 20

(b) PERFORM PARA-A THRU PARA-C  
 .....  
 PARA-C.  
 .....  
 PARA-A.  
 .....

(c) PERFORM PRODUCE-INFORME  
 UNTIL END-OF-FILE  
 .....  
 PRODUCE-INFORME  
 IF CAMPO-ENTRADA NUMERIC  
 PERFORM PROCESALO  
 ELSE  
 PERFORM CAMPO-INVALIDO  
 .....  
 CAMPO-INVALIDO.  
 PERFORM ESCRIBE-MENSAJE-ERROR  
 PERFORM LEE-SIGUIENTE-REGISTRO  
 PERFORM PRODUCE-INFORME

- (a) Error sintáctico: la instrucción PERFORM con varios elementos debe escribirse PERFORM... VARYING...AFTER...AFTER...
- (b) Error sintáctico: el párrafo THRU (o SECTION) debe *seguir físicamente* al primer párrafo (o SECTION) en el rango.
- (c) Error lógico: una instrucción PERFORM recursiva. La instrucción PERFORM inicial ejecuta PRODUCE-INFORME. Dependiendo de la entrada, PRODUCE-INFORME puede ejecutar CAMPO-INVALIDO, después ejecuta PRODUCE-INFORME. De este modo es posible que se ejecute "PERFORM CAMPO-INVALIDO" en PRODUCE-INFORME *antes* de que concluya su primera ejecución. La ejecución de una instrucción PERFORM sin que haya acabado una ejecución anterior (porque no se haya devuelto el control al párrafo anteriormente en ejecución) ocasiona casi siempre un bucle infinito.

7.90 Escriba el pseudocódigo de un programa para llevar una cuenta de cheques. El algoritmo dará entrada al saldo anterior, después a un conjunto de códigos de transacciones ("D" para depósitos, "C" para cheques) con su importe hasta el final del fichero. La salida consistirá en el total de depósitos, total de cheques y el saldo nuevo.

lee el saldo anterior  
 da a total-depósitos y total-cheques el valor cero  
 da a interruptor-fin el valor "no"  
 lee código-transacción, importe  
 al final de las transacciones da a interruptor-fin el valor "si"

```

ejecuta hasta que interruptor-fin = "si"
    si código-transacción = "D"
        suma importe a total-depósitos
    en caso contrario
        si código-transacción = "C"
            suma importe a total cheques
        en caso contrario
            imprima "código de transacción inválido"
        fin-si
    fin-si
    lee código-transacción, importe
        al final da a interruptor-fin el valor "si"
fin-ejecuta
nuevo-saldo = saldo-anterior + total-depósitos - total-cheques
imprime saldo-anterior, total-depósitos, total-cheques, saldo nuevo
parada

```

- 7.91 Escriba el pseudocódigo para implementar la tabla de decisión de la Figura 7-18.

```

si la cuenta no está en activo
    ejecute compra-desaprobada
en caso contrario
    si pago-último-mes recibido
        si compra no es mayor que límite-crédito
            ejecutar compra-aprobada
        en caso contrario
            ejecutar compra-aprobada
            ejecutar consulta-oficina-créditos
        fin-si
    en caso contrario
        si compra no es mayor que límite-crédito
            ejecutar compra-aprobada
            ejecutar consulta-oficina-créditos
        en caso contrario
            ejecutar compra-desaprobada
        fin-si
    fin-si
fin-si

```

Aunque la solución de arriba es directa, caben otras más compactas:

```

si la cuenta no está en activo ..
    ejecuta compra-desaprobada
en caso contrario
    si se ha recibido último-pago y la compra está dentro de límite
        ejecuta compra-aprobada
    en caso contrario
        si el último-pago no se ha recibido y la compra no está
            dentro del límite
            ejecuta compra-desaprobada
    en caso contrario
        ejecuta compra-aprobada
        ejecuta consulta-oficina-créditos
    fin-si
fin-si
fin-si

```

- 7.92 Dadas las siguientes condiciones y acciones, diseñe la tabla de decisión para aceptación de trabajos que le parezca.

<i>Condiciones</i>				<i>Acciones</i>			
(1) Salario y complementos aceptables		(1) Aceptar el trabajo					
(2) Se dispone de programas de formación		(2) Rechazar el trabajo					
(3) Oportunidades de promoción		(3) Negociar mejores condiciones					

Véase la Figura 7-23 que constituye una posible solución.

Salario	T	T	T	T	F	F	F	F
Educación	T	T	F	F	T	T	F	F
Promoción	T	F	T	F	T	F	T	F
Aceptar	X	X	X	X	X			
Rechazar						X	X	X
Negociar					X	X	X	X

Fig. 7-23

### Problemas complementarios

- 7.93 Escriba un programa que pueda utilizarse por un banco para escribir un calendario de amortización de préstamos hipotecarios. El programa de entrada al principal original, el tipo de interés anual (como porcentaje) y el pago mensual deseado. El programa imprimirá un informe con la información de cada mes durante la vida del préstamo: (i) número de pago, (ii) saldo anterior (antes de este pago), (iii) importe del pago, (iv) importe del pago de principal [= (iii)-(v)], (v) importe del pago de intereses es [= (ii)  $\times$  tipo/12], (vi) saldo nuevo [= (ii)-(iv)]. El programa imprime 20 líneas por página a doble espacio. Utilice apropiadamente cabeceras de columna, números de página, fecha, etc. *Respuesta.* Véase la Figura 7-24; un ejemplo de salida aparece en la Figura 7-25.

```

00001 IDENTIFICATION DIVISION.
00002
00003 PROGRAM-ID. MORTGAGE.
00004
00005 AUTHOR. LARRY NEWCOMER.
00006 INSTALLATION. PENN STATE UNIVERSITY -- YORK CAMPUS.
00007 DATE-WRITTEN. MAY 1983.
00008 SECURITY. NONE.
00009
00010 ENVIRONMENT DIVISION.
00011
00012 CONFIGURATION SECTION.
00013 SOURCE-COMPUTER. IBM-3081.
00014 OBJECT-COMPUTER. IBM-3081.
00015
00016 INPUT-OUTPUT SECTION.
00017 FILE-CONTROL.
00018
00019 SELECT FICHERO-TERMINOS

```

Fig. 7-24

```

00020      ASSIGN TO TERMINOS
00021      ORGANIZATION IS SEQUENTIAL
00022      ACCESS IS SEQUENTIAL
00023
00024      SELECT INFORME-PAGOS
00025          "ASSIGN TO PAGOS
00026          ORGANIZATION IS SEQUENTIAL
00027          ACCESS IS SEQUENTIAL
00028
00029
00030      DATA DIVISION.
00031
00032      FILE SECTION.
00033
00034      FD FICHERO-TERMINOS
00035          RECORD CONTAINS 80 CHARACTERS
00036          LABEL RECORDS ARE OMITTED
00037
00038
00039      01 REGISTRO-TERMINOS.
00040          05 TERMINOS-PRINCIPAL      PIC S9(7)V99.
00041          05 TERMINOS-TIPO-INTERES  PIC SV999.
00042          05 TERMINOS-PAGO        PIC S9(5)V99.
00043          05 FILLER             PIC X(61).
00044
00045      FD INFORME-PAGOS
00046          RECORD CONTAINS 132 CHARACTERS
00047          LABEL RECORDS ARE OMITTED
00048              LINAGE IS 60
00049                  WITH FOOTING AT 59
00050                  LINES AT TOP 3
00051                  LINES AT BOTTOM 3
00052
00053
00054      01 REGISTRO-PAGO           PIC X(132).
00055
00056      WORKING-STORAGE SECTION.
00057
00058      01 WS-INTER-Y-CONTADORES.
00059          05 WS-NUMERO-PAGINA    PIC S999      COMP-3.
00060          05 WS-INTER-FIN-FICHERO-TERMINOS  PIC XXX.
00061              88 FICHERO-TERMINOS-VACIO  VALUE "SI".
00062          05 WS-INTER-FIN-PAGOS     PIC XXX.
00063              88 PAGO-ULTIMO       VALUE "SI".
00064              88 PAGO-NORMAL        VALUE "NO".
00065
00066      01 WS-AREAS-CALCULO.
00067          05 WS-IMPORTE-INTERESES  PIC S9(5)V99  COMP-3.
00068          05 WS-IMPORTE-PAGO      PIC S9(5)V99  COMP-3.
00069          05 WS-SALDO            PIC S9(7)V99  COMP-3.
00070          05 WS-NUMERO-PAGO      PIC S9(3)    COMP-3.
00071          05 WS-TIPO-DE-INTERES  PIC SV999   COMP-3.
00072          05 WS-IMPORTE-PRINCIPAL  PIC S9(5)V99  COMP-3.
00073
00074      01 WS-FECHA.
00075          05 AÑO-SISTEMA        PIC 99.
00076          05 MES-SISTEMA        PIC 99.
00077          05 DIA-SISTEMA        PIC 99.
00078
00079      01 WS-CAB-LINEA-1.
00080          05 FILLER             PIC X(10)    VALUE SPACES.
00081          05 FILLER             PIC X(17)    VALUE "PAGOS HIPOTECA".
00082
00083          05 FILLER             PIC X(1)    VALUE SPACES.
00084          05 WS-CAB-MM          PIC Z9.
00085          05 FILLER             PIC X        VALUE "/".

```

Fig. 7-24 (cont.)

```

00086      05 WS-CAB-DD          PIC 99.
00087      05 FILLER           PIC X      VALUE "/".
00088      05 WS-CAB-AA          PIC 99.
00089      05 FILLER           PIC X(4)   VALUE SPACES.
00090      05 FILLER           PIC X(6)   VALUE "PAGINA".
00091      05 WS-CAB-NUM-PAGINA PIC ZZ9.
00092      05 FILLER           PIC X(83)  VALUE SPACES.
00093
00094      01 WS-CAB-LINEA-2.
00095          05 FILLER           PIC X(10)  VALUE "PAGO #".
00096          05 FILLER           PIC X(2)   VALUE SPACES.
00097          05 FILLER           PIC X(15)  VALUE "SALDO ANT".
00098          05 FILLER           PIC (2)   VALUE SPACES.
00099          05 FILLER           PIC (10)  VALUE "PAGO".
00100          05 FILLER           PIC X(2)   VALUE SPACES.
00101          05 FILLER           PIC X(10)  VALUE "PRINCIPAL".
00102          05 FILLER           PIC X(2)   VALUE SPACES.
00103          05 FILLER           PIC X(12)  VALUE "INTERES".
00104          05 FILLER           PIC X(2)   VALUE SPACES.
00105          05 FILLER           PIC X(55)  VALUE SPACES.
00106
00107          05 FILLER           PIC X(10)  VALUE "NUEVO SALDO".
00108          05 FILLER           PIC X(2)   VALUE SPACES.
00109          05 FILLER           PIC X(12)  VALUE SPACES.
00110          05 FILLER           PIC X(55)  VALUE SPACES.
00111
00112          05 FILLER           PIC X(3)   VALUE SPACES.
00113
00114      01 WS-LINEA.
00115          05 FILLER           PIC X(3)   VALUE SPACES.
00116          05 WS-NUMERO          PIC Z9.
00117          05 FILLER           PIC X(7)   VALUE SPACES.
00118          05 WS-SALDO-ANT        PIC Z,ZZZ,ZZZ.99.
00119          05 FILLER           PIC X(5)   VALUE SPACES.
00120          05 WS-PAGO            PIC ZZ,ZZZ.99.
00121          05 FILLER           PIC X(3)   VALUE SPACES.
00122          05 WS-PRINCIPAL         PIC ZZ,ZZZ.99.
00123          05 FILLER           PIC X(3)   VALUE SPACES.
00124          05 WS-INTERES          PIC ZZ,ZZZ.99.
00125          05 FILLER           PIC X(3)   VALUE SPACES.
00126          05 WS-SALDO-NUEVO        PIC Z,ZZZ,ZZZ.99.
00127          05 FILLER           PIC X(55)  VALUE SPACES.
00128
00129      PROCEDURE DIVISION.
00130
00131      000-MODULO-EJECUTIVO.
00132
00133      PERFORM 100-CAPTURA-TERMINOS
00134      PERFORM 200-PRODUCE-LINEA
00135          UNTIL PAGO-ULTIMO
00136      PERFORM 300-PRODUCE-PAGO-ULTIMO
00137      PERFORM 400-CONCLUSION
00138      STOP RUN
00139
00140
00141      100-CAPTURA-TERMINOS.
00142
00143      MOVE "NO" TO WS-INTER-FIN-FICHERO-TERMINOS
00144          WS-INTER-FIN-PAGOS
00145      MOVE ZERO TO WS-NUMERO-PAGINA
00146          WS-NUMERO-PAGO
00147      OPEN    INPUT   FICHERO-TERMINOS
00148          OUTPUT  INFORME-PAGOS
00149      READ FICHERO-TERMINOS
00150          AT END
00151          MOVE "SI" TO WS-INTER-FIN-FICHERO-TERMINOS
00152

```

Fig. 7-24 (cont.)

```

00153    IF FICHERO-TERMINOS-VACIO
00154        MOVE "SI" TO WS-INTER-FIN-PAGOS
00155        DISPLAY "*** FICHERO TERMINOS VACIO ***"
00156    ELSE
00157        MOVE TERMINOS-PRINCIPAL      TO WS-SALDO
00158        MOVE TERMINOS-TIPO-INTERES TO WS-TIPO-DE-INTERES
00159        MOVE TERMINOS-PAGO       TO WS-IMPORTE-PAGO
00160        ACCEPT WS-FECHA FROM DATE
00161        MOVE AÑO-SISTEMA        TO WS-CAB-AA
00162        MOVE MES-SISTEMA        TO WS-CAB-MM
00163        MOVE DIA-SISTEMA        TO WS-CAB-DD
00164        PERFORM 600-PROCEDURE-HEADINGS
00165
00166
00167    200-PRODUCE-LINEA.
00168
00169    ADD 1 TO WS-NUMERO-PAGO
00170    COMPUTE WS-IMPORTE-INTERESES =
00171        (WS-SALDO * WS-TIPO-DE-INTERES) / 12
00172    SUBTRACT WS-IMPORTE-INTERESES FROM WS-IMPORTE-PAGO
00173        GIVING WS-IMPORTE-PRINCIPAL
00174    IF WS-IMPORTE-PRINCIPAL NOT LESS THAN WS-SALDO
00175        MOVE "SI" TO WS-INTER-FIN-PAGOS
00176
00177    IF PAGO-NORMAL
00178        MOVE WS-NUMERO-PAGO      TO WS-NUMERO
00179        MOVE WS-SALDO          TO WS-SALDO-ANT
00180        MOVE WS-IMPORTE-PAGO    TO WS-PAGO
00181        MOVE WS-IMPORTE-PRINCIPAL TO WS-PRINCIPAL
00182        MOVE WS-IMPORTE-INTERESES TO WS-INTERES
00183        SUBTRACT WS-IMPORTE-PRINCIPAL FROM WS-SALDO
00184        GIVING WS-SALDO-NUEVO
00185        WS-SALDO
00186        PERFORM 500-IMPRIME-LINEA
00187
00188
00189    300-PRODUCE-PAGO-ULTIMO.
00190
00191    MOVE WS-NUMERO-PAGO      TO WS-NUMERO
00192    MOVE WS-SALDO          TO WS-SALDO-ANT
00193        WS-PRINCIPAL
00194    ADD WS-SALDO WS-IMPORTE-INTERESES
00195        GIVING WS-PAGO
00196    MOVE WS-IMPORTE-INTERESES      TO WS-INTERES
00197    MOVE ZERO                 TO WS-SALDO-NUEVO
00198    PERFORM 500-IMPRIME-LINEA
00199
00200
00201    400-CONCLUSION.
00202
00203    CLOSE FICHERO-TERMINOS
00204        INFORME-PAGOS
00205
00206
00207    500-IMPRIME-LINEA.
00208
00209    WRITE REGISTRO-PAGO
00210        FROM WS-LINEA
00211        AFTER ADVANCING 2 LINES
00212        AT END-OF-PAGE
00213        PERFORM 600-PRODUCE-CABECERA
00214
00215
00216    600-PRODUCE-CABECERA.
00217
00218    ADD 1 TO WS-NUMERO-PAGINA

```

Fig. 7-24 (cont.)

```

00219      MOVE WS-NUMERO-PAGINA TO WS-CAB-NUM-PAGINA
00220      WRITE REGISTRO-PAGO
00221          FROM WS-CAB-LINEA-1
00222          AFTER ADVANCING PAGE
00223      WRITE REGISTRO-PAGO
00224          FROM WS-CAB-LINEA-2
00225          AFTER ADVANCING 2 LINES
00226

```

Fig. 7-24 (cont.)

PAGO #	PAGOS HIPOTECA SALDO ANTERIOR	5/09/83 PAGO	PAGINA 1 PRINCIPAL	INTERES	NUEVO SALDO
1	6,000.00	1,000.00	950.00	50.00	5,050.00
2	5,050.00	1,000.00	957.92	42.08	4,092.08
3	4,092.08	1,000.00	965.90	34.10	3,126.18
4	3,126.18	1,000.00	973.95	26.05	2,152.23
5	2,152.23	1,000.00	982.07	17.93	1,170.16
6	1,170.16	1,000.00	990.25	9.75	179.91
7	179.91	181.40	179.91	1.49	.00

Fig. 7-25

### Ejercicios de programación

- 7.94 Dados los siguientes registros de ventas de un fichero en tarjetas:

Campo	Longitud y tipo de datos
ID ventas	X(5)
fecha cierre	X(6) en formato aammdd
importe	S9(4)V99
año	
último importe	S9(4)V99
año	
sin utilizar	57 bytes

Imprima un informe que consista sólo en las cabeceras adecuadas y dos líneas: (1) ID ventas, importe este año, importe último año e incremento sobre el año anterior, todo ello relativo al vendedor con el máximo incremento en ventas; (2) la misma información, pero relativa al vendedor con el incremento de ventas más pequeño. El incremento de ventas se define por

$$(importe este año) - (importe año pasado)$$

Es posible en (1) y probable en (2) que el incremento sea negativo, es decir, que las ventas se hayan reducido.

- 7.95 Modifique el Problema 7.94 para que el programa también imprima el total de ventas del año, total de ventas del año anterior, ventas medias del año, ventas medias del año anterior. Haga que la salida sea lo más presentable posible.
- 7.96 Modifique el Problema 7.93 para que el programa imprima, después de cada doce meses, el principal total pagado y los intereses totales pagados durante el período de 12 meses precedentes. Debe además imprimir el interés total pagado en el préstamo al final del informe.
- 7.97 Escriba un programa de validación (o edición) para el fichero de entrada del Problema 7.94. El programa tiene que imprimir un informe listando sólo aquellos registros que contengan un campo no-numérico. Los campos inválidos deben subrayarse en el informe. Recuerde que 1, 2, 3 o los 4 campos pueden ser inválidos en el mismo registro. Imprima títulos apropiados, fecha, números de página, cabeceras de columna. Verifique el programa para todas las combinaciones de registros de entrada posibles.

# Capítulo 8

## Diseño de programas: El método descendente y modular (Top-Down)

Hasta ahora no se ha dicho nada en el libro que permita al lector enfrentarse con un problema de programación. En este capítulo se presenta un *procedimiento de desarrollo de programas*, que ha demostrado su efectividad y que puede servir tanto a los principiantes como a los profesionales.

### 8.1 PASO 1.: DEFINICION DEL PROBLEMA

El programador debe estudiar en primer lugar todo el material de diseño creado por los analistas de sistemas que hayan trabajado en el proyecto. Entre dicho material estarán los diseños de los registros, de las salidas por impresora y una descripción de los programas. Es posible que un *programador analista* tenga que crear, él mismo, todo este material.

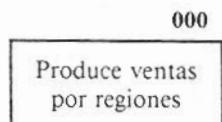
### 8.2 PASO 2.: DISEÑO GENERAL DEL PROGRAMA

El siguiente paso consiste en producir un *diseño descendente y modular* del programa y documentar el mismo con un *organigrama estructurado* [también denominado *tabla de contenidos (VTOC)* u *organigrama jerárquico*]. Se dice que el diseño es *modular* porque divide el programa en partes o *módulos*, cada uno de los cuales lleva a cabo una función concreta. Cada módulo se puede concretar en un párrafo de COBOL. Se dice que el diseño es *descendente* porque se ocupa primero de las funciones de “más alto nivel” (control) que debe realizar el programa y en último lugar se ocupa de las funciones de “más bajo nivel” (detalladas) que hay que llevar a cabo.

**EJEMPLO 8.1** La forma final de un organigrama estructurado para un programa que genere un informe de ventas aparece en la Figura 8-1. Si cada módulo (rectángulo) va a ser un párrafo de COBOL, las líneas de conexión indican qué párrafos ejecutarán a otros párrafos. Observe que cada módulo dispone de un nombre descriptivo que identifica la función que desempeña y también un número de identificación. La combinación de número y nombre de módulo no debe exceder el límite de 30 caracteres que marca COBOL al nombre de un párrafo. Observe que la esquina sombreada indica que el módulo de que se trate es ejecutado mediante *PERFORM* (*llamado*) desde más de un módulo distinto; por ejemplo, 230 se ejecuta por 100 y 220. Obviamente 230-IMPRIME-CABECERAS aparecerá sólo una vez en el programa fuente en COBOL; se duplican módulos en el organigrama para mantener la estructura jerárquica (*arbórea*).

¿Cómo actuó el programador al ir diseñando la Figura 8-1 *módulo a módulo y de arriba a abajo*?

1. El problema consiste en generar un informe de las ventas por regiones; por tanto se situará un módulo en la parte superior del organigrama con el número 000 y la etiqueta “Produce Ventas Regiones”:



2. ¿Qué otras funciones hay que llevar a cabo para generar el informe de ventas por regiones? Tres fundamentales: inicializar el programa para preparar el procesamiento de la primera región, proceso de los registros individuales y manipulación de los totales al final del trabajo. (No existen formas correctas

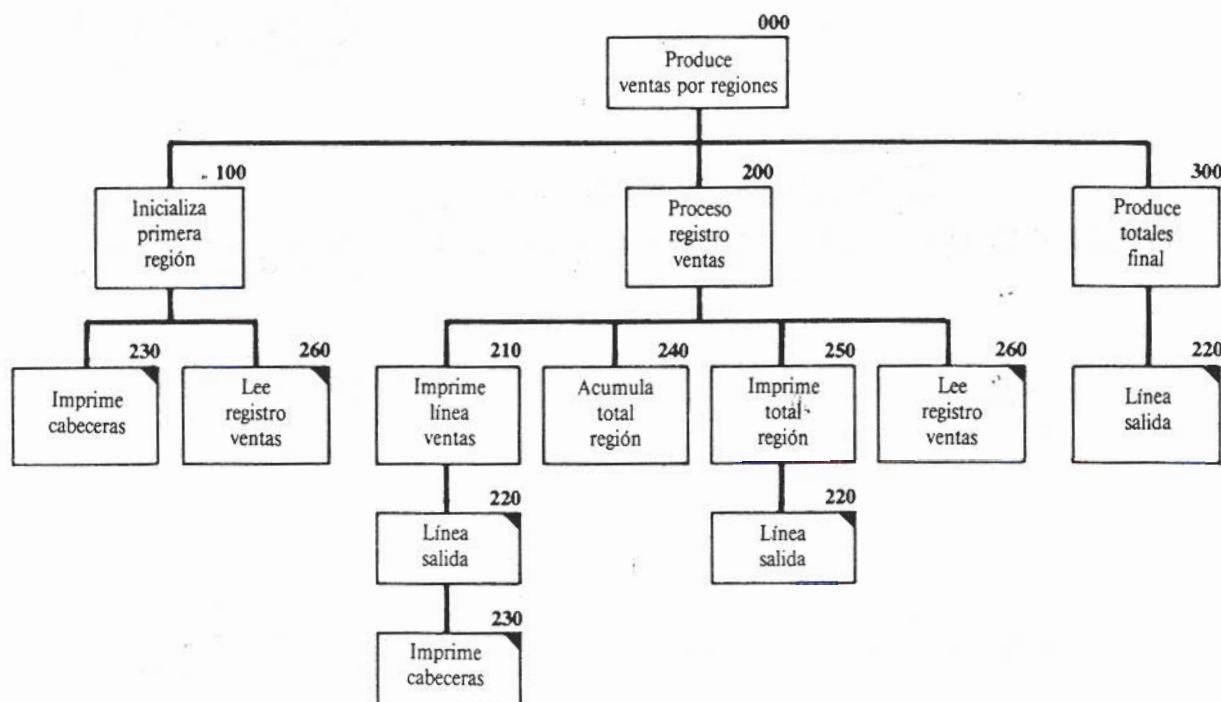


Fig. 8-1

o incorrectas de dividir un problema en módulos. Aunque algunas soluciones son mejores que otras, en general la solución no es única.) Después de identificar las tres funciones principales que debe realizar 000-PRODUCE-VENTAS-REGIONES y de que dichas funciones tengan sus propios módulos, el organigrama tiene el aspecto que se muestra en la Figura 8-2.

3. Despues de dividir el módulo de *nivel 0* en las tres funciones principales de control lógico, se debe hacer lo mismo con cada módulo del *nivel 1*. ¿Qué funciones hay que llevar a cabo para inicializar el procesamiento de la primera región? Habrá que imprimir las cabeceras del informe y dar lectura al primer registro de ventas para ver cuál es la primera región. Del mismo modo, para procesar un registro de ventas el programa debe: (1) imprimir una línea de ventas en el informe con el contenido de este primer registro; (2) acumular el total de ventas de este registro en el total de la región; (3) si fuera necesario, imprimir el total de la región (si la región hubiera cambiado); (4) dar entrada al siguiente registro de ventas. Así se tienen los módulos del *nivel 2* (módulos 230, 260, 210, 240, 250, 260, 220). Ahora es necesario descomponer el nivel 3 y así sucesivamente hasta que todos los módulos contengan subfunciones triviales.

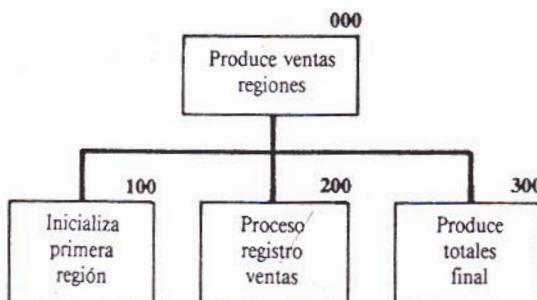


Fig. 8-2

El organigrama completo sirve para organizar la división de procedimientos del programa. La experiencia demuestra que si se intenta "ahorrar tiempo" saltándose el proceso de diseño —o escribiendo el programa primero y diseñando después—, se empleará en realidad más tiempo y el programa será menos eficiente.

### Reglas para la construcción de organigramas estructurados

1. Un módulo, una función.
2. Numere cada módulo. Los Ejemplos 8.2 y 8.3 presentan dos esquemas de numeración. Si un módulo aparece más de una vez en un organigrama estructurado (por ejemplo, si se llama desde más de un módulo), asígnale el número que esté asociado con la llamada más frecuente. También los módulos repetidos deben marcarse especialmente (como en la Fig. 8-1).

**EJEMPLO 8.2** En el sistema de *numeración horizontal* (Fig. 8.3), al nivel 0 se le asigna el número 000; al nivel 1 se le asignan los números 100, 200, 300,...; los "descendientes" de 100 se numeran, nivel por nivel, 110, 120, 130,...; los descendientes de 200 se numeran, nivel por nivel, 210, 220, 230,...; etc. Observe que la numeración se puede ir asignando de arriba a abajo conforme se va desarrollando el organigrama estructurado.

**EJEMPLO 8.3** En el sistema de *numeración vertical* (Fig. 8-4) no se procede nivel por nivel, sino que se procede de forma descendente y de izquierda a derecha en cada nudo. De este modo, se asigna 000 al nivel 0 y se gira a la izquierda para dar los números 100, 110, 120,... Al "final de la línea vertical" se retrocede al último nudo y se toma el primer módulo de la izquierda que no se haya numerado todavía. La numeración continúa con incrementos de 10 en cada paso hasta acabar con todos los descendientes de 100. Después se pasa a 200 (el primero a la izquierda no numerado todavía en el nudo de 000) y se utiliza la misma regla para numerar a todos los descendientes de 200, y así sucesivamente. Una manera simple de hacer esto es dibujar una curva continua rodeando todo el organigrama (véase la Fig. 8-4): se puede numerar siguiendo la curva.

El sistema de numeración vertical tiende a dejar los módulos llamador y llamado cerca uno de otro en el programa y se recomienda su uso con los *sistemas de almacenamiento virtual* (consulte el manual del sistema).

3. Diseñe módulos con *cohesión funcional*, es decir, que cada instrucción del módulo contribuya directamente a llevar a cabo funciones exclusivas del módulo.
4. Diseñe módulos que eviten la *duplicidad*; en otras palabras, intente minimizar el número de datos a los que se accede desde dos o más párrafos COBOL.
5. Un módulo no debe requerir más de 50 líneas de COBOL (o sea, aproximadamente una página del programa fuente). Cualquier módulo mayor debe revisarse para ver si existe posibilidad de trasladar parte de sus operaciones a módulos de nivel inferior.
6. Los módulos deben llamar únicamente a aquellos módulos que sean descendientes suyos.
7. El nivel 0 es el *módulo ejecutivo* y es responsable generalmente de llamar repetidamente al módulo de nivel 1, implementando de este modo el bucle principal del programa. Frecuentemente se trata de un bucle del tipo "ejecuta hasta el final del fichero".
8. Intente utilizar un único módulo READ por fichero de entrada y un módulo WRITE por fichero de salida; el módulo READ/WRITE apropiado se puede llamar desde cualquier módulo que lo necesite. Se exceptúan el caso de los ficheros de impresión para los cuales podría haber un módulo WRITE para las cabeceras y otro módulo WRITE para las líneas.
9. Si un módulo no iterativo llega a ser trivial, debe considerar la posibilidad de eliminarlo y llevar las instrucciones a su padre lógico.

### 8.3 PASO 3.: DISEÑO DETALLADO DEL PROGRAMA

Después de haber construido el organigrama estructurado del programa (VTOC), el paso siguiente es diseñar la lógica detallada de cada módulo. Esto puede hacerse mediante diagramas de flujo

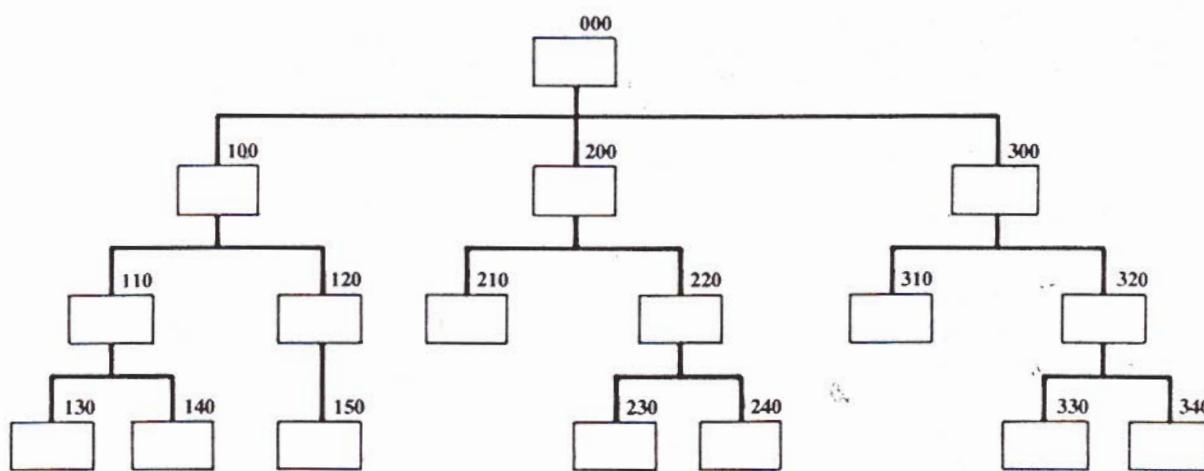


Fig. 8-3

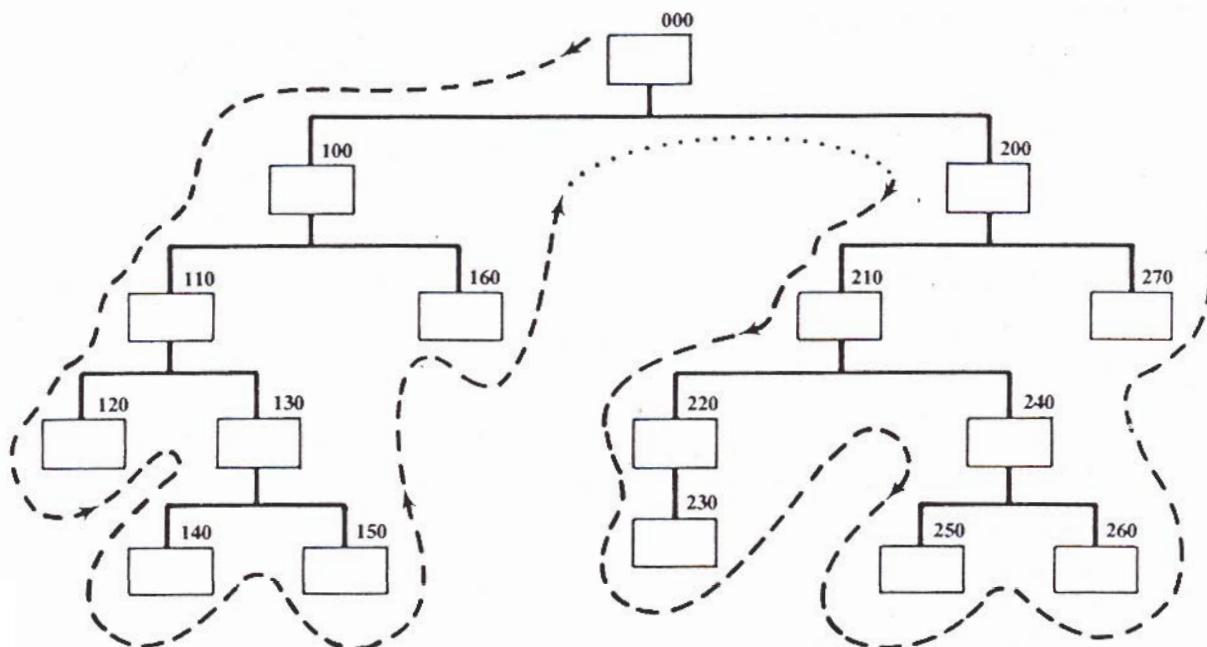


Fig. 8-4

estructurados o mediante pseudocódigo, como se comentó en el Capítulo 7; sin embargo, una de las herramientas más útiles son los *diagramas jerárquicos del ciclo entrada-procesamiento-salida* también llamados *diagramas HIPO*. En una página simple de un diagrama HIPO (Fig. 8-5) se combina la descripción detallada de la lógica de un módulo mediante pseudocódigo con una indicación de qué entradas o variables necesita el módulo y de qué salidas o variables produce o modifica. Es de notar que un mismo elemento puede aparecer en el bloque de entrada y en el de salida. Las páginas HIPO se desarrollan al igual que VTOC con el procedimiento de arriba a abajo. Véase la Figura 8-11 que contiene un ejemplo de un diagrama HIPO.

Cuando se termina el diagrama HIPO, debe verificarse el pseudocódigo de cada bloque para localizar la mayor parte de los errores antes de pasar a la codificación y comprobación.

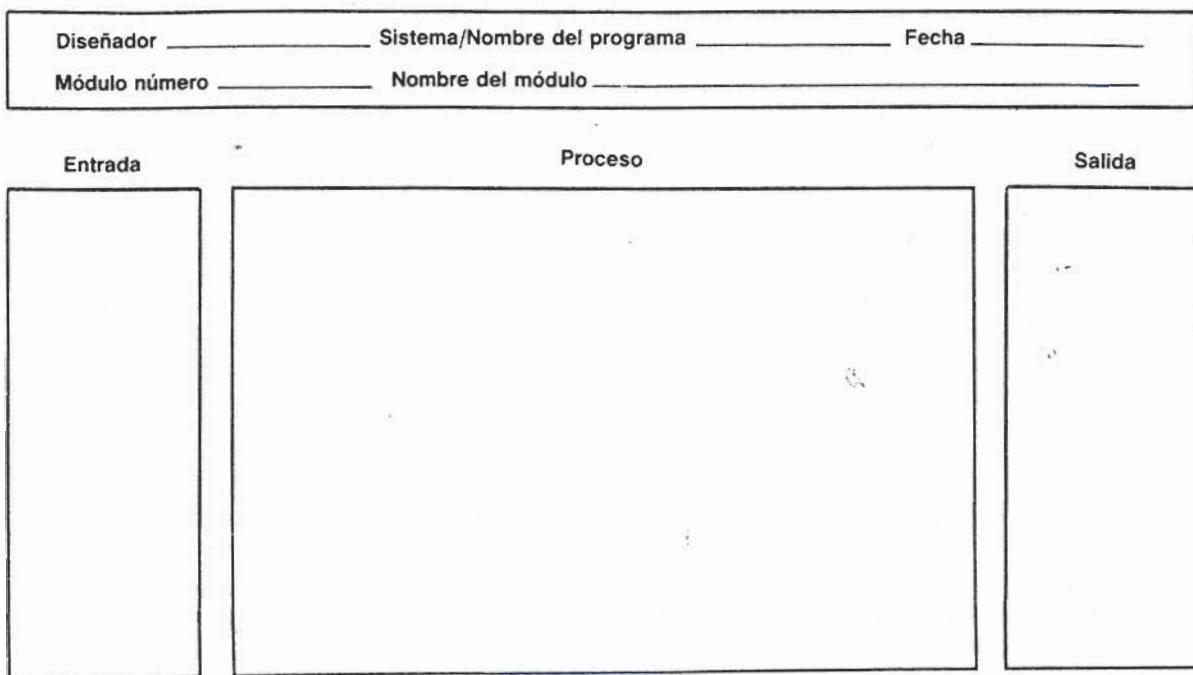


Fig. 8-5

#### 8.4 PASO 4.<sup>o</sup>: PREPARACION DE UN PLAN PARA LA CODIFICACION Y COMPROBACION DEL PROGRAMA

El método descendente puede y debe aplicarse también a la codificación y comprobación del programa. Esto implica:

- (1) Deben codificarse primero los módulos de más alto nivel; sólo después de que éstos hayan sido comprobados y depurados se pasará al siguiente nivel.
- (2) No debe codificarse un módulo hasta que se hayan codificado y comprobado aquellos módulos que le separan del módulo ejecutivo (nivel 000).
- (3) Para comprobar un módulo que llame a otros que no se hayan codificado se pueden utilizar recursos que simulen la ejecución de los mismos. Estos recursos suelen consistir en mensajes impresos mediante la instrucción DISPLAY cuando corresponda la ejecución de un párrafo relativo a un módulo no codificado todavía. (Véanse la Fig. 8-12, líneas 145-158 y la Fig. 8-13, líneas 193-215).

Algunos programadores prefieren codificar y comprobar los módulos nivel por nivel (como en la secuencia de numeración horizontal; véase el Ejemplo 8.2). Otros, por el contrario, emplean el orden de la numeración vertical (véase el Ejemplo 8.3). Siempre que se siga la regla 2, cualquier combinación de estos dos procedimientos es válida.

**EJEMPLO 8.4** Un plan de codificación y comprobación para el programa de la Figura 8-1 sería:

<i>Orden</i>	<i>Módulos</i>	<i>Comprobación con</i>
1	000	ficheros vacíos, mensajes para 100, 200 y 300
2	100, 200, 300	datos de una única región; mensajes para 230, 260, 210, 240, 250, 220
3	210, 220, 230, 260	datos para dos regiones
4	240, 250	(a) ficheros vacíos (b) datos de todas las regiones, incluida una región con un solo registro (c) datos suficientes para que se rebase una página

## 8.5 PASO 5.: ENSAMBLAJE DE DATOS DE PRUEBA

Algunas veces los datos de comprobación se preparan por el programador, otras pueden generarse por programas especiales y otras se pueden tomar copiando los ficheros que manipulará el programa cuando esté ultimado. Una vez que estén preparados los datos de comprobación, el programador debe determinar la salida correcta que corresponde a dichos datos para que puedan detectarse las diferencias y efectuar la depuración del programa correspondiente.

Lo ideal es que después del proceso de comprobación, *cada componente lógico de cada módulo* haya sido comprobado. En particular, algunos datos estarán preparados para ejecutar las rutinas correspondientes a las condiciones “verdaderas” de las instrucciones IF y otros datos para las condiciones “falsas”. Las iteraciones deben comprobarse para los casos de cero y unas repeticiones. En los programas largos puede que sea imposible disponer de datos suficientes para comprobar completamente la lógica del programa. En este caso deben comprobarse por lo menos las partes más importantes del mismo.

## 8.6 PASO 6.: CODIFICACION Y COMPROBACION DESCENDENTE

A continuación el programa debe codificarse y comprobarse módulo por módulo de acuerdo con la secuencia establecida en el plan de codificación y comprobación y utilizando los datos preparados al efecto. Véase el Problema 8.30 para la implementación de este paso.

## 8.7 PASO 7.: CONFECCION DE LA DOCUMENTACION DEL PROGRAMA

Después de comprobar y depurar el programa totalmente, deben juntarse el organigrama estructurado, diagrama HIPO, plan de codificación y comprobación, datos de comprobación, salida del proceso de comprobación y listado fuente para conformar la documentación del programa. En los diagramas VTOC e HIPO debe incluirse cualquier cambio que haya habido que hacer durante el proceso de comprobación/depuración. Toda esta documentación debe conservarse para que sirva de ayuda a los programadores de mantenimiento que tengan que realizar modificaciones posteriores.

### Preguntas de repaso

- 8.1 Explique el primer paso en el diseño y desarrollo de programas.
- 8.2 ¿Qué es un organigrama estructurado (VTOC)? ¿Para qué se utilizan en el diseño de programas?
- 8.3 ¿Qué se entiende por diseño descendente?
- 8.4 ¿Qué es un “módulo”?
- 8.5 Defina el concepto de “nivel” en un organigrama estructurado.
- 8.6 Compare y comente los sistemas de numeración horizontal y vertical.
- 8.7 Explique la importancia de la regla “un módulo, una función”.
- 8.8 ¿Por qué deben tener los módulos cohesión funcional? ¿Por qué deben evitar la duplicidad?
- 8.9 ¿Qué es el módulo “principal” o “ejecutivo”?
- 8.10 ¿Por qué se recomienda sólo una instrucción READ o WRITE por cada fichero utilizado en el programa? ¿Cómo se puede implementar esto?
- 8.11 Explique el uso de los diagramas HIPO en el diseño detallado de un programa.

- 8.12 Comente lo que aparece en los bloques de entrada, proceso y salida de un diagrama HIPO.
- 8.13 ¿Qué se entiende por “codificación y comprobación descendente”?
- 8.14 ¿Qué es un “mensaje de comprobación”?
- 8.15 Compare los sistemas horizontal y vertical en la codificación y comprobación descendente.
- 8.16 ¿Para qué sirve la comprobación del programa? Explique cómo deben alterarse los datos de comprobación durante el proceso de comprobación descendente.
- 8.17 ¿Qué importancia tiene la documentación del programa? Explique de qué modo las herramientas de diseño sirven como documentación del programa.

### Problemas resueltos

- 8.18 Numere los módulos de la Figura 8-6: (a) horizontalmente, (b) verticalmente.

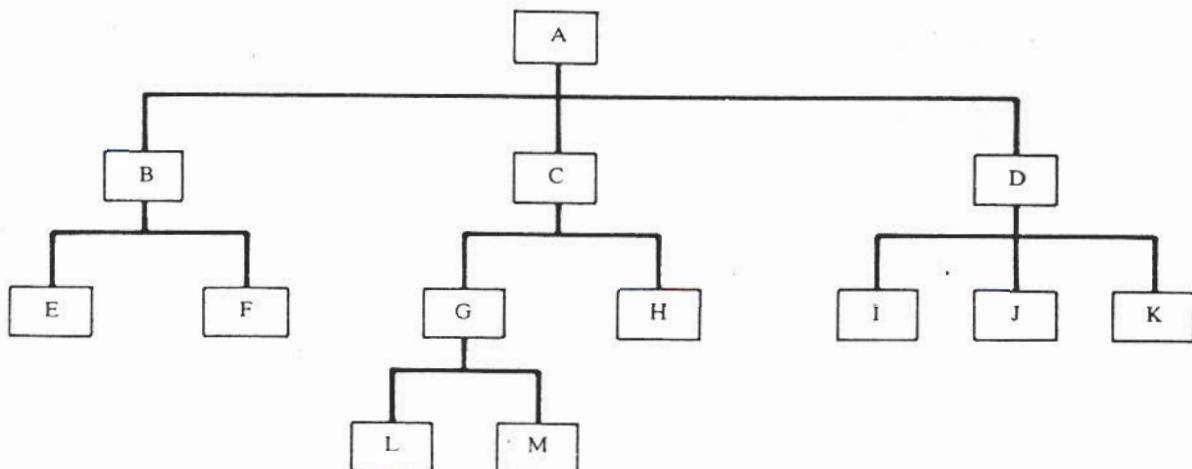


Fig. 8-6

- (a) A 000; B 100, E 110, F 120; C 200, G 210, H 220, L 230, M 240; D 300, I 310, J 320, K 330.  
 (b) A 000; B 100, E 110, F 120; C 200, G 210, L 220, M 230, H 240; D 300, I 310, J 320, K 330.

- 8.19 ¿Tiene cohesión el siguiente módulo (párrafo COBOL)?

EJEMPLO-PARA.

```

MOVE SPACES TO AREA-MENSAJE-SALARIOS
MOVE ZEROS TO IMPORTE-SALARIO-TOTAL
PERFORM LECTURA-TARJETAS-TIEMPO
ADD HORAS-TIEMPO-TARJETA TO HORAS-TOTALES
IF ID-TARJETA-TIEMPO NOT NUMERIC
  MOVE "SI" TO INTERRUPTOR-INVALIDO

PERFORM CALCULO-IMPUESTOS
IF CONT-LINEAS-INFORME GREATER THAN 30
  PERFORM IMPRESION-CABEZAS
  
```

Un módulo se dice que está cohesionado cuando todas las instrucciones que en él aparecen tienen una función bien definida. Un módulo que (1) inicializa algunos valores, (2) da entrada a registros, (3) acumula un total, (4) valida un campo, (5) calcula la retención del impuesto sobre la renta y (6) imprime las cabeceras al saltar de página, tiene una cohesión muy pobre.

.....

CAPTURA-REGISTRO-ENTRADA.  
FICHERO-ENTRADA  
AT END  
MOVE "SI" TO INTER-FIN-FICHERO

- 8.22 Uno de los módulos de la Figura 8-1 es trivial; diga cuál.

240-ACUMULA-TOTAL-REGION contendrá únicamente una instrucción ADD, que puede incluirse fácilmente en el módulo que lo llama, es decir, en 200-PROCESO-REGISTRO-VENTAS.

- 8.23 ¿Por qué los primeros datos de comprobación deben ser simples?

Por lo general, un programa nuevo tendrá fallos. Si los datos iniciales fueran complicados, los fallos interfirirían entre sí haciendo más difícil identificarlos. Mas aún, en el proceso de comprobación, cuando la mayor parte de los errores hayan sido corregidos, se pueden utilizar datos más complejos.

- 8.24 Suponga que A, B y C son campos de un registro de entrada. Diseñe un conjunto de valores para A, B y C que sirvan para comprobar el contenido lógico del siguiente programa:

```

IF A LESS THAN B
  MOVE...
  ADD...
  IF A LESS THAN C
    PERFORM...
  ELSE
    PERFORM...
ELSE
  SUBTRACT...
  IF B EQUAL C
    MOVE...
  ELSE
    PERFORM
    IF A EQUAL C
      MOVE...
  
```

Se dibuja el *árbol de decisión* para los IF anidados de la Figura 8-7; así se aprecian todas las ramificaciones lógicas desde la "raíz"  $A < B$  hasta las "hojas" que son los puntos finales del árbol. Hay cinco combinaciones posibles para cuya comprobación los datos apropiados aparecen en la Tabla 8-1.

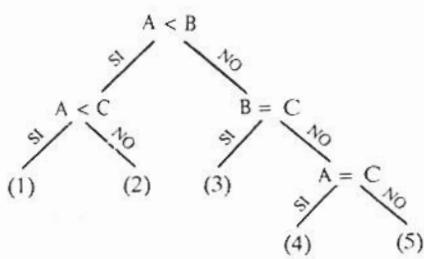


Fig. 8-7

Tabla 8-1

Combinación	A	B	C
(1)	1	2	3
(2)	1	2	0
(3)	2	1	1
(4)	2	1	2
(5)	2	1	3

- 8.25 Muchos programas que generan informes de negocios obtienen la información de ficheros en cinta o en disco. Estos programas se construyen alrededor de un bucle principal en el que se da entrada a un registro, se le procesa y si fuera necesario se da salida al resultado. Hay dos formas de diseñar la lógica de dicho programa. En la primera forma, el cuerpo del bucle principal (1) da entrada a un registro lógico; (2) comprueba si se ha alcanzado el final del fichero; (3) si no se ha alcanzado el final del fichero, se procesa el registro lógico al que se acaba de dar entrada. Escriba el pseudocódigo necesario para ilustrar esta lógica en un programa lista el salario mensual de los empleados leyendo el fichero maestro de empleados.

```

fija interruptor-fin-de-fichero con "no"
abre maestro-empleados y fichero-informe
ejecuta hasta que interruptor-fin-de-fichero sea "si" (bucle principal):
    lee registro maestro-empleados
        al final da a interruptor-fin-de-fichero el valor "si"
    si interruptor-fin-de-fichero es "no"
        ejecuta proceso-registro-empleado
        ejecuta salida-resultado
    fin-si
fin-ejecuta
cierra maestro-empleados y fichero-informe
parada

```

Es muy importante comprobar si se ha alcanzado el fin del fichero antes de procesar un registro lógico (véase el Ejemplo 6.10). En algunos sistemas el área destinada al registro lógico (FD) todavía contiene el último registro leído cuando tuvo lugar la condición de fin de fichero. En ese caso, no comprobar la condición de fin de fichero antes de procesar un registro implicaría procesar dos veces el último registro.

- 8.26 La segunda forma de realizar un programa como el del Problema 8.25 utiliza una *lectura inicial* del fichero antes del bucle principal del programa. Como ya se ha dado entrada a un registro lógico cuando se entra en el bucle principal, lo primero que éste debe hacer es (1) procesar el registro lógico y dar salida al resultado; después (2) dar lectura al siguiente registro lógico (que será procesado en el siguiente bucle durante el paso 1). Este sistema tiene dos ventajas: (i) no es necesario comprobar la condición de fin de fichero en el interior del bucle; y (ii) como al primer registro se le da entrada por separado, puede recibir un procesamiento especial (véase el Problema 8.27). Vuelva a resolver el Problema 8.25 con el sistema de lectura inicial.

```

fija interruptor-fin-de-fichero con "no"
abre maestro-empleados y fichero-informe
lee registro maestro-empleados (lectura inicial)
    al final: da a interruptor-fin-de-fichero el valor "si"
...(cualquier procesamiento preciso para el primer registro lógico)
ejecuta hasta que interruptor-fin-de-fichero sea "si" (bucle principal):
    ejecuta proceso-registro-empleado
    ejecuta salida-resultado
    lee registro maestro-empleados
        al final: da a interruptor-fin-de-fichero el valor "si"
fin-ejecuta
cierra maestro-empleados y fichero-informe
parada

```

- 8.27 Resuelva el Problema 8.25 (no por el procedimiento de lectura inicial) suponiendo que el primer registro tiene que recibir un procesamiento especial.

```

fija interruptor-fin-de-fichero con "no"
fija interruptor-primeravez con "si"
abre maestro-empleados, fichero-informe
ejecuta hasta que interruptor-fin-de-fichero sea "si" (bucle principal):

```

```

lee registro maestro-empleados
    al final: da a interruptor-fin-de-fichero el valor "si"
    si interruptor-fin-de-fichero es "no"
        si interruptor-primer-vez es "si"
            ejecuta proceso-primer-registro
            ejecuta salida-primer-registro
            fija interruptor-primer-vez con "no"
        en caso contrario
            ejecuta proceso-registro-empleado
            ejecuta salida-resultados
        fin-si
    fin-si
fin-ejecuta
cierra maestro-empleados y fichero-informe
parada

```

Observe la importancia de dar el valor "no" al interruptor-primer-vez después de que el primer registro reciba su procesamiento especial. Si no se hiciera esto, cada registro del fichero se trataría como si fuera el primero.

- 8.28 La técnica de lectura inicial es especialmente apropiada en los *problemas de control de ruptura*, en los que los registros lógicos de un fichero están clasificados por uno o más *campos de control*. Cuando el valor de un campo de control de un registro lógico es distinto del valor del mismo campo del registro anterior, se produce un *control de ruptura*; en ese momento tendrá lugar un procesamiento especial. Por ejemplo, supóngase que el fichero del Problema 8.25 incluye el número del departamento de cada empleado y que el fichero está clasificado por orden creciente del número de departamento. Imprima un informe a partir del fichero que incluya no sólo los salarios individuales, sino también los subtotales por departamento y el total.

```

fija fin con "no" (interruptor fin de fichero)
abre ficheros
fija con cero: total-departamento, gran-total
lee registro maestro-empleados
    al final: fija fin con "si"
    si fin es "no":
        mueve núm-departamento-actual a núm-departamento-anterior
    fin-si
ejecuta hasta que fin sea "si" (bucle principal):
    si núm-departamento-actual es distinto de núm-departamento-anterior
        (control de ruptura)
            ejecuta imprime-totales-departamento
            suma total-departamento a gran-total
            mueve cero a total-departamento
            mueve núm-departamento-actual a núm-departamento-anterior
    fin-si
ejecuta imprime-salario-empleado
suma salario-empleado a total-departamento
lee registro maestro-empleados
    al final: fija fin con "si"
fin-ejecuta
(el fin de fichero debe tratarse como una ruptura para imprimir el total del último departamento)
ejecuta imprime-totales-departamento
suma total-departamento a gran-total
ejecuta imprime-gran-total
cierra ficheros
parada

```

A continuación se dan algunos comentarios al programa de control de ruptura con un solo campo de control (*nivel-único*):

- (1) Se ha comprobado la condición de fin de fichero en el primer registro (lectura inicial). Normalmente no se habrá llegado al final, pero es posible (por ejemplo, por errores en el lenguaje de control de trabajos, errores del operador durante la ejecución del programa, etc.). Al dar al número del departamento anterior el valor del número del departamento del primer registro se asegura que no se efectuará erróneamente el control de ruptura al entrar en el bucle principal.
- (2) Debe inicializarse a cero el total-departamento cada vez que cambia el número de departamento (en caso de no hacerse el total-departamento acumularía *todos* los salarios de todos los departamentos).
- (3) Después de que se produzca un cambio en el campo de control, el *siguiente* cambio será ocasionado por un valor distinto del que acaba de producir el cambio. Por tanto, el núm-departamento-anterior debe tomar el valor del núm-departamento en cada ruptura de control.
- (4) El gran-total se obtiene del modo más eficiente sumando los totales de los departamentos. Esto debe hacerse en cada ruptura de control.
- (5) Cuando tiene lugar el fin de fichero, se trata como un control de ruptura en el que no se modifica el campo de control. Esto implica que el total del departamento tiene que imprimirse al final del fichero; también el último total de departamento debe sumarse al gran total.

**8.29** Suponga que en el Problema 8.28 cada registro del fichero maestro de empleados contiene un número de división y que el fichero está clasificado primero por número de división (*primer campo de control*) y dentro de cada división por número de departamento (*segundo campo de control*). Vuelva a diseñar la lógica del Problema 8.28 para que imprima un total de departamento cada vez que cambie el departamento (*control de ruptura de segundo orden*) y un total de división cada vez que cambie el número de división (*control de ruptura de primer orden*). El informe debe tener el formato de la Figura 8-8.

```

salario empleado (división 1000, departamento 1)
salario empleado (división 1000, departamento 1)
total departamento 1
salario empleado (división 1000, departamento 2)
salario empleado (división 1000, departamento 2)
total departamento 2
total división 1000
salario empleado (división 2000, departamento 1)
total departamento 1
salario empleado (división 2000, departamento 2)
salario empleado (división 2000, departamento 2)
total departamento 2
total división 2000
gran total

```

Fig. 8-8

```

fija fin con "no"
abre ficheros
fija con cero: total-departamento, total-división, gran-total
lee registro maestro-empleados
    al final: fija fin con "si"
si fin es "no":
    mueve núm-departamento a núm-departamento-anterior
    mueve núm-división a núm-división-anterior
fin-si

ejecuta hasta que fin sea "si" (bucle principal):
    si núm-división es distinto de núm-división-anterior
        (control de ruptura de primer orden)

```

```

ejecuta manejo-ruptura-departamento
ejecuta imprime-total-división
suma total-división a gran-total
mueve cero a total-división
mueve núm-división a núm-división-anterior
en caso contrario si núm-departamento es distinto de núm-departamento-anterior
(control de ruptura de segundo orden)
    ejecuta manejo-ruptura-departamento
fin-si
ejecuta imprime-salario-empleado
suma salario-empleado a total-departamento
lee registro maestro-empleados
    al final: fija fin con "si"
fin-ejecuta
(el fin de fichero debe tratarse como una ruptura de control de primer orden)
ejecuta manejo-ruptura-departamento
ejecuta imprime-total-división
suma total-división a gran-total
ejecuta imprime-gran-total
cierra ficheros
parada

```

El párrafo “maneja-ruptura-departamento” sería el siguiente:

```

manejo-ruptura-departamento
ejecuta imprime-total-departamento
suma total-departamento a total-división
mueve cero a total-departamento
mueve núm-departamento a núm-departamento-anterior

```

A continuación se dan algunos comentarios al algoritmo de doble control de ruptura:

- (1) Tanto el primer como el segundo campo de ruptura deben inicializarse con los valores del primer registro lógico. Esto es sencillo si se utiliza el sistema de lectura inicial.
- (2) La lógica del bucle principal del problema empieza comprobando si se produce ruptura de control de primer orden. Observe que como consecuencia de inicializar el núm-división-anterior con núm-división *no* se puede producir control de ruptura de primer orden en el primer registro del fichero.
- (3) El proceso de control de ruptura de primer orden empieza con el manejo del control de ruptura de segundo orden. Es un error muy común entre principiantes olvidarse del hecho de que cuando el campo de control de primer orden (núm-división) cambia, deben imprimirse los totales del último campo de control de segundo orden (núm-departamento) y también deben inicializarse a cero después dichos totales. Consulte la Figura 8-8 para asegurarse de que comprende el motivo de los anterior.
- (4) El control de ruptura de primer orden, después del manejo del control de ruptura de segundo orden, se encarga de cambiar el valor del primer campo de control. Observe que núm-división-anterior toma el valor de núm-división y que los totales del campo de control se inicializan con cero.
- (5) El bucle principal, después de comprobar el control de ruptura de primer orden, debe pasar a comprobar el control de ruptura de segundo orden. Esto puede llevarse a cabo con una estructura IF lineal. La rutina de control de ruptura de segundo orden se implementa como un párrafo que se ejecuta mediante PERFORM y al que se acude desde las rutinas de control de ruptura de primer y segundo orden.
- (6) La lógica de ambas rutinas de control de ruptura es como sigue: (i) llamada a la siguiente rutina de control de ruptura de nivel más bajo (si existe), (ii) imprime los totales de control de esta ruptura, (iii) suma estos totales de control a los totales del nivel superior, (iv) inicializa este total de control a cero, (v) da al campo de control anterior el valor del campo de control.
- (7) Una vez más, el fin de fichero debe tratarse como una ruptura de primer orden (de otro modo los totales del *último* departamento y de la *última* división no se imprimirían ni se sumarían al gran total).

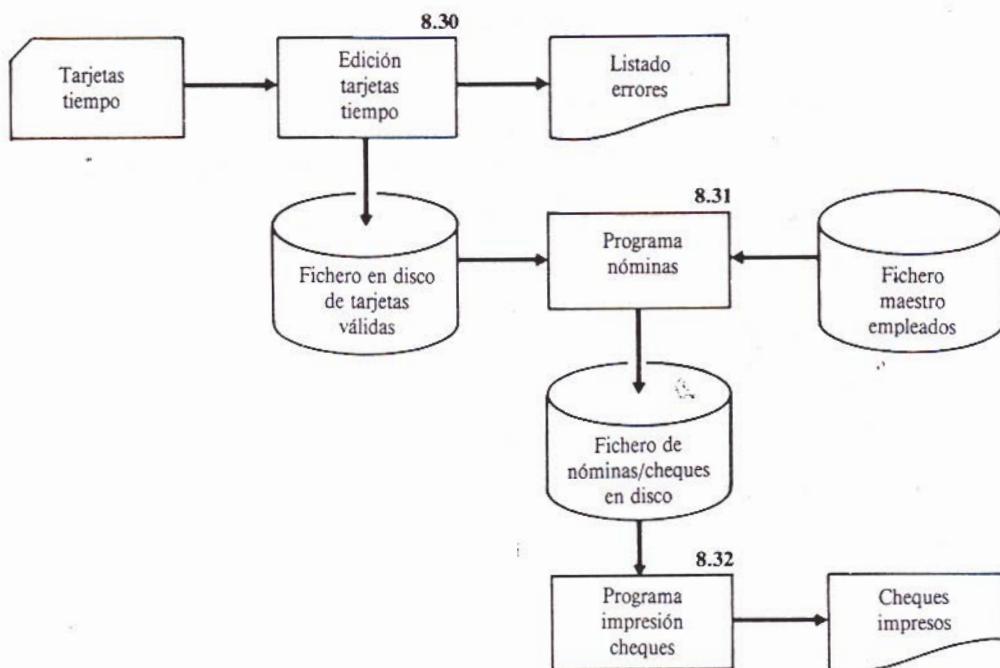


Fig. 8-9

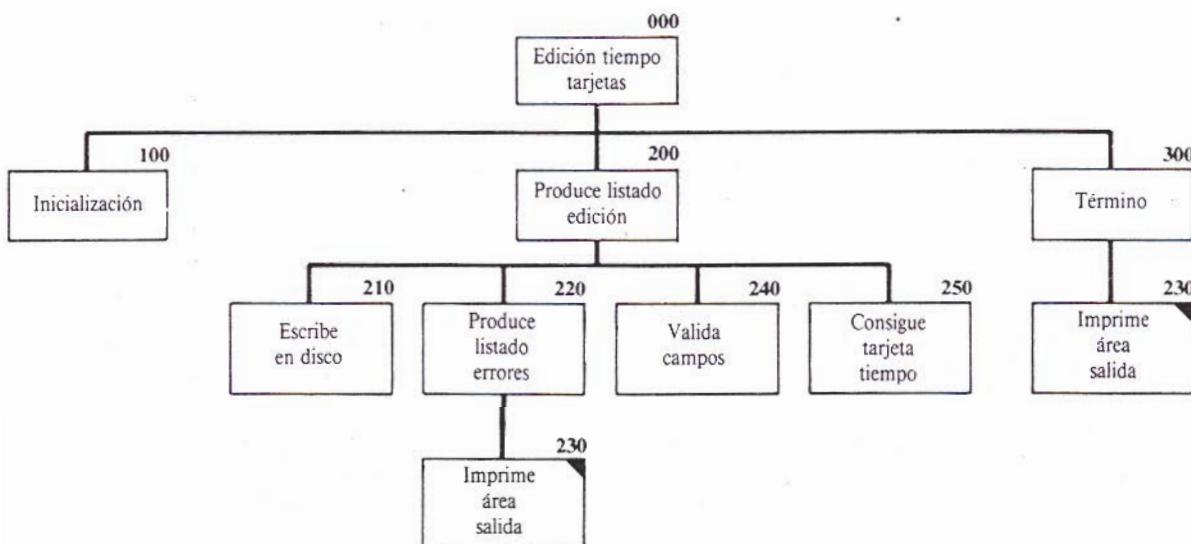


Fig. 8-10

- 8.30 La Figura 8-9 es un *diagrama de flujo* —en el que se ven las relaciones entre el programa y los ficheros— de una aplicación de nóminas (muy simplificada). (Los símbolos utilizados para los ficheros en tarjetas y en disco son evidentes en sí mismos; el “símbolo en forma de piano de cola” simboliza un fichero de impresora.) Diseñe y compruebe el programa que implementa esta aplicación de nóminas.

Como ya se ha visto (Ejemplo 6.35), el objetivo de un programa de edición es validar la información introducida desde un teclado. Específicamente, un programa de edición debe dar respuesta a las siguientes preguntas:

- (1) ¿Son NUMERIC los campos "numéricos"? ¿Son ALPHABETIC los campos "alfabéticos"? etcétera.
- (2) ¿Están los valores dentro del rango apropiado? (Por ejemplo, el día del mes debe estar comprendido entre 1 y 31.)
- (3) ¿Son los valores del registro consistentes entre sí? (Por ejemplo, la fecha de recepción no puede exceder la fecha de envío.)
- (4) ¿Contienen los campos codificados sólo valores apropiados?
- (5) ¿Están los registros lógicos, que supuestamente deben estar ordenados, realmente ordenados? (Esta validación se conoce como *comprobación de secuencia*.)
- (6) ¿Responden los totales de control por lotes a la realidad? [Como los lotes de registros se introducen por teclado, los *totales de control por lotes* se calculan manualmente sumando el contenido de los campos apropiados. Cuando un programa procesa el fichero resultante, también puede acumular el total del mismo campo(s). Si el(es) total(es) de la computadora coinciden con el(es) total(es) de control, se puede asumir que todos los datos han sido procesados.]

Ahora se dan: un organigrama estructurado (Fig. 8-10), un diagrama HIPO (Fig. 8-11), las ejecuciones correspondientes a los primeros y segundos datos de control (Figs. 8-12 y 8-13) y la ejecución final (Fig. 8-14) del programa de edición de nóminas. Debe prestar especial atención al manejo de la paginación *sin* la cláusula LINAGE (véanse los módulos 100-INICIALIZACION y 230-IMPRIME-AREA-SALIDA). En el Problema 11.22 se mejora esta solución.

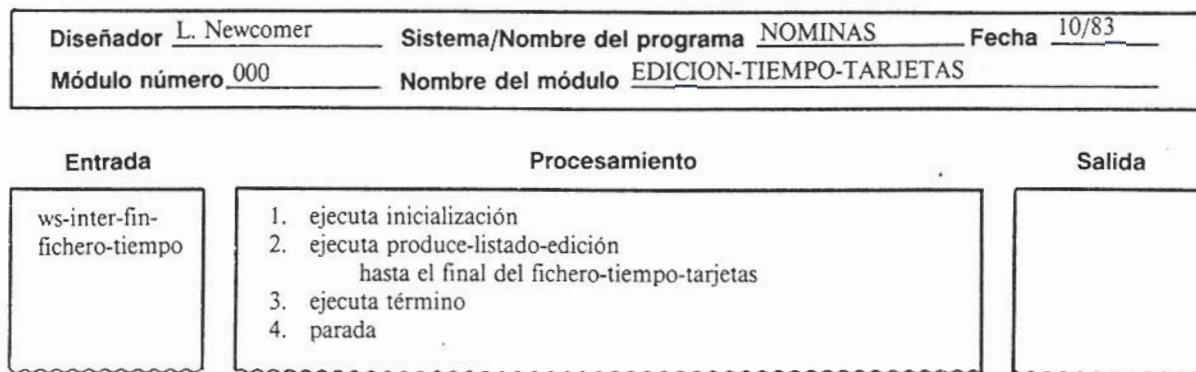


Fig. 8-11. Página 1

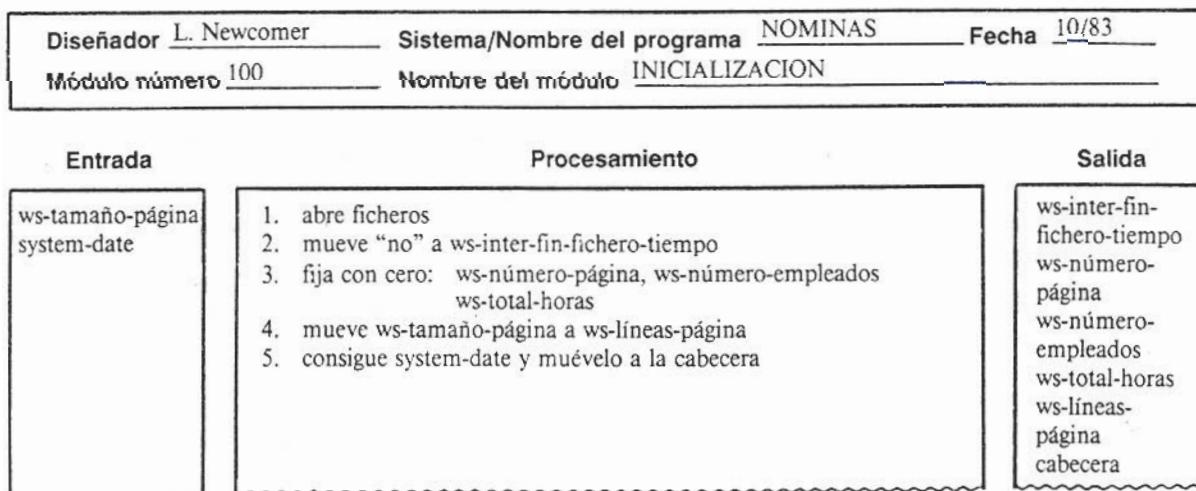


Fig. 8-11. Página 2

Diseñador	L. Newcomer	Sistema/Nombre del programa	NOMINAS	Fecha	10/83
Módulo número	200	Nombre del módulo	PRODUCE-LISTADO-EDICION		

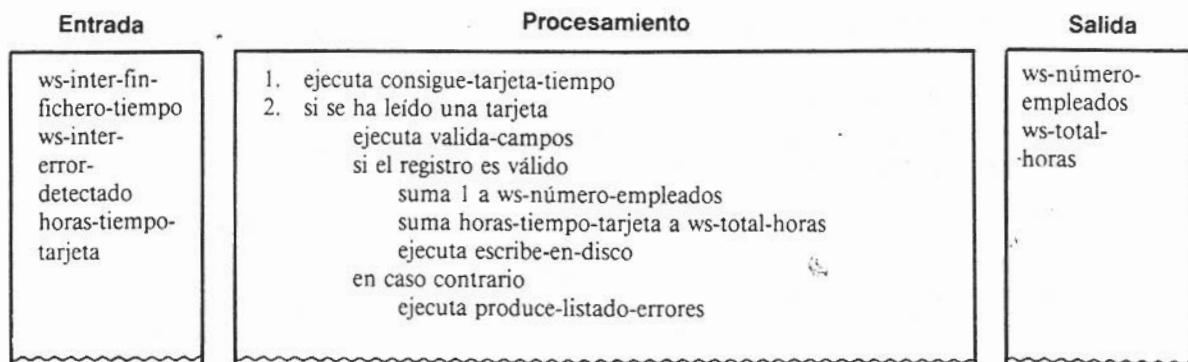


Fig. 8-11. Página 3

Diseñador	L. Newcomer	Sistema/Nombre del programa	NOMINAS	Fecha	10/83
Módulo número	300	Nombre del módulo	TERMINO		

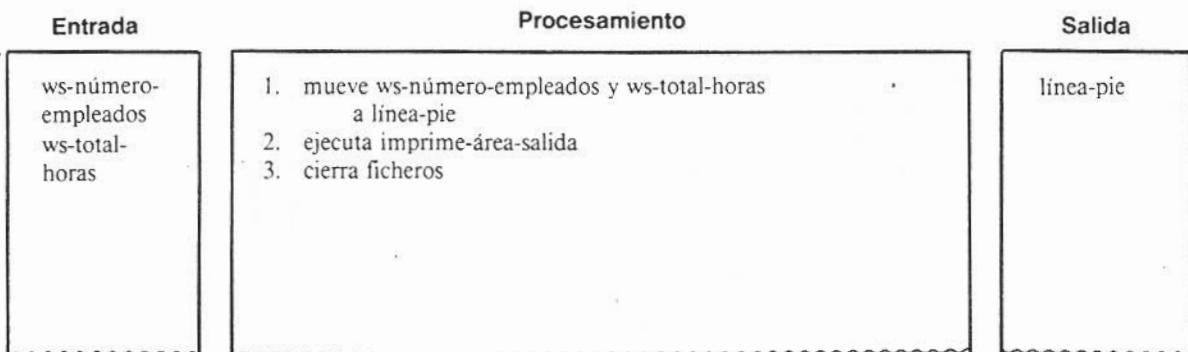


Fig. 8-11. Página 4

Diseñador	L. Newcomer	Sistema/Nombre del programa	NOMINAS	Fecha	10/83
Módulo número	220	Nombre del módulo	PRODUCE-LISTADO-ERRORES		

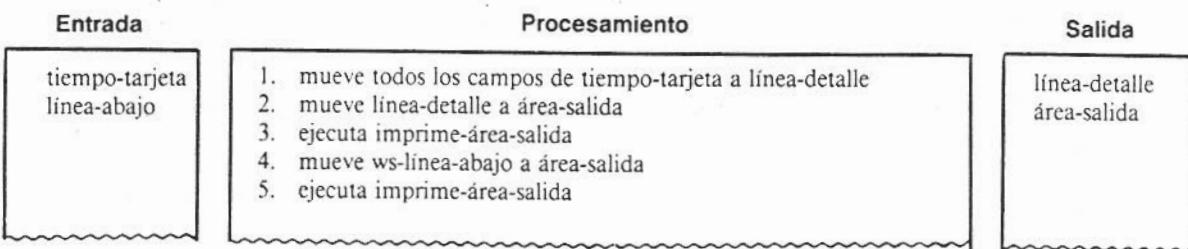


Fig. 8-11. Página 5

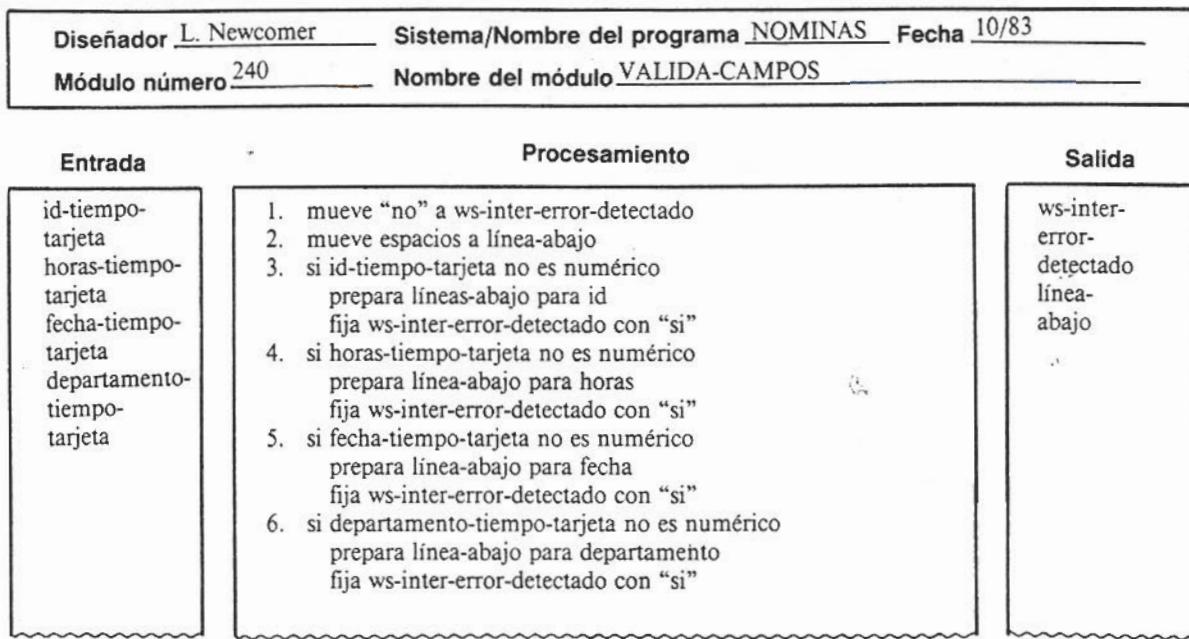


Fig. 8-11. Página 6

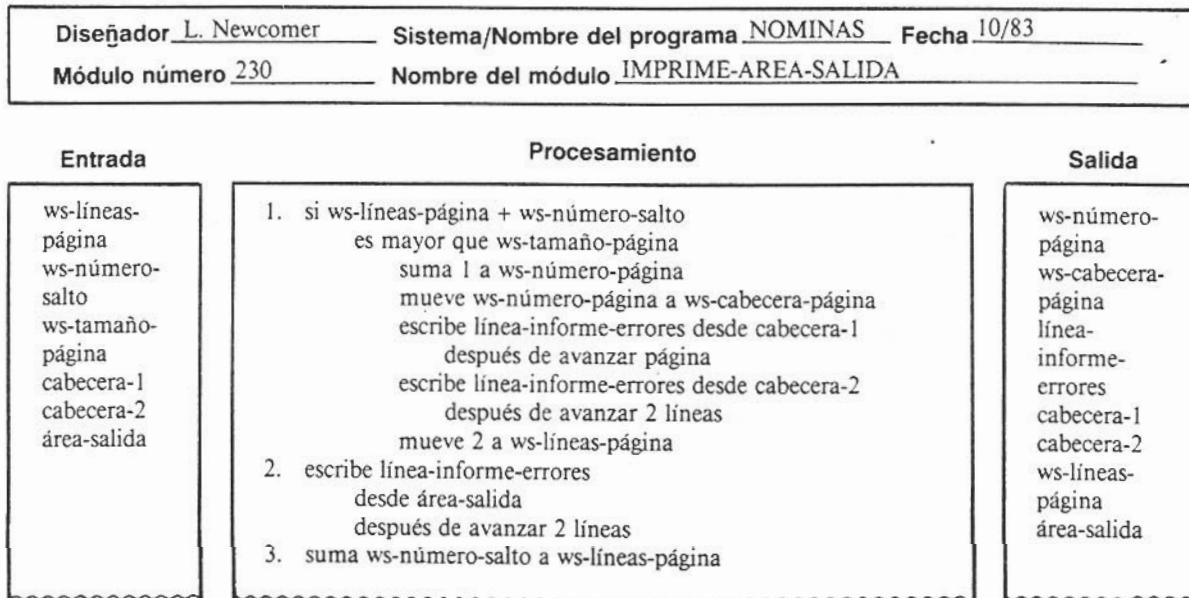


Fig. 8-11. Página 7

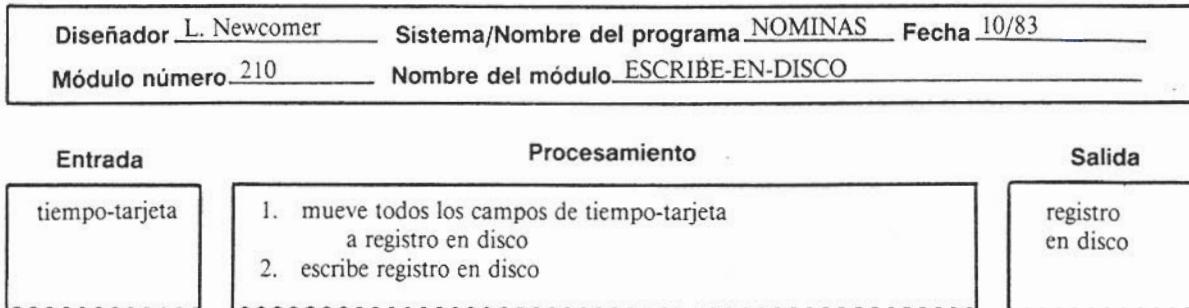


Fig. 8-11. Página 8

Diseñador	L. Newcomer	Sistema/Nombre del programa	NOMINAS	Fecha	10/83
Módulo número	250	Nombre del módulo	CONSIGUE-TARJETA-TIEMPO		

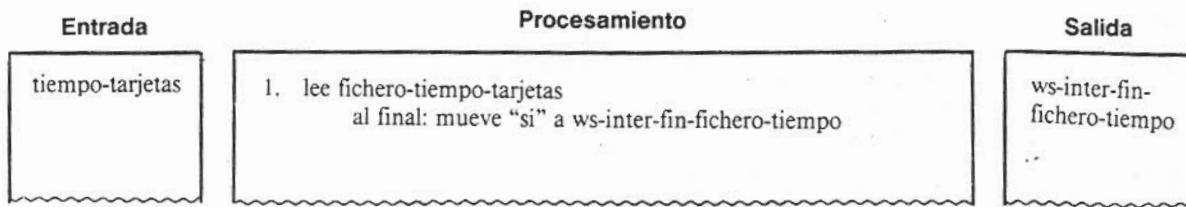


Fig. 8-11. Página 9

```

00001 IDENTIFICATION DIVISION.
00002
00003 PROGRAM-ID. TIMEEDIT.
00004
00005 AUTHOR. LARRY NEWCOMER.
00006 INSTALLATION. PENN STATE UNIVERSITY -- YORK CAMPUS.
00007
00008 ENVIRONMENT DIVISION.
00009
00010 CONFIGURATION SECTION.
00011 SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.
00012 OBJECT-COMPUTER. IBM-3081.
00013
00014 INPUT-OUTPUT SECTION.
00015 FILE-CONTROL.
00016
00017   SELECT FICHERO-TIEMPO-TARJETAS
00018     ASSIGN TO TIEMPOTA
00019     ORGANIZATION IS SEQUENTIAL
00020     ACCESS IS SEQUENTIAL
00021
00022   SELECT FICHERO-TIEMPO-VALIDO
00023     ASSIGN TO TIEMPODI
00024     ORGANIZATION IS SEQUENTIAL
00025     ACCESS IS SEQUENTIAL
00026
00027   SELECT FICHERO-LISTADO-ERRORES
00028     ASSIGN TO ERROENT
00029     ORGANIZATION IS SEQUENTIAL
00030     ACCESS IS SEQUENTIAL
00031
00032 DATA DIVISION.
00033
00034 FILE SECTION.
00035
00036 FD FICHERO-TIEMPO-TARJETAS
00037   RECORD CONTAINS 80 CHARACTERS
00038   LABEL RECORDS ARE OMITTED
00039
00040
00041   01 REGISTRO-TIEMPO-TARJETA.
00042     05 ID-TIEMPO-TARJETA          PIC X(5).
00043     05 HORAS-TIEMPO-TARJETA      PIC 9(2)V9.
00044     05 HORAS-EXTRAS-TIEMPO-TARJETA REDEFINES HORAS-TIEMPO-TARJETA
00045                                         PIC X(3).
00046     05 FECHA-CIERRE-TIEMPO-TARJETA PIC X(6).
00047     05 DEPARTAMENTO-TIEMPO-TARJETA PIC X(4).
00048     05 FILLER                      PIC X(62).
00049

```

Fig. 8-12

```

00050
00051      FD  FICHERO-TIEMPO-VALIDO
00052          BLOCK CONTAINS 0 RECORDS
00053          RECORD CONTAINS 23 CHARACTERS
00054          LABEL RECORDS ARE STANDARD
00055
00056
00057      01  REGISTRO-TIEMPO-VALIDO.
00058          05  ID-TIEMPO-VALIDO          PIC X(5).
00059          05  HORAS-TIEMPO-VALIDO      PIC S9(2)V9    COMP-3.
00060          05  FECHA-CIERRE-TIEMPO-VALIDO  PIC X(6).
00061          05  DEPARTAMENTO-TIEMPO-VALIDO  PIC X(4).
00062          05  FECHA-EDICION-TIEMPO-VALIDO  PIC X(6).
00063
00064      FD  FICHERO-LISTADO-ERRORES
00065          RECORD CONTAINS 132 CHARACTERS
00066          LABEL RECORDS ARE OMITTED
00067
00068
00069      01  LINEA-INFORME-ERRORES        PIC X(132).
00070
00071      WORKING-STORAGE SECTION.
00072
00073      01  INTERRUPTORES-PROGRAMA.
00074          05  WS-INTER-FIN-FICHERO-TIEMPO  PIC X(3).
00075              88  NO-HAY-MAS-REGISTROS   VALUE "SI".
00076              88  REGISTRO-DISPONIBLE    VALUE "NO".
00077          05  WS-INTER-ERROR-DETECTADO  PIC X(3).
00078              88  REGISTRO-VALIDO      VALUE "NO".
00079              88  REGISTRO-INVALIDO    VALUE "SI".
00080
00081      01  CONTADORES-PROGRAMA.
00082          05  WS-NUMERO-PAGINA       PIC S9(3)     COMP-3.
00083          05  WS-NUMERO-EMPLEADOS    PIC S9(5)     COMP-3.
00084          05  WS-NUMERO-SALTO      PIC S9        COMP SYNC.
00085          05  WS-LINEAS-PAGINA     PIC S9(2)     COMP SYNC.
00086
00087      01  AREAS-TOTAL-PROGRAMA.
00088          05  WS-TOTAL-HORAS        PIC S9(7)V9   COMP-3.
00089
00090      01  CONSTANTES-PROGRAMA.
00091          05  WS-TAMAÑO-PAGINA      PIC S9(2)     COMP SYNC.
00092              VALUE +50.
00093
00094      01  WS-CABECERA-LINEA-1.
00095          05  FILLER             PIC X(5)      VALUE SPACES.
00096          05  FILLER             PIC X(15)
00097              VALUE "EDICION TIEMPO TARJETA".
00098          05  WS-CABECERA-MM      PIC Z9.
00099          05  FILLER             PIC X         VALUE "/".
00100          05  WS-CABECERA-DD      PIC 99.
00101          05  FILLER             PIC X         VALUE "/".
00102          05  WS-CABECERA-AA      PIC 99.
00103          05  FILLER             PIC X(3)     VALUE SPACES.
00104          05  FILLER             PIC X(6)     VALUE "PAGINA ".
00105          05  WS-CABECERA-PAGINA  PIC ZZ9.
00106          05  FILLER             PIC X(92)    VALUE SPACES.
00107
00108      01  WS-CABECERA-LINEA-2.
00109          05  FILLER             PIC X(7)      VALUE " ID".
00110          05  FILLER             PIC X(5)      VALUE "HRS".
00111          05  FILLER             PIC X(8)      VALUE " FECHA".
00112          05  FILLER             PIC X(112)   VALUE "DEPT".
00113
00114      01  WS-LINEA-DETALLE.

```

Fig. 8-12 (cont.)

```

00115      05 WS-ID-DETALLE          PIC X(5).
00116      05 FILLER                PIC X(2)      VALUE SPACES.
00117      05 WS-HORAS-DETALLE     PIC X(3).
00118      05 FILLER                PIC X(2).    VALUE SPACES.
00119      05 WS-FECHA-DETALLE    PIC X(6).
00120      05 FILLER                PIC X(2)      VALUE SPACES.
00121      05 WS-DEPARTAMENTO-DETALLE PIC X(4).
00122      05 FILLER                PIC X(108)   VALUE SPACES.
00123
00124      01 WS-LINEA-ABAJO.
00125      05 WS-ID-ABAJO           PIC X(5).
00126      05 FILLER                PIC X(2)      VALUE SPACES.
00127      05 WS-HORAS-ABAJO       PIC X(3).
00128      05 FILLER                PIC X(2)      VALUE SPACES.
00129      05 WS-FECHA-ABAJO      PIC X(6).
00130      05 FILLER                PIC X(2)      VALUE SPACES.
00131      05 WS-DEPARTAMENTO-ABAJO PIC X(4).
00132      05 FILLER                PIC X(108)   VALUE SPACES.
00133
00134      PROCEDURE DIVISION.
00135
00136      000-EDICION-TIEMPO-TARJETAS.
00137
00138      PERFORM 100-INICIALIZACION
00139      PERFORM 200-PRODUCE-LISTADO-EDICION
00140          UNTIL NO-HAY-MAS-REGISTROS
00141      PERFORM 300-TERMINO
00142      STOP RUN
00143
00144
00145      100-INICIALIZACION.
00146
00147      D      DISPLAY "SE ENTRA EN 100"
00148
00149      200-PRODUCE-LISTADO-EDICION.
00150
00151      D      DISPLAY "SE ENTRA EN 200"
00152      D      MOVE "SI" TO WS-INTER-FIN-FICHERO-TIEMPO
00153
00154
00155      300-TERMINO.
00156
00157      D      DISPLAY "SE ENTRA EN 300"
00158

```

Fig. 8-12 (cont.)

```

00153      PROCEDURE DIVISION.
00154
00155      000-EDICION-TIEMPO-TARJETAS
00156
00157      PERFORM 100-INICIALIZACION.
00158      PERFORM 200-PRODUCE-LISTADO-EDICION
00159          UNTIL NO-HAY-MAS-REGISTROS
00160      PERFORM 300-TERMINO
00161      STOP RUN
00162
00163
00164      100-INICIALIZACION.
00165
00166      D      DISPLAY "SE ENTRA EN 100"
00167      OPEN    INPUT   FICHERO-TIEMPO-TARJETAS
00168          OUTPUT  FICHERO-TIEMPO-VALIDO
00169          FICHERO-LISTADO-ERRORES
00170      MOVE "NO" TO      WS-INTER-FIN-FICHERO-TIEMPO
00171      MOVE ZERO TO     WS-NUMERO-PAGINA

```

Fig. 8-13

```

00172                      WS-NUMERO-EMPLEADOS
00173                      WS-TOTAL-HORAS
00174      MOVE WS-TAMAÑO-PAGINA TO      WS-LINEAS-PAGINA
00175      ACCEPT WS-SYSTEM-DATE FROM DATE
00176      MOVE SYSTEM-YY TO          WS-CABECERA-AA
00177      MOVE SYSTEM-MM TO          WS-CABECERA-MM
00178      MOVE SYSTEM-DD TO          WS-CABECERA-DD
00179
00180
00181      200-PRODUCE-LISTADO-EDICION.
00182
00183      D    DISPLAY "SE ENTRA EN 200"
00184      PERFORM 250-CONSIGUE-TARJETA-TIEMPO
00185      IF REGISTRO-DISPONIBLE
00186          PERFORM 240-VALIDA-CAMPOS
00187          IF REGISTRO-VALIDO
00188              PERFORM 210-ESCRIBE-EN-DISCO
00189          ELSE
00190              PERFORM 220-PRODUCE-LISTADO-ERRORES
00191
00192
00193      210-ESCRIBE-EN-DISCO.
00194
00195      D    DISPLAY "SE ENTRA EN 210"
00196
00197
00198      220-PRODUCE-LISTADO-ERRORES.
00199
00200      D    DISPLAY "SE ENTRA EN 220"
00201
00202
00203      230-IMPRIME-AREA-SALIDA.
00204
00205      D    DISPLAY "SE ENTRA EN 230"
00206
00207
00208      240-VALIDA-CAMPOS.
00209
00210      D    DISPLAY "SE ENTRA EN 240"
00211      D    ON 1 AND EVERY 2
00212          MOVE "SI" TO WS-INTER-ERROR-DETECTADO
00213      D    ELSE
00214          MOVE "NO" TO WS-INTER-ERROR-DETECTADO
00215
00216
00217      250-CONSIGUE-TARJETA-TIEMPO.
00218
00219          READ FICHERO-TIEMPO-TARJETAS
00220          AT END
00221              MOVE "SI" TO WS-INTER-FIN-FICHERO-TIEMPO
00222
00223
00224      300-TERMINO.
00225
00226      D    DISPLAY "SE ENTRA EN 300"
00227          MOVE WS-NUMERO-EMPLEADOS TO WS-CONT-PIE
00228          MOVE WS-TOTAL-HORAS TO WS-TOTAL-PIE
00229          MOVE WS-LINEA-PIE TO WS-AREA-SALIDA
00230          MOVE WS-SALTO-ANTES-PIE TO WS-NUMERO-SALTO
00231          MOVE WS-LINEA-PIE TO WS-AREA-SALIDA
00232          PERFORM 230-IMPRIME-AREA-SALIDA
00233
00234          CLOSE   FICHERO-TIEMPO-TARJETAS
00235              FICHERO-TIEMPO-VALIDO
00236              FICHERO-LISTADO-ERRORES
00237

```

Fig. 8-13 (cont.)

```

00001 IDENTIFICATION DIVISION.
00002
00003 PROGRAM-ID. TIMEEDIT.
00004
00005 AUTHOR. LARRY NEWCOMER.
00006 INSTALLATION. PENN STATE UNIVERSITY -- YORK CAMPUS.
00007
00008 ENVIRONMENT DIVISION.
00009
00010 CONFIGURARION SECTION.
00011 SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.
00012 OBJECT-COMPUTER. IBM-3081.
00013
00014 INPUT-OUTPUT SECTION.
00015 FILE-CONTROL.
00016
00017   SELECT FICHERO-TIEMPO-TARJETAS
00018     ASSIGN TO TIEMPOTA
00019       ORGANIZATION IS SEQUENTIAL
00020       ACCESS IS SEQUENTIAL
00021
00022   SELECT FICHERO-TIEMPO-VALIDO
00023     ASSIGN TO TIEMPODI
00024       ORGANIZATION IS SEQUENTIAL
00025       ACCESS IS SEQUENTIAL
00026
00027   SELECT FICHERO-LISTADO-ERRORES
00028     ASSIGN TO ERRORENT
00029       ORGANIZATION IS SEQUENTIAL
00030       ACCESS IS SEQUENTIAL
00031
00032 DATA DIVISION.
00033
00034 FILE SECTION.
00035
00036
00037 FD FICHERO-TIEMPO-TARJETAS
00038   RECORD CONTAINS 80 CHARACTERS
00039   LABEL RECORDS ARE OMITTED
00040
00041
00042   01 REGISTRO-TIEMPO-TARJETA.
00043     05 ID-TIEMPO-TARJETA          PIC X(5).
00044     05 HORAS-TIEMPO-TARJETA      PIC 9(2)V9.
00045     05 HORAS-EXTRAS-TIEMPO-TARJETA REDEFINES HORAS-TIEMPO-TARJETA
00046                           PIC X(3).
00047     05 FECHA-CIERRE-TIEMPO-TARJETA PIC X(6).
00048     05 DEPARTAMENTO-TIEMPO-TARJETA PIC X(4).
00049     05 FILLER                   PIC X(62).
00050
00051 FD FICHERO-TIEMPO-VALIDO
00052   BLOCK CONTAINS 0 RECORDS
00053   RECORD CONTAINS 23 CHARACTERS
00054   LABEL RECORDS ARE STANDARD
00055
00056
00057   01 REGISTRO-TIEMPO-VALIDO.
00058     05 ID-TIEMPO-VALIDO          PIC X(5).
00059     05 HORAS-TIEMPO-VALIDO      PIC S9(2)V9  COMP-3.
00060     05 FECHA-CIERRE-TIEMPO-VALIDO PIC X(6).
00061     05 DEPARTAMENTO-TIEMPO-VALIDO PIC X(4).
00062     05 FECHA-EDICION-TIEMPO-VALIDO PIC X(6).
00063
00064 FD FICHERO-LISTADO-ERRORES
00065   RECORD CONTAINS 132 CHARACTERS
00066   LABEL RECORDS ARE OMITTED
00067

```

Fig. 8-14

```

00068      01 LINEA-INFORME-ERRORES      PIC X(132).
00069
00070
00071      WORKING-STORAGE SECTION.
00072
00073      01 INTERRUPTORES-PROGRAMA.
00074          05 WS-INTER-FIN-FICHERO-TIEMPO  PIC X(3).
00075              88 NO-HAY-MAS-REGISTROS  VALUE "SI".
00076              88 REGISTRO-DISPONIBLE  VALUE "NO".
00077          05 WS-INTER-ERROR-DETECTADO  PIC X(3).
00078              88 REGISTRO-VALIDO    VALUE "NO ".
00079              88 REGISTRO-INVALIDO  VALUE "SI".
00080
00081      01 CONTADORES-PROGRAMA.
00082          05 WS-NUMERO-PAGINA      PIC S9(3)   COMP-3.
00083          05 WS-NUMERO-EMPLEADOS  PIC S9(5)   COMP-3.
00084          05 WS-NUMERO-SALTO     PIC S9      COMP SYNC.
00085          05 WS-LINEAS-PAGINA    PIC S9(2)   COMP SYNC.
00086
00087      01 AREAS-TOTAL-PROGRAMA.
00088          05 WS-TOTAL-HORAS      PIC S9(7)V9  COMP-3.
00089
00090      01 CONSTANTES-PROGRAMA.
00091          05 WS-TAMAÑO-PAGINA    PIC S9(2)   COMP SYNC
00092              VALUE +50.
00093          05 WS-SALTO-ANTES-CABECERA  PIC S9(2)   COMP SYNC
00094              VALUE +2.
00095          05 WS-SALTO-ANTES-PIE    PIC S9(2)   COMP SYNC S.
00096              VALUE +4.
00097          05 WS-SALTO-ANTES-DETALLE  PIC S9(2)   COMP SYNC
00098              VALUE +2.
00099          05 WS-SALTO-ANTES-ABAJO    PIC S9(2)   COMP SYNC
00100              VALUE +1.

00159      PROCEDURE DIVISION.
00160
00161      000-EDICION-TIEMPO-TARJETAS.
00162
00163          PERFORM 100-INICIALIZACION
00164          PERFORM 200-PRODUCE-LISTADO-EDICION
00165              UNTIL NO-HAY-MAS-REGISTROS
00166          PERFORM 300-TERMINO
00167          STOP RUN
00168
00169
00170      100-INICIALIZACION.
00171
00172      D  DISPLAY "SE ENTRA EN 100"
00173          OPEN INPUT FICHERO-TIEMPO-TARJETAS
00174              OUTPUT FICHERO-TIEMPO-VALIDO
00175                  FICHERO-LISTADO-ERRORES
00176          MOVE "NO" TO                WS-INTER-FIN-FICHERO-TIEMPO
00177          MOVE ZERO TO               WS-NUMERO-PAGINA
00178                               WS-NUMERO-EMPLEADOS
00179                               WS-TOTAL-HORAS
00180          MOVE WS-TAMAÑO-PAGINA TO  WS-LINEAS-PAGINA
00181          ACCEPT WS-SYSTEM-DATE FROM DATE
00182          MOVE SYSTEM-YY TO        WS-CABECERA-AA
00183          MOVE SYSTEM-MM TO        WS-CABECERA-MM
00184          MOVE SYSTEM-DD TO        WS-CABECERA-DD
00185
00186
00187      200-PRODUCE-LISTADO-EDICION.
00188
00189      D  DISPLAY "SE ENTRA EN 200"
00190          PERFORM 250-CONSIGUE-TARJETA-TIEMPO

```

Fig. 8-14 (cont.)

```

00191      IF REGISTRO-DISPONIBLE
00192          PERFORM 240-VALIDA-CAMPOS
00193          IF REGISTRO-VALIDO
00194              ADD 1                      TO WS-NUMERO-EMPLEADOS
00195              ADD HORAS-TIEMPO-TARJETA TO WS-TOTAL-HORAS
00196              PERFORM 210-ESCRIBE-EN-DISCO
00197          ELSE
00198              PERFORM 220-PRODUCE-LISTADO-ERRORES
00199
00200
00201      210-ESCRIBE-EN-DISCO.
00202
00203      D    DISPLAY "SE ENTRA EN 210"
00204          MOVE ID-TIEMPO-TARJETA      TO ID-TIEMPO-VALIDO
00205          MOVE HORAS-TIEMPO-TARJETA  TO HORAS-TIEMPO-VALIDO
00206          MOVE FECHA-CIERRE-TIEMPO-TARJETA TO FECHA-CIERRRE-TIEMPO-VALIDO
00207          MOVE DEPARTAMENTO-TIEMPO-TARJETA TO DEPARTAMENTO-TIEMPO-VALIDO
00208          MOVE WS-SYSTEM-DATE        TO FECHA-EDICION-TIEMPO-VALIDO
00209          WRITE REGISTRO-TIEMPO-VALIDO
00210
00211
00212      220-PRODUCE-LISTADO-ERRORES..
00213
00214      D    DISPLAY "SE ENTRA EN 220"
00215          MOVE ID-TIEMPO-TARJETA      TO WS-ID-DETALLE
00216          MOVE HORAS-EXTRAS-TIEMPO-TARJETA TO WS-HORAS-DETALLE
00217          MOVE FECHA-CIERRE-TIEMPO-TARJETA TO WS-FECHA-DETALLE
00218          MOVE DEPARTAMENTO-TIEMPO-TARJETA TO WS-DEPARTAMENTO-DETALLE
00219
00220          MOVE WS-LINEA-DETALLE       TO WS-AREA-SALIDA
00221          MOVE WS-SALTO-ANTES-DETALLE TO WS-NUMERO-SALTO
00222          PERFORM 230-IMPRIME-AREA-SALIDA
00223
00224          MOVE WS-LINEA-ABAJO         TO WS-AREA-SALIDA
00225          MOVE WS-SALTO-ANTES-ABAJO   TO WS-NUMERO-SALTO
00226          PERFORM 230-IMPRIME-AREA-SALIDA
00227
00228
00229      230-IMPRIME-AREA-SALIDA.
00230
00231      D    DISPLAY "SE ENTRA EN 230"
00232          IF WS-LINEAS-PAGINA + WS-NUMERO-SALTO
00233              GREATER THAN WS-TAMAÑO-PAGINA
00234                  ADD 1 TO WS-NUMERO-PAGINA
00235                  MOVE WS-NUMERO-PAGINA TO WS-CABECERA-PAGINA
00236                  WRITE LINEA-INFORME-ERRORES
00237                      FROM WS-CABECERA-LINEA-1
00238                      AFTER ADVANCING PAGE
00239                  WRITE LINEA-INFORME-ERRORES
00240                      FROM WS-CABECERA-LINEA-2
00241                      AFTER ADVANCING WS-SALTO-ANTES-CABECERA
00242                  MOVE WS-SALTO-ANTES-CABECERA TO WS-LINEAS-PAGINA
00243
00244                  WRITE LINEA-INFORME-ERRORES
00245                      FROM WS-AREA-SALIDA
00246                      AFTER ADVANCING WS-NUMERO-SALTO LINES
00247                      ADD WS-NUMERO-SALTO TO WS-LINEAS-PAGINA
00248
00249
00250      240-VALIDA-CAMPOS.
00251
00252      D    DISPLAY "SE ENTRA EN 240"
00253          MOVE "NO"    TO WS-INTER-ERROR-DETECTADO
00254          MOVE SPACES TO WS-LINEA-ABAJO
00255          IF ID-TIEMPO-TARJETA NOT NUMERIC
00256              MOVE ALL "-" TO WS-ID-ABAJO

```

Fig. 8-14 (cont.)

```

00257      MOVE "SI"      TO WS-INTER-ERROR-DETECTADO
00258
00259      IF HORAS-TIEMPO-TARJETA NOT NUMERIC
00260          MOVE ALL "-" TO WS-HORAS-ABAJO
00261          MOVE "SI"      TO WS-INTER-ERROR-DETECTADO
00262
00263      IF FECHA-CIERRE-TIEMPO-TARJETA NOT NUMERIC
00264          MOVE ALL "-" TO WS-HORAS-ABAJO
00265          MOVE "SI"      TO WS-INTER-ERROR-DETECTADO
00266
00267      IF DEPARTAMENTO-TIEMPO-TARJETA NOT NUMERIC
00268          MOVE ALL "-" TO WS-DEPARTAMENTO-ABAJO
00269          MOVE "SI"      TO WS-INTER-ERROR-DETECTADO
00270
00271
00272      250-CONSIGUE-TARJETA-TIEMPO.
00273
00274      READ FICHERO-TIEMPO-TARJETAS
00275          AT END
00276              MOVE "SI" TO WS-INTER-FIN-FICHERO-TIEMPO
00277
00278
00279      300-TERMINO.
00280
00281      D      DISPLAY "SE ENTRA EN 300"
00282          MOVE WS-NUMERO-EMPLEADOS TO WS-CONT-PIE
00283          MOVE WS-TOTAL-HORAS    TO WS-TOTAL-PIE
00284          MOVE WS-LINEA-PIE      TO WS-AREA-SALIDA
00285          MOVE WS-SALTO-ANTES-PIE TO WS-NUMERO-SALIDA
00286          MOVE WS-LINEA-PIE      TO WS-AREA-SALIDA
00287          PERFORM 230-IMPRIME-AREA-SALIDA
00288
00289          CLOSE  FICHERO-TIEMPO-TARJETAS
00290              FICHERO-TIEMPO-VALIDO
00291              FICHERO-LISTADO-ERRORES
00292

```

EDICION TIEMPO TARJETA 5/09/83 PAGINA 1

ID	HRS	FECHA	DEPT
1 111	111	111111	1111
-----			
11 11	111	111111	1111
-----			
111 1	111	111111	1111
-----			
11111	11	111111	1111
-----			
11111	1 1	111111	1111
-----			

11111 1 1 111111 11 1

-----

EDICION TIEMPO TARJETA 5/09/83 PAGINA 2

ID	HRS	FECHA	DEPT
11111	111	111111	111
-----			

----- ----- -----

# EMPLEADOS= 2 TOTAL HORAS= 33.3

Fig. 8-14 (cont.)

El Problema 11.22 mejora la solución de arriba.

**8.31** Construya un organigrama estructurado para el programa de nóminas del Problema 8.30.

Véase la Figura 8-15. El corazón del algoritmo reside en el módulo que compara los registros lógicos tarjeta-tiempo-válido (obtenidos del fichero en disco producido por el programa del Problema 8.30) con los registros lógicos del fichero-maestro-empleados correspondientes, donde se obtiene el salario-hora del empleado. Asumiendo que ambos ficheros están clasificados por orden creciente de ID-empleado, el diagrama HIPO podría ser como el de la Figura 8-16. (Véase el Problema 11.23 para el programa COBOL completo.)

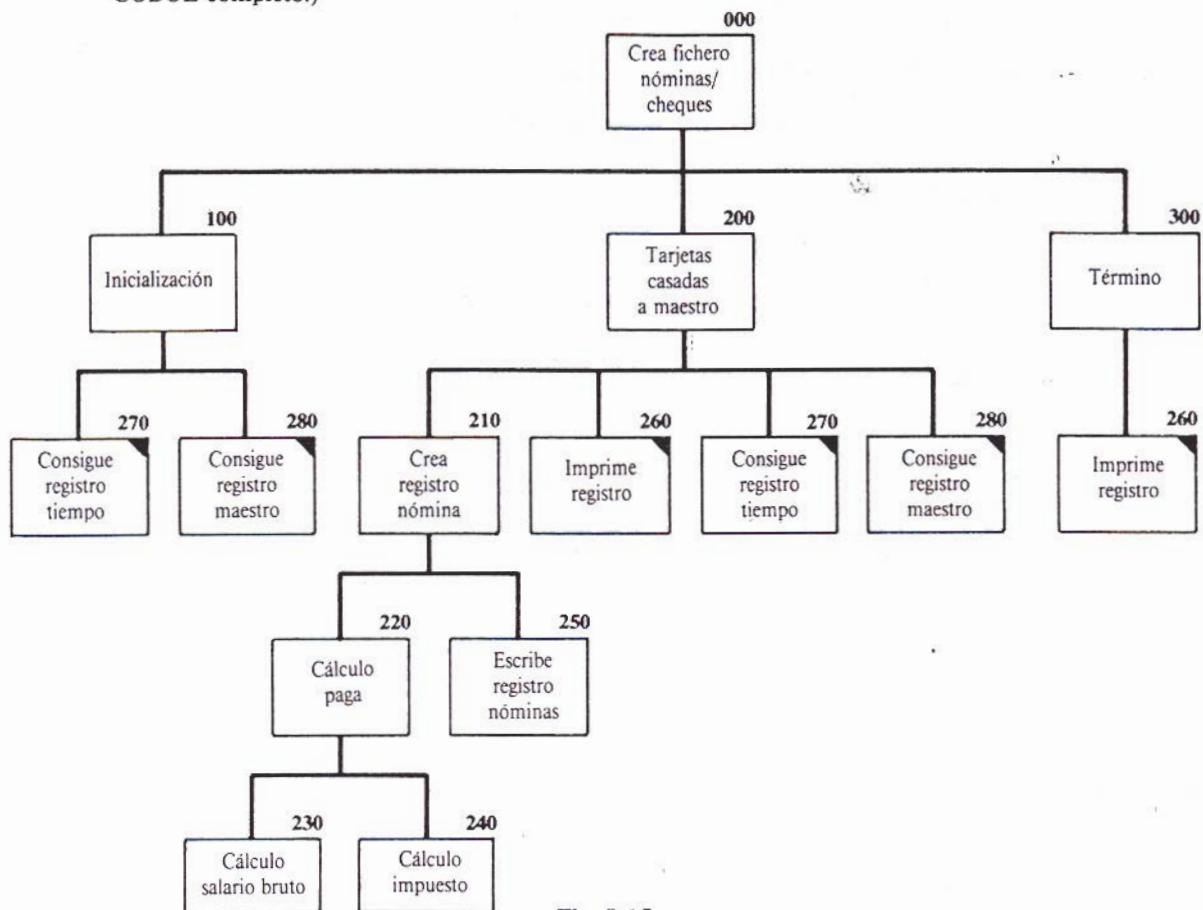


Fig. 8-15

Diseñador L. Newcomer	Sistema/Nombre del programa NOMINAS	Fecha 10/83
Módulo número 200	Nombre del módulo CASA-TIEMPO-TARJ-Y-MAESTRO	
Entrada	Procesamiento	Salida
id-tiempo id-maestro	1. fija línea-detalle con espacios 2. si id-tiempo = id-maestro ejecuta 210-crea-registro-nómina ejecuta 270-consigue-registro-tiempo ejecuta 280-consigue-registro-maestro en caso contrario si id-tiempo < id-maestro formatea error "registro sin casar" ejecuta 270-consigue-registro-tiempo en caso contrario formatea mensaje "empleado inactivo" ( <i>no casa con el maestro</i> ) ejecuta 280-consigue-registro-maestro fin-si 3. ejecuta 260-imprime-registro	línea- detalle área- mensajes

Fig. 8-16

- 8.32 Como continuación de la aplicación de nóminas de los Problemas 8.30 y 8.31, construya un organigrama estructurado de la impresión de los cheques correspondiente.

Véase la Figura 8-17. En el párrafo COBOL 100-INICIALIZACION, la instrucción ACCEPT se utilizará para obtener la fecha del día y el número del primer cheque a imprimir; los números siguientes serán consecutivos. (El Ejemplo 12.10 contiene el programa COBOL completo.)

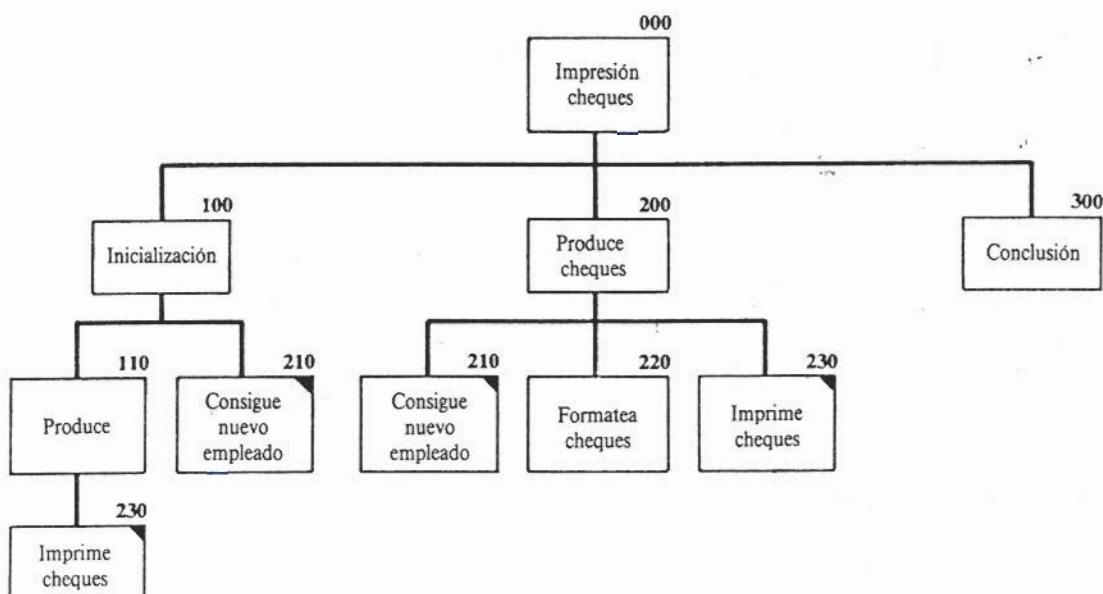


Fig. 8-17

- 8.33 Con relación al Problema 8.30, escriba el pseudocódigo para una *comprobación secuencial* que se asegure de que los registros contenidos en las tarjetas están en orden creciente de ID-empleados (y en este caso coordinados además con los registros del fichero maestro de empleados, como se requiere en el Problema 8.31).

fija fin-de-fichero con "no"  
 fija campo-control-anterior con valor-inferior  
 lee fichero-entrada  
 al final: fija fin-de-fichero con "si"  
 ejecuta hasta que fin-de-fichero sea "si"  
 si campo-control es mayor que campo-control-anterior  
 (procesamiento de un registro en orden)  
 fija campo-control-anterior con campo-control  
 en caso contrario  
 (registro fuera de orden)  
 ejecuta rutina-registro-desordenado  
 fin-si  
 lee fichero-entrada  
 al final: fija fin-de-fichero con "si"  
 fin-ejecuta.

Obsérvese que la inicialización del campo-control-anterior con valor-inferior garantiza que el valor del campo-control-anterior es inferior que el de campo-control del primer registro. Del mismo modo, es de notar que cada vez que se procesa un registro que está en orden, el campo-control-anterior toma el valor del campo-control. Esto prepara el campo-control-anterior para su comparación con el *siguiente* registro lógico del fichero.

# Capítulo 9

## Depuración

La *depuración* es el proceso de identificar las causas de los errores de un programa y corregirlos después. Hay tres tipos básicos de errores en los programas:

**Errores sintácticos**, también llamados errores gramaticales en el uso del lenguaje de programación (COBOL). Son detectados por el compilador al intentar traducir el programa en COBOL al lenguaje máquina.

**Errores lógicos graves**, que hacen abortar el programa ABEND. Son detectados por el equipo o el sistema operativo durante la ejecución de prueba (con los errores sintácticos totalmente corregidos), al encontrarse con alguna instrucción en lenguaje máquina que no puede ejecutarse.

**Errores lógicos leves**, en los que el programa se ejecuta íntegramente hasta la instrucción STOP RUN, pero los resultados no son correctos. Es preciso eliminar del programa todos los errores sintácticos y lógicos graves para que puedan apreciarse los errores lógicos leves.

### 9.1 DEPURACION DE ERRORES SINTACTICOS

Los errores sintácticos son los más fáciles de eliminar porque el compilador COBOL imprime, al detectarlos, mensajes de diagnóstico que identifican los errores y las instrucciones donde se encuentran. Estos mensajes aparecen por lo general en la parte inferior del listado fuente; sólo hay que corregirlos para que todas las instrucciones estén conforme a las reglas de COBOL.

**EJEMPLO 9.1** La Figura 9-1 es un listado fuente con algunos errores sintácticos y sus respectivos mensajes de diagnóstico. Cada mensaje indica el número dado por el compilador a la instrucción, un código de identificación del error y una descripción en inglés del error. Después se comentan algunos de los errores que aparecen en el programa.

```
00001      IDENTIFICATION DIVISION.  
00002  
00003      PROGRAM-ID.  SYNTAX.  
00004  
00005      AUTHOR.  LARRY NEWCOMER.  
00006      INSTALLATION.  PENN STATE UNIVERSITY -- YORK CAMPUS.  
00007  
00008      ENVIRONMENT DIVISION.  
00009  
00010      CONFIGURATION SECTION.  
00011      SOURCE-COMPUTER.  IBM-3081  
00012      OBJECT-COMPUTER.  IBM-3081.  
00013  
00014      INPUT-OUTPUT SECTION.  
00015      FILE-CONTROL.  
00016  
00017      SELECT TIME-CORD-FILE  
00018          ASSIGN TO TIMECARD  
00019          ORGANIZATION IS SEQUENTIAL  
00020          ACCESS IS SEQUENTIAL  
00021  
00022      SELECT VALID-TIME-FILE  
00023          ASSIGN TO DISKTIME  
00024          ORGANIZATION IS SEQUENTIAL  
00025          ACCESS IS SEQUENTIAL  
00026  
00027      SELECT ERROR-LISTING-FILE  
00028          ASSIGN TO ERRORLOG  
00029          ORGANIZATION IS SEQUENTIAL  
00030          ACCESS IS SEQUENTIAL  
00031  
00032
```

Fig. 9-1

```

00033      DATA DIVISION.
00034
00035      FILE SECTION.
00036
00037      FD TIME-CARD-FILE
00038          RECORD CONTAINS 80 CHARACTERS
00039          LABEL RECORDS ARE OMITTED
00040
00041
00042      01 TIME-CARD-RECORD.
00043          05 TIME-CARD-ID          PIC S(5).
00044          05 TIME-CARD-HOURS    PIC 9(2)V9.
00045          05 TIME-CARD-X-HOURS  REDEFINES TIME-CARD-HOURS
00046                  PIC X(3).
00047          05 TIME-CARD-CLOSING-DATE PIC X(6).
00048          05 TIME-CARD-DEPARTMENT  PIC X(4).
00049          05 FILLER             PIC X(62).
00050
00051      FD VALID-TIME-FILE
00052          BLOCK CONTAINS 0 RECORDS
00053          RECORD CONTAINS 23 CHARACTERS
00054          LABEL RECORDS ARE STANDARD
00055
00056      01 VALID-TIME-RECORD.
00057          05 VALID-TIME-ID        PIC X(5).
00058          05 VALID-TIME-HOURS    PIC S9(2)V9  COMP-3.
00059          05 VALID-TIME-CLOSING-DATE PIC X(6).
00060          05 VALID-TIME-DEPARTMENT  PIC X(4).
00061          05 VALID-TIME-EDIT-DATE   PIC X(6).
00062
00063      FD ERROR-LISTING-FILE
00064          RECORD CONTAINS 132 CHARACTERS
00065          LABEL RECORDS ARE OMITTED
00066
00067
00068      01 ERROR-REPORT-LINE      PIC X(132).
00069
00070      WORKING-STORAGE SECTION.
00071

.
.

00106      01 WS-HEADING-LINE-1.
00107          05 FILLER             PIC X(5)      VALUE SPACES.
00108          05 FILLER             PIC X(15)
00109          05 FILLER             VALUE "TIME CARD EDIT".
00110          05 WS-HEADING-MM      PIC Z9.
00111          05 FILLER             PIC X      VALUE "//".
00112          05 WS-HEADING-DD      PIC 99.
00113          05 FILLER             PIC X      VALUE "/".
00114          05 WS-HEADING-YY      PIC 99.
00115          05 FILLER             PIC X(3)      VALUE SPACES.
00116          05 FILLER             PIC X(5)      VALUE "PAGE ".
00117          05 WS-HEADING-PAGE    PIC ZZ9.
00118          05 FILLER             PIC X(93)     VALUE SPACES.
00135
00136      01 WS-UNDER-LINE.
00137          05 WS-UNDER-ID        PIC X(5).
00138          05 FILLER             PIC X(2)      VALUE SPICES.
00139          05 WS-UNDER-HOURS    PIC X(3).
00140          05 FILLER             PIC X(2)      VALUE SPACES.
00141          05 WS-UNDER-DATE      PIC X(6).
00142          05 FILLER             PIC X(2)      VALUE SPACES.
00143          05 WS-UNDER-DEPARTMENT PIC X(4).
00144          05 FILLER             PIC X(108)    VALUE SPACES.

.
.

00158      PROCEDURE DIVISION.
00159
00160      000-EDIT-TIME-CARDS.
00161
00162          PERFORM 100-INITIALIZE
00163          PERFORM 200-PRODUCE-EDIT-LISTING
00164              UNTIL NO-MORE-RECORDS.
00165          PERFORM 300-TERMINATE
00166          STOP RUN
00167
00168
00169      100-INITIALIZE.
00170
00171      D      DISPLAY "100 ENTERED"
00172          OPEN    INPUT  TIME-CARD-FILE
00173              OUTPUT VALID-TIME-FILE
00174                  ERROR-LISTING-FILE
00175          MOVE "NO" TO      WS-END-TIME-FILE-SW
00176          MOVE ZERO TO     WS-PAGE-NUMBER
00177          MOVE WS-PAGE-SIZE TO WS-NUMBER-EMPLOYEES
00178          MOVE WS-TOTAL-HOURS TO WS-LINES-ON-PAGE
00179          ACCEPT WS-SYSTEM-DATE FROM DATE
00180          MOVE SYSTEM-YY TO   WS-HEADING-YY

```

Fig. 9-1 (cont.)

```

00182      MOVE SYSTEM-MM TO          WS-HEADING-MM
00183      MOVE SYSTEM-DD TO          WS-HEADING-DD
00184
00185
00186      200-PRODUCE-EDIT-LISTING.
00187
00188      D    DISPLAY "200 ENTERED"
00189      PERFORM 250-GET-TIME-CARD
00190      IF RECORD-AVAILABLE
00191          PERFORM 240 VALIDATE-FIELDS
00192          IF VALID-RECORD EQUAL "YES"
00193              ADD 1           TO WS-NUMBER-EMPLOYEES
00194              ADD TIME-CARD-HOURS TO WS-TOTAL-HOURS
00195              PERFORM 210-WRITE-TO-DISK
00196      ELSE
00197          PERFORM 220-PRODUCE-ERROR-LISTING
00198
00199
00200      210-WRITE-TO-DISK.
00201
00202      D    DISPLAY "210 ENTERED"
00203      MOVE TIME-CARD-ID          TO VALID-TIME-ID
00204      MOVE TIME-CARD-HOURS        TO VALID-TIME-HOURS
00205      MOVE TIME-CARD-CLOSING-DATE TO VALID-TIME-CLOSING-DATE
00206      MOVE TIME-CARD-DEPARTMENT    TO VALID-TIME-DEPARTMENT
00207      MOVE WS-SYSTEM-DATE         TO VALID-TIME-EDIT-DATE
00208      WRITE VALID-TIME-FILE
00209

.
.

CARD  ERROR MESSAGE
12   IKF1043I-W    END OF SENTENCE SHOULD PRECEDE OBJECT-COMPUTER . ASSUMED PRESENT.
40   IKF1056I-E    FILE-NAME NOT DEFINED IN A SELECT. DESCRIPTION IGNORED.
56   IKF1043I-W    END OF SENTENCE SHOULD PRECEDE 01 . ASSUMED PRESENT.
111  IKF2126I-C    VALUE CLAUSE LITERAL TOO LONG. TRUNCATED TO PICTURE SIZE.
138  IKF1017I-E    SPICES INVALID IN VALUE CLAUSE. SKIPPING TO NEXT CLAUSE.
172  IKF3001I-E    TIME-CARD-FILE NOT DEFINED. DELETING TILL LEGAL ELEMENT FOUND.
191  IKF3001I-E    240 NOT DEFINED. STATEMENT DISCARDED.
191  IKF3001I-E    VALIDATE-FIELDS NOT DEFINED.
192  IKF4030I-E    FOUND ' EQUAL ' AFTER CONDITION. EXPECT 'OR', 'AND' OR VERB TO IMMEDIATELY FOLLOW
                 CONDITION. DELETING TILL ONE OF THESE IS FOUND.
194  IKF3001I-E    TIME-CARD-HOURS NOT DEFINED. SUBSTITUTING TALLY .
203  IKF3001I-E    TIME-CARD-ID NOT DEFINED. DISCARDED.
204  IKF3001I-E    TIME-CARD-HOURS NOT DEFINED. DISCARDED.
205  IKF3001I-E    TIME-CARD-CLOSING-DATE NOT DEFINED. DISCARDED.
206  IKF3001I-E    TIME-CARD-DEPARTMENT NOT DEFINED. DISCARDED.

```

Fig. 9-1 (cont.)

**Tarjeta 12.** En el COBOL del sistema IBM OS/VS, los códigos de error acabados en "W" (IKF1043I-W) indican *precaución*, es decir, que los errores son ligeros y el compilador los puede corregir. En este caso, el programador olvidó el punto al final del párrafo SOURCE-COMPUTER; de todos modos el compilador ha asumido que el punto existía. El compilador *puede* traducir aquellas instrucciones para las que genera un mensaje de precaución.

**Tarjeta 111.** En el COBOL del sistema IBM OS/VS, los códigos de error acabados en "C" (IKF2126I-C) indican *errores condicionales*. Estos errores son más severos que los anteriores, pero el compilador todavía intenta "adivinar" lo que pretendía el programador y con ello traduce la instrucción al lenguaje máquina; no obstante, la probabilidad de que el compilador acierte es inferior que en los errores anteriores. En este caso, la cláusula VALUE contenía una constante de mayor tamaño que el permitido por PICTURE; por ello el compilador trunca la constante (VALUE "//" se convierte en "/" para estar de acuerdo con PIC X).

**Tarjeta 192.** En el COBOL del sistema IBM OS/VS, los códigos de error acabados en "E" (IKF4030I-E) indican *errores severos*. En este caso, el compilador no puede hacer las rectificaciones necesarias y por tanto no puede efectuar la traducción al lenguaje máquina. Observe que en este ejemplo el mensaje de error no indica al programador el fallo que hay en la instrucción. El error es que el programador ha utilizado el nombre de una condición ("VALID-RECORD") en lugar del elemento de datos ("WS-ERROR-DETECTED-SW") en una condición relación ("VALID-RECORD EQUAL "YES").

**Tarjeta 40.** En este error sintáctico se ilustra el hecho de que los mensajes para el diagnóstico de errores no siempre se refieren a la línea en la que realmente figura el error. Aquí la línea 40 es un punto estructurado para finalizar la instrucción FD para TIME-CARD-FILE. El mensaje de error dice: FILE-NAME NOT DEFINED IN A SELECT. Aunque el mensaje se refiera a la linea 40, el verdadero fallo está en la línea 17 y es tipográfico puesto que pone "SELECT TIME-CARD-FILE".

**Tarjetas 194, 203, 204, 205, 206.** Debido al error de la tarjeta 40, el compilador no procesa la instrucción FD ni la descripción del registro al nivel 01 para el fichero TIME-CARD-FILE. Por ello, cualquier referencia a los campos de TIME-CARD-RECORD en la división de procedimientos es un error sintáctico. El mensaje

indica que el elemento correspondiente está sin definir ("NOT DEFINED") porque para el compilador así sucede —por más que el programador ha definido estos elementos entre las líneas 42-49 de la división de datos.

## 9.2 DEPURACION DE ERRORES LOGICOS GRAVES

Cuando un error lógico grave ocasiona el aborto del programa, el sistema operativo cancela la ejecución del resto del mismo. Más aún, si el lenguaje de control de trabajos lo requiere, el sistema operativo imprimirá un *mensaje de término* (en un fichero que se ha de especificar en el lenguaje de control de trabajos). Los mensajes de término contienen información de uno de los dos tipos siguientes (o de ambos):

**Memoria.** Contenido de la memoria de la computadora en el instante en que tuvo lugar el error lógico grave. Con frecuencia se puede determinar la causa del error examinando el contenido de algunos datos.

**Programa.** Instrucciones ejecutadas por la computadora antes y en el momento del error. Por ejemplo, se podría imprimir el número de línea de cada instrucción antes de ejecutar su código máquina equivalente.

En algunos sistemas de computadoras sólo se dispone de un mensaje de término en hexadecimal producido por el sistema operativo; es decir, el mensaje está constituido por una serie de números en base 16. Un programador en COBOL que se precie deberá aprender por lo menos suficiente *lenguaje ensamblador* como para poder interpretar los mensajes de término. Otras computadoras disponen de *mensajes simbólicos* en los cuales se imprimen algunos o todos los elementos de la división de datos junto con la trayectoria del programa; los elementos se imprimen en formato DISPLAY y con sus nombres en COBOL.

## 9.3 DEPURACION DE ERRORES LOGICOS LEVES

Los errores sintácticos ocasionan mensajes de error, los errores lógicos graves producen mensajes de término, pero los errores lógicos leves son más difíciles de depurar al no existir ningún tipo de mensaje. Depende del ingenio del programador determinar lo que está fallando y por qué.

Algunas de las posibilidades que ofrece el lenguaje COBOL son especialmente útiles al programador en el proceso de identificar los errores lógicos leves. Como estas instrucciones especiales son también bastante útiles a la hora de proporcionar información respecto del estado de ejecución del programa antes de un error lógico grave (véase el Ejemplo 9.6), se presentan por separado en las Secciones 9-4 a 9-6.

## 9.4 OBTENCION DE INFORMACION DE TRAZAS DEL PROGRAMA

Una forma de obtener información de trazas del programa es colocar al principio de cada párrafo de la división de procedimientos una línea de depuración (con una "D" en la columna 7) que contenga una instrucción DISPLAY. Cuando se entra en el párrafo se ejecuta la instrucción DISPLAY, que imprime un mensaje que sirve para conocer la traza seguida por el programa.

### EJEMPLO 9.2

SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.

.....  
PROCEDURE DIVISION.

D DISPLAY "COMIENZO EJECUCION PROGRAMA"

.....

PARA-ONE.  
D DISPLAY "SE ENTRA EN PARA-UNO"

PARA-TWO.  
D DISPLAY "SE ENTRA EN PARA-DOS"

Cuando el programa está depurado en su totalidad, se puede eliminar la frase WITH DEBUGGING MODE del párrafo SOURCE-COMPUTER. De este modo, todas las líneas que contengan una "D" en la columna 7 serán tratadas como comentarios, eliminándose así la impresión de la traza del programa.

El COBOL del sistema IBM OS/VS dispone de una instrucción especial cuya función es producir la traza del programa:

#### READY TRACE

hace que se imprima el número de línea de cada párrafo (o el propio nombre de párrafo según se especifique con el lenguaje de control de trabajos en el momento de la compilación) al ejecutarse. Como los párrafos de un bucle puede que se ejecuten muchas veces, existe otra instrucción,

#### RESET TRACE

que desactiva la impresión de la traza. READY TRACE y RESET TRACE pueden ejecutarse tantas veces como se deseé en un programa.

**EJEMPLO 9.3** La división de procedimientos de la Figura 9-2 contiene la instrucción READY TRACE; a continuación se ve una muestra de la salida. Las instrucciones READY/RESET TRACE no están disponibles en todos los compiladores. En el COBOL del sistema IBM OS/VS, la instrucción READY TRACE imprime la traza en el fichero especial del sistema SYSOUT; por tanto, debe estar presente una instrucción del lenguaje de control de trabajos al utilizar READY TRACE (de otro modo el programa abortaría). Observe que en la traza se escribe el nombre COBOL del párrafo cuando se ejecuta. Alterando el lenguaje de control de trabajos se podría conseguir que se imprimieran los *números de línea* generados automáticamente por el compilador en lugar de los nombres de párrafo.

```

00070      PROCEDURE DIVISION.
00072      *-----*
00073          READY TRACE
00074      *-----*
00075
00076          MOVE "NO" TO INTER-FIN-TARJETAS
00077          MOVE ZERO TO VENTAS-TOTAL
00078
00079          OPEN    INPUT        FICHERO-TARJETAS
00080              OUTPUT       FICHERO-IMPRESION
00081
00082          PERFORM 100-ENTRADA-REGISTRO
00083
00084          PERFORM 200-PROD-LISTADO-VENTAS
00085              UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00086
00087          WRITE LINEA-IMPRESION
00088              FROM LINEA-TOTAL-VENTAS
00089
00090          CLOSE   FICHERO-TARJETAS
00091              FICHERO-IMPRESION
00092          STOP RUN
00093
00094
00095          100-ENTRADA-REGISTRO
00096
00097          READ FICHERO-TARJETAS RECORD
00098              INTO TARJETA-TRAB-VENTAS
00099              AT END
00100                  MOVE "SI" TO INTER-FIN-TARJETAS
00101

```

Fig. 9-2

```

00103      200-PROD-LISTADO-VENTAS.
00104
00105      ADD IMPORTE-VENTAS-TRAB TO VENTAS-TOTAL
00106
00107      MOVE ID-VENTAS-TRAB      TO ID-VENTAS-LISTADO
00108      MOVE ID-CLIENTE-TRAB    TO ID-CLIENTE-LISTADO
00109      MOVE IMPORTE-VENTAS-TRAB TO IMPORTE-VENTAS-LISTADO
00110
00111      WRITE LINEA-IMPRESION
00112          FROM LINEA-LISTADO-VENTAS
00113
00114      PERFORM 100-ENTRADA-REGISTRO
00115
100-ENTRADA-REGISTRO ,200-PROD-LISTADO-VENTAS ,100-ENTRADA-REGISTRO ,200-PROD-LISTADO-VENTAS ,
100-ENTRADA-REGISTRO ,200-PROD-LISTADO-VENTAS ,100-ENTRADA-REGISTRO ,200-PROD-LISTADO-VENTAS ,
100-ENTRADA-REGISTRO ,200-PROD-LISTADO-VENTAS ,100-ENTRADA-REGISTRO ,200-PROD-LISTADO-VENTAS ,
100-ENTRADA-REGISTRO ,

```

Fig. 9-2 (cont.)

Como complemento a READY RESET TRACE el sistema IBM OS/VS dispone de la siguiente ampliación del COBOL ANS:

$$\begin{aligned}
 & \text{ON } \left\{ \begin{array}{l} \text{entero-1} \\ \text{identificador-1} \end{array} \right\} \left[ \text{AND EVERY } \left\{ \begin{array}{l} \text{entero-2} \\ \text{identificador-2} \end{array} \right\} \right] \left[ \text{UNTIL } \left\{ \begin{array}{l} \text{entero-3} \\ \text{identificador-3} \end{array} \right\} \right] \\
 & \quad \left[ \begin{array}{l} \text{ELSE} \\ \text{instrucción imperativa} \end{array} \right]
 \end{aligned}$$

#### EJEMPLO 9.4

SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.

.....

PROCEDURE DIVISION.

.....

PERFORM PARA-A  
    UNTIL UNA-CONDICION-U-OTRA

.....

PARA-A.  
D    ON 1 AND EVERY 2 UNTIL 9  
D    READY TRACE  
D    ELSE  
D    RESET TRACE  
D

.....

La instrucción READY TRACE se ejecuta al entrar en PARA-A por primera, tercera, quinta, séptima y novena vez. En el resto de las pasadas de PARA-A se ejecuta RESET TRACE (en lugar de READY TRACE). Observe la utilización del carácter "D" en la columna 7 en todas las instrucciones de depuración (incluyendo el punto estructurado al final de la instrucción ON). De este modo, todas estas instrucciones se traducen al lenguaje máquina sólo si la cláusula WITH DEBUGGING MODE se especifica en el párrafo SOURCE-COMPUTER.

Algunos ejemplos más de usos correctos de la instrucción ON son:

- ON 3 AND EVERY 5 READY TRACE ELSE RESET TRACE.
- ON 1 READY TRACE ELSE RESET TRACE.
- ON 1 AND EVERY 2 DISPLAY REGISTRO-ENTRADA.

- ON 1 AND EVERY 2  
DISPLAY REGISTRO-ENTRADA  
READY TRACE  
ELSE  
RESET TRACE

## 9.5 LECTURA DE DATOS DURANTE LA EJECUCION DEL PROGRAMA

La mejor manera de conocer y leer el contenido de las áreas de memoria de una forma dinámica es incluir líneas de depuración con instrucciones DISPLAY en el programa. Cuando se ejecute DISPLAY, se escribirá el contenido del elemento de datos especificado. Suele ser buena idea escribir cada registro de entrada justo después de la instrucción READ y cada registro de salida justo antes de cada instrucción WRITE.

### EJEMPLO 9.5

SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.

PROCEDURE DIVISION.

READ FICHERO-ENTRADA  
AT END MOVE "SI" TO INTER-FIN-FICHERO

D DISPLAY "REGISTRO ENTRADA ES:" REG-ENTRADA

D DISPLAY "REGISTRO SALIDA ES:" REG-SALIDA  
WRITE REG-SALIDA

ADD 1 TO CONTADOR-EJEMPLO

D DISPLAY "CONTADOR-EJEMPLO DESPUES DE INCREMENTARSE:"  
CONTADOR-EJEMPLO

Observe el uso de literales no numéricos en las instrucciones DISPLAY para facilitar la depuración. Con estas *etiquetas* también se obtiene información sobre la trayectoria del programa. La salida de las instrucciones de arriba podría ser la siguiente:

REGISTRO ENTRADA ES:0003A789108264  
REGISTRO SALIDA ES:172A12 CARTONSB97634900000  
CONTADOR-EJEMPLO DESPUES DE INCREMENTARSE:0007

Recuerde: al utilizar DISPLAY con el COBOL del sistema IBM OS/VS debe definirse, mediante una instrucción del lenguaje de control de trabajos, el fichero SYSOUT, de lo contrario el programa abortará.

En muchas versiones de COBOL se dispone de una instrucción que imprime no sólo el contenido de un dato, sino también su nombre COBOL; la sintaxis es

EXHIBIT { NAMED  
CHANGED NAMED  
CHANGED } { identificador-1 } { literal-1 } { identificador-2 } { literal-2 } ...

EXHIBIT NAMED... imprime el contenido de cada elemento de datos junto con su nombre COBOL en la forma nombre-cobol = contenido.

EXHIBIT CHANGED NAMED... imprime el nombre COBOL de un elemento de datos seguido de los contenidos que va teniendo cada vez que se ejecuta la instrucción EXHIBIT.

**EXHIBIT CHANGED...** imprime el contenido del elemento de datos indicado si hubiera cambiado desde la última ejecución de EXHIBIT; *no* se imprime el nombre COBOL.

En el COBOL del sistema IBM OS/VS debe definirse el fichero SYSOUT, mediante una instrucción del lenguaje de control de trabajos, cuando se utiliza la instrucción EXHIBIT.

**EJEMPLO 9.6** Al programa de la Figura 9-3 se le han añadido instrucciones READY TRACE y EXHIBIT NAMED para ayudar en la depuración de un ABEND (aborts). La salida para la depuración, Figura 9-4, ilustra: el modo en que READY TRACE y EXHIBIT... se entremezclan en el fichero de impresión SYSOUT, cómo EXHIBIT... comienza automáticamente una nueva línea si la información no cabe en una, y que EXHIBIT... y READY TRACE generan una salida que sirve como "historia" de lo que ha ocurrido durante la ejecución del programa. Compare el informe del final de la Figura 9-3 con la salida de depuración para asegurarse de que sigue la lógica del programa al procesar cada registro. El último párrafo que se ejecuta (de acuerdo con la trayectoria) es 200-PRODUCE-LISTADO-TIEMPO. Observe que la computadora llegó por lo menos hasta la instrucción EXHIBIT de este párrafo, puesto que el contenido de HORAS-TRABAJO-REGULAR y HORAS-EXTRAS-TRABAJO aparecen en el listado de salida. Se puede deducir que el error ABEND ocurrió durante la ejecución de la instrucción COMPUTE de la línea 116, porque la salida desde la instrucción EXHIBIT muestra que HORAS-EXTRAS-TRABAJO contiene *espacios en blanco*. Este ABEND es, por tanto, consecuencia de los *datos de entrada* y no del programa. El programa debería rediseñarse para que validase la entrada de datos antes de procesarlos.

```

00001      IDENTIFICATION DIVISION.
00002
00003      PROGRAM-ID. TIMELIST.
00004
00005      AUTHOR. LARRY NEWCOMER.
00006      INSTALLATION. PENN STATE UNIVERSITY, YORK CAMPUS.
00007
00008      DATE-COMPILED. MAY 9,1983.

00011      ENVIRONMENT DIVISION.

00013      CONFIGURATION SECTION.

00015      SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.
00016      OBJECT-COMPUTER. IBM-3081.

00018      INPUT-OUTPUT SECTION.

00020      FILE-CONTROL.

00022          SELECT FICHERO-TARJETAS   ASSIGN TO CARDS.
00023          SELECT FICHERO-IMPRESION  ASSIGN TO PRINTER.

00025      DATA DIVISION.

00027      FILE SECTION.

00029          FD  FICHERO-TARJETAS
00030              RECORD CONTAINS 80 CHARACTERS
00031              LABEL RECORDS ARE OMITTED
00032
00033
00034          01  TARJETA-TIEMPO-ENTRADA    PIC X(80).

00036          FD  FICHERO-IMPRESION
00037              RECORD CONTAINS 132 CHARACTERS
00038              LABEL RECORDS ARE OMITTED
00039
00040
00041          01  LINEA-IMPRESION        PIC X(132).

00043      WORKING-STORAGE SECTION.

```

Fig. 9-3

```

00045      01 INTER-FIN-TARJETAS      PIC X(3).
00047      01 TARJETA-TIEMPO-TRAB.
00048
00049          05 NOMBRE-TRABAJO      PIC X(20).
00050          05 HORAS-TRABAJO-REGULAR  PIC 99.
00051          05 HORAS-EXTRAS-TRABAJO  PIC 99.
00052          05 FILLER            PIC X(56).

00054      01 LINEA-LISTADO-TIEMPO.
00055
00056          05 NOMBRE-LISTADO      PIC X(20).
00057          05 FILLER            PIC X(5)   VALUE SPACES.
00058          05 HORAS-REGULARES-LISTADO  PIC 99.
00059          05 FILLER            PIC X(5)   VALUE SPACES.
00060          05 HORAS-EXTRAS-LISTADO  PIC 99.
00061          05 FILLER            PIC X(5)   VALUE SPACES.
00062          05 HORAS-TOTALES-LISTADO  PIC 999.
00063          05 FILLER            PIC X(85)  VALUE SPACES.

00065      01 LINEA-TOTAL-EMPLEADO.
00066
00067          05 FILLER            PIC X(25)
00068          05 NUMERO-EMPLEADOS    VALUE "NUM. DE EMPLEADOS ES".
00069          05 FILLER            PIC 999.
00070          05 FILLER            PIC X(104)  VALUE SPACES.

00072      PROCEDURE DIVISION.

00074      D READY TRACE
00075
00076          MOVE "NO" TO INTER-FIN-TARJETAS
00077          MOVE ZERO TO NUMERO-EMPLEADOS
00078
00079          OPEN   INPUT      FICHERO-TARJETAS
00080          OUTPUT     FICHERO-IMPRESION
00081
00082          PERFORM 100-ENTRADA-REGISTRO
00083
00084          PERFORM 200-PRODUCE-LISTADO-TIEMPO
00085          UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00086
00087          WRITE LINEA-IMPRESION
00088          FROM LINEA-TOTAL-EMPLEADO
00089
00090          CLOSE   FICHERO-TARJETAS
00091          FICHERO-IMPRESION
00092          STOP RUN
00093

00095      100-ENTRADA-REGISTRO.
00096
00097          READ FICHERO-TARJETAS RECORD
00098          INTO TARJETA-TIEMPO-TRAB
00099          AT END
00100          MOVE "SI" TO INTER-FIN-TARJETAS
00101
00102          D EXHIBIT NAMED TARJETA-TIEMPO-TRAB
00103          D INTER-FIN-TARJETAS
00104          D

00106      200-PRODUCE-LISTADO-TIEMPO.
00107
00108          ADD 1 TO NUMERO-EMPLEADOS
00109
00110          MOVE NOMBRE-TRABAJO      TO NOMBRE-LISTADO

```

Fig. 9-3 (cont.)

```

00111      MOVE HORAS-TRABAJO-REGULAR TO HORAS-REGULARES-LISTADO
00112      MOVE HORAS-EXTRAS-TRABAJO TO HORAS-EXTRAS-LISTADO
00113
00114      D EXHIBIT NAMED HORAS-TRABAJO-REGULAR
00115          HORAS-EXTRAS-TRABAJO
00116          COMPUTE HORAS-TOTALES-LISTADO = HORAS-TRABAJO-REGULAR
00117              + HORAS-EXTRAS-TRABAJO
00118
00119      WRITE LINEA-IMPRESION
00120          FROM LINEA-LISTADO-TIEMPO
00121
00122      PERFORM 100-ENTRADA-REGISTRO
00123

```

-----  
PRINTED REPORT:

ABEL FRED	40	03	043
BAKER SUE	30	00	030
CHARLIE CHUCK	40	12	052
PERELMAN BARNEY	40	15	055
PERELMAN MIKE	03	00	003

Fig. 9-3 (cont.)

```

100-ENTRADA-REGISTRO ,
TARJETA-TIEMPO-TRAB = ABEL FRED        4003           INTER-FIN-TARJETAS
= NO
200-PRODUCE-LISTADO-TIEMPO ,
HORAS-TRABAJO-REGULAR = 40 HORAS-EXTRAS-TRABAJO = 03
100-ENTRADA-REGISTRO ,
TARJETA-TIEMPO-TRAB = BAKER SUE        3000           INTER-FIN-TARJETAS
= NO
200-PRODUCE-LISTADO-TIEMPO ,
HORAS-TRABAJO-REGULAR = 30 HORAS-EXTRAS-TRABAJO = 00
100-ENTRADA-REGISTRO ,
TARJETA-TIEMPO-TRAB = CHARLIE CHUCK    4012           INTER-FIN-TARJETAS
= NO
200-PRODUCE-LISTADO-TIEMPO ,
HORAS-TRABAJO-REGULAR = 40 HORAS-EXTRAS-TRABAJO = 12
100-ENTRADA-REGISTRO ,
TARJETA-TIEMPO-TRAB = PERELMAN BARNEY  4015           INTER-FIN-TARJETAS
= NO
200-PRODUCE-LISTADO-TIEMPO ,
HORAS-TRABAJO-REGULAR = 40 HORAS-EXTRAS-TRABAJO = 15
100-ENTRADA-REGISTRO ,
TARJETA-TIEMPO-TRAB = PERELMAN MIKE    0300           INTER-FIN-TARJETAS
= NO
200-PRODUCE-LISTADO-TIEMPO ,
HORAS-TRABAJO-REGULAR = 03 HORAS-EXTRAS-TRABAJO = 00
100-ENTRADA-REGISTRO ,
TARJETA-TIEMPO-TRAB = TROUBLE TOM      40           INTER-FIN-TARJETAS
= NO
200-PRODUCE-LISTADO-TIEMPO ,
HORAS-TRABAJO-REGULAR = 40 HORAS-EXTRAS-TRABAJO =

```

Fig. 9-4

## 9.6 LAS INSTRUCCIONES DECLARATIVES Y USE FOR DEBUGGING

EL COBOL norma ANSI proporciona unas rutinas especiales de depuración denominadas *DECLARATIVES*, que deben escribirse al principio de la división de procedimientos. Estas rutinas son únicas en el sentido de que no se ejecutan como parte de la lógica del programa; sino sólo si suceden determinados eventos.

### EJEMPLO 9.7

```

PROCEDURE DIVISION.
  DECLARATIVES.
    DECLARATIVES-SECTION-1 SECTION.
    .....
      DECLARATIVE-SECTION-2 SECTION.
      .....
    END DECLARATIVES.
  ...(instrucciones normales de la PROCEDURE DIVISION)
```

Observe la cabecera “DECLARATIVES” y la línea final “END DECLARATIVES” (escritas en el margen A).

Cada DECLARATIVE SECTION debe comenzar con una instrucción *USE*, en la que se define cuándo debe ejecutarse. El formato general de las DEBUGGING DECLARATIVES aparece en la Figura 9-5.

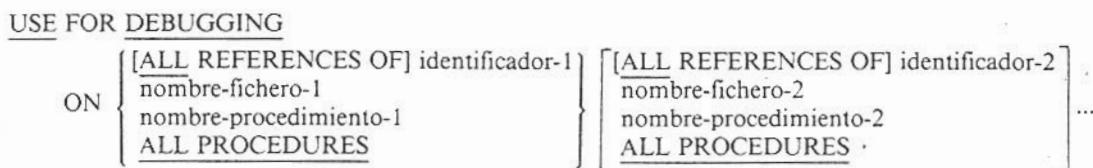


Fig. 9-5

### EJEMPLO 9.8

```

PROCEDURE DIVISION.
  DECLARATIVES.
    COMPROBACION-CONTADOR.
    USE FOR DEBUGGING
      ON ALL REFERENCES OF NUMERO-CLIENTES-FACTURADOS
      EXHIBIT NAMED NUMERO-CLIENTES-FACTURADOS
      EXHIBIT NAMED ID-CLIENTE-CORRIENTE
    END DECLARATIVES.
  .....
```

Aquí se ha utilizado la opción “ON ALL REFERENCES OF identificador-1”. La SECTION (en este caso dos instrucciones EXHIBIT NAMED) se ejecutarán inmediatamente antes de cada instrucción WRITE de “identificador-1” y también inmediatamente después de cualquier instrucción COBOL que manipule “identificador-1”.

A continuación se dan otras variaciones de USE FOR DEBUGGING...

- ON identificador-1 (ON REGISTRO-EJEMPLO-SALIDA)

“Identificador-1” normalmente será el nombre COBOL de un registro para un fichero de salida. Se ejecuta DECLARATIVE inmediatamente antes de cada instrucción WRITE para el fichero (por ejemplo, antes de WRITE REGISTRO-EJEMPLO-SALIDA).

- ON nombre-fichero-1 (ON FICHERO-MAESTRO-EMPLEADOS)

Se ejecuta DECLARATIVE después de cualquier instrucción OPEN, CLOSE o READ para el fichero indicado.

- ON nombre-procedimiento-1 (ON 200-PRODUCE-ETIQUETAS-CORREO)

“Nombre-procedimiento-1” es un párrafo o nombre de sección de la división de procedimientos. DECLARATIVE se ejecuta inmediatamente *antes* que el procedimiento en cuestión.

- ON ALL PROCEDURES

DECLARATIVE se ejecuta *antes* que cualquier párrafo o sección (excepto la propia sección DECLARATIVE).

Al igual que sucede con otras instrucciones de depuración, las DEBUGGING DECLARATIVES se traducen al lenguaje máquina sólo si se especifica la cláusula WITH DEBUGGING MODE en el párrafo SOURCE-COMPUTER.

Cuando se ejecuta DEBUGGING DECLARATIVE, el sistema sitúa información valiosa en un área de datos especial denominada DEBUG-ITEM. DEBUG-ITEM es una palabra reservada de COBOL, no debe definirse por el programador en la división de datos; su composición es la que aparece en la Figura 9-6.

01 DEBUG-ITEM.		
05 DEBUG-LINE	PIC X(6).	
05 FILLER	PIC X	VALUE SPACES.
05 DEBUG-NAME	PIC X(30).	
05 FILLER	PIC X	VALUE SPACES.
05 DEBUG-SUB-1	PIC S9999	SIGN IS LEADING SEPARATE CHARACTER.
05 FILLER	PIC X	VALUE SPACES.
05 DEBUG-SUB-2	PIC S9999	SIGN IS LEADING SEPARATE CHARACTER.
05 FILLER	PIC X	VALUE SPACES.
05 DEBUG-SUB-3	PIC S9999	SIGN IS LEADING SEPARATE CHARACTER.
05 FILLER	PIC X	VALUE SPACES.
05 DEBUG-CONTENTS	PIC X(n).	

Fig. 9-6

El contenido de los campos de nivel 05 es el siguiente:

- (1) DEBUG-LINE contiene el número de línea generado por el compilador para la instrucción que hizo que se ejecutara DECLARATIVE.
- (2) DEBUG-NAME contiene los 30 primeros caracteres del nombre que originó la ejecución de DECLARATIVE.
- (3) DEBUG-SUB-1, DEBUG-SUB-2 y DEBUG-SUB-3 contienen los subíndices del valor del elemento DEBUG-NAME, si éste fuera una tabla (véase el Capítulo 10); de otro modo, estos campos de DEBUG-ITEM contienen espacios en blanco.
- (4) DEBUG-CONTENTS tiene el contenido del elemento (cuyo nombre está en DEBUG-NAME) que ocasionó la ejecución de DECLARATIVE. Si se está ejecutando DECLARATIVE como consecuencia del nombre de procedimiento que aparece en USE FOR DEBUGGING ON, DEBUG-NAME contendrá el nombre del procedimiento y DEBUG-CONTENTS contendrá una frase corta en inglés describiendo el tipo de procedimiento.

DEBUG-ITEM contiene una información tan útil que con frecuencia lo único que se necesita en la DECLARATIVE SECTION es una instrucción DISPLAY DEBUG-ITEM o EXHIBIT DEBUG-ITEM.

**EJEMPLO 9.9** Se reconsidera el error ABEND del Ejemplo 9.6, utilizando en esta ocasión DECLARATIVE como herramienta de depuración. En la Figura 9-7 se tiene la nueva división de procedimientos en la que DECLARATIVE está en las líneas 74-88. Cuando se incluyen varias condiciones en la cláusula ON, la ocurrencia de cualquiera de ellas hará que se ejecute DECLARATIVE. Observe que lo único que lleva a cabo DECLARATIVE es EXHIBIT DEBUG-ITEM. La salida ejemplo de la Figura 9-8 muestra la copiosa información que, sin embargo, esta instrucción simple produce. (El formato de la Figura 9-8 es el que se indicó en la Figura 9-6, con blancos en los campos DEBUG-SUB y con "PERFORM LOOP" como contenido de DEBUG-CONTENTS para 200-PRODUCE-LISTADO-TIEMPO.) Recalcamos de nuevo que el programa alcanzó durante su ejecución por lo menos la línea 124 (la última línea que aparece en el listado de depuración) y que en ese momento HORAS-EXTRAS-TRABAJO contenía blancos. Como la línea 125 del programa COBOL está en blanco y la línea 126 utiliza HORAS-EXTRAS-TRABAJO (que entonces contenía SPACES) en un cálculo, la causa del error ABEND está clara (véase el Ejemplo 9.6).

```

00072 PROCEDURE DIVISION.

00074 DECLARATIVES.

00075 IDENTIFY-ABEND SECTION.

00077      USE FOR DEBUGGING
00079          ON ALL REFERENCES OF TARJETA-TIEMPO-TRABAJO
00080              ALL REFERENCES OF HORAS-TRABAJO-REGULAR
00081                  ALL REFERENCES OF HORAS-EXTRAS-TRABAJO
00082                      ALL REFERENCES OF 200-PRODUCE-LISTADO-TIEMPO

00083
00084
00085      EXHIBIT NAMED DEBUG-ITEM
00086
00087
00088 END DECLARATIVES.

00089
00090
00091      MOVE "NO" TO INTER-FIN-TARJETAS
00092      MOVE ZERO TO NUMERO-EMPLEADOS
00093
00094      OPEN    INPUT        FICHERO-TARJETAS
00095          OUTPUT       FICHERO-IMPRESION
00096
00097      PERFORM 100-ENTRADA-REGISTRO
00098
00099      PERFORM 200-PRODUCE-LISTADO-TIEMPO
00100          UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00101
00102      WRITE LINEA-IMPRESION
00103          FROM LINEA-TOTAL-EMPLEADO
00104
00105      CLOSE   FICHERO-TARJETAS
00106          FICHERO-IMPRESION
00107
00108      STOP RUN

00110
00111 100-ENTRADA-REGISTRO.

00112      READ FICHERO-TARJETAS RECORD
00113          INTO TARJETA-TIEMPO-TRABAJO
00114          AT END
00115              MOVE "SI" TO INTER-FIN-TARJETAS
00116

00118
00119 200-PRODUCE-LISTADO-TIEMPO.

00120      ADD 1 TO NUMERO-EMPLEADOS
00121
00122      MOVE NOMBRE-TRABAJO      TO NOMBRE-LISTADO
00123      MOVE HORAS-TRABAJO-REGULAR  TO HORAS-REGULARES-LISTADO
00124      MOVE HORAS-EXTRAS-TRABAJO  TO HORAS-EXTRAS-LISTADO
00125
00126      COMPUTE HORAS-TOTALES-LISTADO = HORAS-TRABAJO-REGULAR
00127          + HORAS-EXTRAS-TRABAJO
00128
00129      WRITE LINEA-IMPRESION
00130          FROM LINEA-LISTADO-TIEMPO
00131
00132      PERFORM 100-ENTRADA-REGISTRO
00133

```

Fig. 9-7

```

DEBUG-ITEM = 000112 TARJETA-TIEMPO-TRABAJO      ABEL FRED      4003
DEBUG-ITEM = 000099 200-PRODUCE-LISTADO-TIEMPO    PERFORM LOOP
DEBUG-ITEM = 000123 HORAS-TRABAJO-REGULAR        40
DEBUG-ITEM = 000124 HORAS-EXTRAS-TRABAJO         03
DEBUG-ITEM = 000126 HORAS-TRABAJO-REGULAR        40
DEBUG-ITEM = 000126 HORAS-EXTRAS-TRABAJO         03
DEBUG-ITEM = 000112 TARJETA-TIEMPO-TRABAJO      BAKER SUE      3000

DEBUG-ITEM = 000099 200-PRODUCE-LISTADO-TIEMPO    PERFORM LOOP
DEBUG-ITEM = 000123 HORAS-TRABAJO-REGULAR        30
DEBUG-ITEM = 000124 HORAS-EXTRAS-TRABAJO         00
DEBUG-ITEM = 000126 HORAS-TRABAJO-REGULAR        30
DEBUG-ITEM = 000126 HORAS-EXTRAS-TRABAJO         00
DEBUG-ITEM = 000112 TARJETA-TIEMPO-TRABAJO      CHARLIE CHUCK   4012

DEBUG-ITEM = 000099 200-PRODUCE-LISTADO-TIEMPO    PERFORM LOOP
DEBUG-ITEM = 000123 HORAS-TRABAJO-REGULAR        40
DEBUG-ITEM = 000124 HORAS-EXTRAS-TRABAJO         12
DEBUG-ITEM = 000126 HORAS-TRABAJO-REGULAR        40
DEBUG-ITEM = 000126 HORAS-EXTRAS-TRABAJO         12
DEBUG-ITEM = 000112 TARJETA-TIEMPO-TRABAJO      PERELMAN BARNEY  4015

DEBUG-ITEM = 000099 200-PRODUCE-LISTADO-TIEMPO    PERFORM LOOP
DEBUG-ITEM = 000123 HORAS-TRABAJO-REGULAR        40
DEBUG-ITEM = 000124 HORAS-EXTRAS-TRABAJO         15
DEBUG-ITEM = 000126 HORAS-TRABAJO-REGULAR        40
DEBUG-ITEM = 000126 HORAS-EXTRAS-TRABAJO         15
DEBUG-ITEM = 000112 TARJETA-TIEMPO-TRABAJO      PERELMAN MIKE   0300

DEBUG-ITEM = 000099 200-PRODUCE-LISTADO-TIEMPO    PERFORM LOOP
DEBUG-ITEM = 000123 HORAS-TRABAJO-REGULAR        03
DEBUG-ITEM = 000124 HORAS-EXTRAS-TRABAJO         00
DEBUG-ITEM = 000126 HORAS-TRABAJO-REGULAR        03
DEBUG-ITEM = 000126 HORAS-EXTRAS-TRABAJO         00
DEBUG-ITEM = 000112 TARJETA-TIEMPO-TRABAJO      TROUBLE TOM     40

DEBUG-ITEM = 000099 200-PRODUCE-LISTADO-TIEMPO    PERFORM LOOP
DEBUG-ITEM = 000123 HORAS-TRABAJO-REGULAR        40
DEBUG-ITEM = 000124 HORAS-EXTRAS-TRABAJO

```

Fig. 9-8

### Preguntas de repaso

- 9.1 ¿En qué consiste la comprobación de un programa?, ¿y su depuración?
- 9.2 Comente los tres tipos de errores que pueden aparecer durante la comprobación de un programa.
- 9.3 ¿Qué son los mensajes de diagnóstico de error?
- 9.4 Comente las diferencias entre los errores sintácticos de niveles W, C y E.
- 9.5 ¿Qué es una visualización de la memoria al producirse el aborto de un programa? Diga las diferencias que existen entre los mensajes simbólicos y en hexadecimal.
- 9.6 ¿Qué es la traza de un programa? ¿Por qué es útil en su depuración?
- 9.7 ¿Qué información proporciona por lo general un mensaje simbólico?
- 9.8 ¿Cuántas instrucciones DISPLAY se pueden utilizar para la depuración?

- 9.9 Indique las diferencias entre READY TRACE y RESET TRACE.
- 9.10 Indique la sintaxis y el uso de la instrucción ON.
- 9.11 Comente la sintaxis y el uso de la instrucción EXHIBIT.
- 9.12 ¿Qué formato tiene la salida de EXHIBIT NAMED...?
- 9.13 ¿Qué son las DECLARATIVES?
- 9.14 ¿En qué se distinguen las DECLARATIVES del resto de las instrucciones de la división de procedimientos?
- 9.15 Comente los efectos en USE FOR DEBUGGING de lo siguiente: (a) ON ALL REFERENCES..., (b) ON nombre-fichero, (c) ON nombre-procedimiento y (d) ON ALL PROCEDURES.
- 9.16 Comente el efecto del párrafo SOURCE-COMPUTER en las DECLARATIVES y en las líneas con un carácter "D" en la columna 7.
- 9.17 Indique el contenido de DEBUG-ITEM y cómo se utiliza este elemento.

### Problemas resueltos

- 9.18 Depure el siguiente programa, que contiene errores lógicos leves. Dispone de listado fuente (Fig. 9-9), salida de comprobación (Fig. 9-10) y salida de depuración de una DEBUGGING DECLARATIVE (Fig. 9-11). La salida es incorrecta porque (i) el contador de empleados debe ser 006 en lugar de 012; (ii) en ningún empleado la suma de las horas regulares y las horas extras da las horas totales trabajadas (las columnas de la Figura 9-10 lo ponen de manifiesto).

```

00001      IDENTIFICATION DIVISION.
00002
00003      PROGRAM-ID. TIMELIST.
00004
00005      AUTHOR. LARRY NEWCOMER.
00006      INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS.
00007
00008      DATE-WRITTEN. MAY 1983.
00009      DATE-COMPILED. MAY 9,1983.
00011      *      TIMELIST IMPRIME UNA LINEA POR CADA EMPLEADO MOSTRANDO:
00012      *      NUMERO DE HORAS REGULARES TRABAJADAS, NUMERO DE HORAS EXTRAS
00013      *      TRABAJADAS Y TOTAL HORAS TRABAJADAS. AL FINAL DEL INFORME,
00014      *      IMPRIME UN CONTADOR DEL NUMERO DE EMPLEADOS PROCESADOS.
00015      *      LA ENTRADA SE REALIZA CON UN BLOQUE DE TARJETAS,
00016      *      ORDENADAS POR NOMBRE DEL EMPLEADO.

00018      ENVIRONMENT DIVISION.

00020      CONFIGURATION SECTION.

00022      SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.
00023      OBJECT-COMPUTER. IBM-3081.

00025      INPUT-OUTPUT SECTION.

```

Fig. 9-9

```

00027      FILE-CONTROL.

00029          SELECT FICHERO-TARJETAS      ASSIGN TO CARDS.
00030          SELECT FICHERO-IMPRESION    ASSIGN TO PRINTER.

00032      DATA DIVISION.

00034      FILE SECTION.

00036          FD  FICHERO-TARJETAS
00037              RECORD CONTAINS 80 CHARACTERS
00038              LABEL RECORDS ARE OMITTED

00039
00040
00041          01  TARJETA-TIEMPO-ENTRADA      PIC X(80).

00043          FD  FICHERO-IMPRESION
00044              RECORD CONTAINS 132 CHARACTERS
00045              LABEL RECORDS ARE OMITTED

00046
00047
00048          01  LINEA-IMPRESION           PIC X(132).

00050      WORKING-STORAGE SECTION.

00052          01  INTER-FIN-TARJETAS       PIC X(3).

00054          01  TARJETA-TIEMPO-TRABAJO.

00055
00056          05  NOMBRE-TRABAJO          PIC X(20).
00057          05  HORAS-TRABAJO-REGULAR   PIC 99.
00058          05  HORAS-EXTRAS-TRABAJO   PIC 99.
00059          05  FILLER                 PIC X(56).

00061          01  LINEA-LISTADO-TIEMPO.
00062
00063          05  NOMBRE-LISTADO         PIC X(20).
00064          05  FILLER                PIC X(5)      VALUE SPACES.
00065          05  HORAS-REGULARES-LISTADO PIC 99.
00066          05  FILLER                PIC X(5)      VALUE SPACES.
00067          05  HORAS-EXTRAS-LISTADO  PIC 99.
00068          05  FILLER                PIC X(5)      VALUE SPACES.
00069          05  HORAS-TOTALES-LISTADO PIC 999.
00070          05  FILLER                PIC X(85)     VALUE SPACES.

00072          01  LINEA-TOTAL-EMPLEADO.
00073
00074          05  FILLER                PIC X(25)    VALUE "NUM. DE EMPLEADOS ES".
00075
00076          05  NUMERO-EMPLEADOS      PIC 999.
00077          05  FILLER                PIC X(104)   VALUE SPACES.

00079      PROCEDURE DIVISION.

00081      DECLARATIVES.

00082
00083      COMPROBACION LOGICA SECTION.

00084
00085          USE FOR DEBUGGING
00086              ON ALL REFERENCES OF NUMERO-EMPLEADOS
00087                  ALL REFERENCES OF HORAS-TRABAJO-REGULAR
00088                  ALL REFERENCES OF HORAS-EXTRAS-TRABAJO
00089                  ALL REFERENCES OF HORAS-TOTALES-LISTADO

00090
00091          EXHIBIT NAMED DEBUG-ITEM

```

Fig. 9-9 (cont.)

```

00092
00093
00094      END DECLARATIVES.

00096      MOVE "NO" TO INTER-FIN-TARJETAS
00097      MOVE ZERO TO NUMERO-EMPLEADOS
00098
00099      OPEN   INPUT           FICHERO-TARJETAS
00100             OUTPUT          FICHERO-IMPRESION
00101
00102      PERFORM ENTRADA-REGISTRO
00103
00104      PERFORM PRODUCE-LISTADO-TIEMPO
00105             UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00106
00107      WRITE LINEA-IMPRESION
00108             FROM LINEA-TOTAL-EMPLEADO
00109
00110      CLOSE   FICHERO-TARJETAS
00111             FICHERO-IMPRESION
00112      STOP RUN
00113

00115      ENTRADA-REGISTRO.
00116
00117      READ FICHERO-TARJETAS
00118             INTO TARJETA-TIEMPO-TRABAJO
00119             AT END
00120             MOVE "SI" TO INTER-FIN-TARJETAS
00121

00123      PRODUCE-LISTADO-TIEMPO.
00124
00125      *           INCREMENTA CONTADOR UNO POR EMPLEADO PROCESADO
00126
00127      ADD 2 TO NUMERO-EMPLEADOS
00128
00129      *           COPIA LA TARJETA EN EL AREA DE CONSTRUCCION
00130
00131      MOVE NOMBRE-TRABAJO      TO NOMBRE-LISTADO
00132      MOVE HORAS-TRABAJO-REGULAR  TO HORAS-REGULARES-LISTADO
00133      MOVE HORAS-EXTRAS-TRABAJO  TO HORAS-EXTRAS-LISTADO
00134
00135      *           CALCULA TOTAL HORAS SUMANDO LAS NORMALES
00136      *           Y LAS EXTRAS.
00137
00138      ADD HORAS-TRABAJO-REGULAR HORAS-EXTRAS-TRABAJO
00139             GIVING HORAS-TOTALES-LISTADO
00140
00141      WRITE LINEA-IMPRESION
00142             FROM LINEA-LISTADO-TIEMPO
00143
00144      PERFORM ENTRADA-REGISTRO
00145

```

Fig. 9-9 (cont.)

ABEL FRED	40	03	080
BAKER SUE	30	00	060
CHARLIE CHUCK	40	12	080
PERELMAN BARNEY	40	15	080
PERELMAN MIKE	03	00	006
TROUBLE TOM	40	10	080
NUMBER OF EMPLOYEES IS	012		

Fig. 9-10

DEBUG-ITEM = 000097 NUMERO-EMPLEADOS	000
DEBUG-ITEM = 000127 NUMERO-EMPLEADOS	002
DEBUG-ITEM = 000132 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000133 HORAS-EXTRAS-TRABAJO	03
DEBUG-ITEM = 000138 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000138 HORAS-TOTALES-LISTADO	080
DEBUG-ITEM = 000127 NUMERO-EMPLEADOS	004
DEBUG-ITEM = 000132 HORAS-TRABAJO-REGULAR	30
DEBUG-ITEM = 000133 HORAS-EXTRAS-TRABAJO	00
DEBUG-ITEM = 000138 HORAS-TRABAJO-REGULAR	30
DEBUG-ITEM = 000138 HORAS-TOTALES-LISTADO	060
DEBUG-ITEM = 000127 NUMERO-EMPLEADOS	006
DEBUG-ITEM = 000132 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000133 HORAS-EXTRAS-TRABAJO	12
DEBUG-ITEM = 000138 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000138 HORAS-TOTALES-LISTADO	080
DEBUG-ITEM = 000127 NUMERO-EMPLEADOS	008
DEBUG-ITEM = 000132 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000133 HORAS-EXTRAS-TRABAJO	15
DEBUG-ITEM = 000138 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000138 HORAS-TOTALES-LISTADO	080
DEBUG-ITEM = 000127 NUMERO-EMPLEADOS	010
DEBUG-ITEM = 000132 HORAS-TRABAJO-REGULAR	03
DEBUG-ITEM = 000133 HORAS-EXTRAS-TRABAJO	00
DEBUG-ITEM = 000138 HORAS-TRABAJO-REGULAR	03
DEBUG-ITEM = 000138 HORAS-TOTALES-LISTADO	006
DEBUG-ITEM = 000127 NUMERO-EMPLEADOS	012
DEBUG-ITEM = 000132 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000133 HORAS-EXTRAS-TRABAJO	10
DEBUG-ITEM = 000138 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000138 HORAS-TOTALES-LISTADO	080

Fig. 9-11

Un modo de depurar el programa sería el siguiente:

1. Comenzamos con el contador del número de empleados. Utilizamos la salida de depuración producida por la instrucción EXHIBIT DEBUG-ITEM ON ALL REFERENCES OF NUMERO-DE-EMPLEADOS. Se observa que NUMERO-DE-EMPLEADOS se pone a cero en la línea 97.
2. La siguiente referencia a NUMERO-DE-EMPLEADOS aparece en la línea 127, donde toma el valor 002. ¿Cómo salta de 000 a 002 sin pasar por 001?
3. Examinando la linea 127, encontramos:

ADD 2 TO NUMERO-DE-EMPLEADOS

En esta línea se incrementa el contador cada vez que se da entrada a un registro de empleado; obviamente debe sumarse 1 y no 2. Por tanto, se tiene un sencillo error de escritura en el programa COBOL. Observe, sin embargo, que este error no es detectado por el compilador (la sintaxis de la línea 127 es correcta), y el programa tampoco aborta. Este error debe ser encontrado por el programador durante la comprobación del programa.

4. Pasamos ahora a la columna de las horas totales. Nuestra salida de depuración nos muestra el contenido de los campos HORAS-TRABAJO-REGULAR y HORAS-EXTRAS-TRABAJO durante la ejecución de las instrucciones MOVE de las líneas 132 y 133. Por ejemplo, para el primer empleado las horas regulares son 40 y las horas extras son 3.
5. La siguiente referencia a HORAS-TRABAJO-REGULAR se produce en la línea 138, en la cual también se hace referencia a HORAS-TOTALES-LISTADO, que no es el campo apropiado. Observe que en la línea 138 no se utiliza HORAS-EXTRAS-TRABAJO, sino que se usa HORAS-TOTALES-LISTADO, cuyo contenido es siempre dos veces el valor de HORAS-TRABAJO-REGULAR.
6. Examinando la instrucción de la línea 138 del programa encontramos

ADD HORAS-TRABAJO-REGULAR HORAS-TRABAJO-REGULAR  
GIVING HORAS-TOTALES-LISTADO

en lugar de lo que obviamente se pretende, es decir,

ADD HORAS-TRABAJO-REGULAR HORAS-EXTRAS-TRABAJO  
GIVING HORAS-TOTALES-LISTADO

Este es otro error que no hará que el programa aborte.

- 9.19 Depure el siguiente programa, que contiene un bucle sin fin. El listado fuente, la salida de comprobación y la salida de depuración aparecen en las Figuras 9-12, 9-13 y 9-14, respectivamente.

```

00001      IDENTIFICATION DIVISION.
00002
00003      PROGRAM-ID. TIMELIST.
00004
00005      AUTHOR. LARRY NEWCOMER.
00006      INSTALLATION. PENN STATE UNIVERSITY--YORK CAMPUS.
00007
00008      DATE-WRITTEN. MAY 1983.
00009      DATE-COMPILED. MAY 9,1983.
00011      * TIMELIST IMPRIME UNA LINEA POR CADA EMPLEADO MOSTRANDO:
00012      * NUMERO DE HORAS REGULARES TRABAJADAS, NUMERO DE HORAS EXTRAS
00013      * TRABAJADAS Y TOTAL HORAS TRABAJADAS. AL FINAL DEL INFORME,
00014      * IMPRIME UN CONTADOR DEL NUMERO DE EMPLEADOS PROCESADOS.
00015      * LA ENTRADA SE REALIZA CON UN BLOQUE DE TARJETAS,
00016      * ORDENADAS POR NOMBRE DEL EMPLEADO.

00018      ENVIRONMENT DIVISION.

00020      CONFIGURATION SECTION.

00022      SOURCE-COMPUTER. IBM-3081 WITH DEBUGGING MODE.
00023      OBJECT-COMPUTER. IBM-3081.

00025      INPUT-OUTPUT SECTION.

00027      FILE-CONTROL.

00029          SELECT FICHERO-TARJETAS      ASSIGN TO CARDS.
00030          SELECT FICHERO-IMPRESION    ASSIGN TO PRINTER.

00032      DATA DIVISION.

00034      FILE SECTION.

00036          FD  FICHERO-TARJETAS
00037          RECORD CONTAINS 80 CHARACTERS
00038          LABEL RECORDS ARE OMITTED
00039
00040          01  TARJETA-TIEMPO-ENTRADA      PIC X(80).
00041
00043          FD  FICHERO-IMPRESION
00044          RECORD CONTAINS 132 CHARACTERS
00045          LABEL RECORDS ARE OMITTED
00046
00047          01  LINEA-IMPRESION          PIC X(132).
00048
00050      WORKING-STORAGE SECTION.

00052          01  INTER-FIN-TARJETAS      PIC X(3).
00054          01  TARJETA-TIEMPO-TRABAJO.
00055
00056          05  NOMBRE-TRABAJO          PIC X(20).
00057          05  HORAS-TRABAJO-REGULAR    PIC 99.
00058          05  HORAS-EXTRAS-TRABAJO    PIC 99.

```

Fig. 9-12

```

00059      05 FILLER          PIC X(56).

00061 01 LINEA-LISTADO-TIEMPO.

00062
00063      05 NOMBRE-LISTADO    PIC X(20).
00064      05 FILLER           PIC X(5)   VALUE SPACES.
00065      05 HORAS-REGULARES-LISTADO  PIC 99.
00066      05 FILLER           PIC X(5)   VALUE SPACES.
00067      05 HORAS-EXTRAS-LISTADO  PIC 99.
00068      05 FILLER           PIC X(5)   VALUE SPACES.
00069      05 HORAS-TOTALES-LISTADO  PIC 999.
00070      05 FILLER           PIC X(85)  VALUE SPACES.

00072 01 LINEA-TOTAL-EMPLEADO.

00073
00074      05 FILLER           PIC X(25)
00075                      VALUE "NUM. DE EMPLEADOS ES".
00076      05 NUMERO-EMPLEADOS  PIC 999.
00077      05 FILLER           PIC X(104)  VALUE SPACES.

00079 PROCEDURE DIVISION.

00081 DECLARATIVES.

00082
00083 COMPROBACION-LOGICA SECTION.

00084
00085     USE FOR DEBUGGING
00086         ON ALL REFERENCES OF NUMERO-EMPLEADOS
00087             ALL REFERENCES OF HORAS-TRABAJO-REGULAR
00088             ALL REFERENCES OF HORAS-EXTRAS-TRABAJO
00089             ALL REFERENCES OF HORAS-TOTALES-LISTADO

00090
00091     EXHIBIT NAMED DEBUG-ITEM
00092
00093
00094 END DECLARATIVES.

00096 D READY TRACE
00097
00098     MOVE "NO " TO INTER-FIN-TARJETAS
00099     MOVE ZERO TO NUMERO-EMPLEADOS

00100
00101     OPEN   INPUT          FICHERO-TARJETAS
00102         OUTPUT          FICHERO-IMPRESION
00103
00104     PERFORM ENTRADA-REGISTRO
00105
00106     PERFORM PRODUCE-LISTADO-TIEMPO
00107         UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"
00108
00109     WRITE LINEA-IMPRESION
00110         FROM LINEA-TOTAL-EMPLEADO
00111
00112     CLOSE   FICHERO-TARJETAS
00113         FICHERO-IMPRESION
00114     STOP RUN
00115

00117 ENTRADA-REGISTRO.
00118
00119     READ FICHERO-TARJETAS
00120         INTO TARJETA-TIEMPO-TRABAJO
00121         AT END
00122         MOVE "SI" TO INTER-FIN-TARJETAS
00123

```

Fig. 9-12 (cont.)

```

00125      PRODUCE-LISTADO-TIEMPO.
00126
00127      *      INCREMENTA CONTADOR UNO POR EMPLEADOS PROCESADOS
00128
00129          ADD 2 TO NUMERO-EMPLEADOS
00130
00131      *      COPIA LA TARJETA EN EL AREA DE CONSTRUCCION
00132
00133          MOVE NOMBRE-TRABAJO      TO NOMBRE-LISTADO
00134          MOVE HORAS-TRABAJO-REGULAR  TO HORAS-REGULARES-LISTADO
00135          MOVE HORAS-EXTRAS-TRABAJO   TO HORAS-EXTRAS-LISTADO
00136
00137      *      CALCULA TOTAL HORAS SUMANDO LAS NORMALES
00138      *      Y LAS EXTRAS.
00139
00140          ADD HORAS-TRABAJO-REGULAR HORAS-EXTRAS-TRABAJO
00141                  GIVING HORAS-TOTALES-LISTADO
00142
00143          WRITE LINEA-IMPRESION
00144                  FROM LINEA-LISTADO-TIEMPO
00145
00146

```

Fig. 9-12 (cont.)

ABEL FRED	40	03	080
ABEL FRED	40	03	080
ABEL FRED	40	03	080
ABEL FRED	40	03	080

Fig. 9-13

DEBUG-ITEM = 000099 NUMERO-EMPLEADOS	000
ENTRADA-REGISTRO , PRODUCE-LISTADO-TIEMPO ,	
DEBUG-ITEM = 000129 NUMERO-EMPLEADOS	002
DEBUG-ITEM = 000134 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000135 HORAS-EXTRAS-TRABAJO	03
DEBUG-ITEM = 000140 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000140 HORAS-TOTALES-LISTADO	080
PRODUCE-LISTADO-TIEMPO ,	
DEBUG-ITEM = 000129 NUMERO-EMPLEADOS	004
DEBUG-ITEM = 000134 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000135 HORAS-EXTRAS-TRABAJO	03
DEBUG-ITEM = 000140 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000140 HORAS-TOTALES-LISTADO	080
PRODUCE-LISTADO-TIEMPO ,	
DEBUG-ITEM = 000129 NUMERO-EMPLEADOS	006
DEBUG-ITEM = 000134 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000135 HORAS-EXTRAS-TRABAJO	03
DEBUG-ITEM = 000140 HORAS-TRABAJO-REGULAR	40
DEBUG-ITEM = 000140 HORAS-TOTALES-LISTADO	080

Fig. 9-14

El procedimiento para la depuración sería:

1. Examen de la salida de comprobación: parece que el primer registro se imprime una y otra vez.
2. Examen de la salida de depuración, que contiene una instrucción READY TRACE y otra instrucción DISPLAY DEBUG-ITEM ON ALL REFERENCES OF NUMERO-DE-EMPLEADOS, HORAS-TRABAJO-REGULAR, HORAS-EXTRAS-TRABAJO y LISTADO-HORAS-TOTALES. Al examinar la trayectoria encontramos:

ENTRADA-REGISTRO, PRODUCE-LISTADO-TIEMPO,  
PRODUCE-LISTADO-TIEMPO,  
PRODUCE-LISTADO-TIEMPO,

...

Obviamente se está repitiendo la ejecución de PRODUCE-LISTADO-TIEMPO.

3. Al examinar el contenido de DEBUG-ITEM en cada ejecución de PRODUCE-LISTADO-TIEMPO, encontramos que HORAS-TRABAJO-REGULAR y HORAS-EXTRAS-TRABAJO nunca cambian, lo que confirma que se está procesando el mismo registro una y otra vez.
4. Al examinar el programa, encontramos que las líneas 106 y 107 contienen

PERFORM PRODUCE-LISTADO-TIEMPO  
UNTIL INTER-FIN-TARJETAS IS EQUAL TO "SI"

5. Comprobando las líneas 125-146, encontramos que PRODUCE-LISTADO-TIEMPO no cambia INTER-FIN-TARJETAS ni ejecuta ningún otro párrafo que lo altere. O sea, no hay forma de que PERFORM...UNTIL... finalice. También se pone de manifiesto que PRODUCE-LISTADO-TIEMPO procesa siempre el mismo registro porque no da entrada a nuevos registros lógicos.
6. Ambos problemas se pueden corregir añadiendo la instrucción

PERFORM ENTRADA-REGISTRO

al final del párrafo PRODUCE-LISTADO-TIEMPO. ENTRADA-REGISTRO alterará el valor de INTER-FIN-TARJETAS al final del fichero y también dará entrada al siguiente registro lógico para la ejecución de PRODUCE-LISTADO-TIEMPO que sigue.

Observe que el programa contiene los fallos que ya se corrigieron en el Problema 9.18.

que es un *entero positivo* (en forma de literal o variable numéricos) que se escribe entre paréntesis a continuación del nombre de la tabla.

**EJEMPLO 10.2** En el siguiente programa se emplean cuatro subíndices para sumar los elementos de una tabla.

```

01 VENTAS-TRIMESTRALES.
  05 TOTAL-TRIMESTRE      PIC S9(7)V99      COMP-3.
                                         OCCURS 4 TIMES.

.
.
.
PROCEDURE DIVISION.

.
.
.

ADD TOTAL-TRIMESTRE (1)
TOTAL-TRIMESTRE (2)
TOTAL-TRIMESTRE (3)
TOTAL-TRIMESTRE (4)          GIVING TOTAL-AÑO

```

La verdadera utilidad de las tablas y los subíndices estriba en que se puede utilizar una *variable entera* como subíndice, de forma que el valor de dicha variable indica el elemento de la tabla que se está procesando. Con frecuencia la variable subíndice se controla por medio de una instrucción **PERFORM...VARYING**.

**EJEMPLO 10.3**

```

01 TOTAL-AREAS.
  05 TOTAL-AÑO-PASADO      PIC S9(9)V99      COMP-3.
  05 TOTAL-ESTE-AÑO       PIC S9(9)V99      COMP-3.

.
.
.

01 AREAS-SUBINDICES.
  05 MES                   PIC S9(2)        COMP SYNC.

.
.
.

01 INFORMACION-VENTAS.
  05 VENTAS-MENSUALES     OCCURS 12 TIMES.
    10 AÑO-PASADO         PIC S9(7)V99      COMP-3.
    10 ESTE-AÑO           PIC S9(7)V99      COMP-3.

.
.
.

MOVE ZERO TO TOTAL-AÑO-PASADO
      TOTAL-ESTE-AÑO
PERFORM CALCULO-TOTALES-AÑO
  VARYING MES FROM 1 BY 1
  UNTIL MES GREATER THAN 12

.
.
.

CALCULO-TOTALES-AÑO.
  ADD AÑO PASADO (MES) TO TOTAL-AÑO-PASADO
  ADD ESTE AÑO (MES) TO TOTAL-ESTE-AÑO

```

En este caso, a partir de una tabla de 12 elementos compuestos denominados VENTAS-MENSUALES, se suman separadamente los datos elementales AÑO-PASADO y ESTE-AÑO utilizando como subíndice la variable MES.

En el COBOL del sistema IBM OS/VS, la forma más eficiente de definir un elemento numérico que vaya a ser utilizado como subíndice es con PIC S9(n) COMP SYNC. Los subíndices son necesarios para todos los elementos de una cláusula OCCURS.

### 10.3 MANIPULACION DE TABLAS UNIDIMENSIONALES

#### Inicialización con ceros de una tabla

Con frecuencia, se desea inicializar todos los elementos de una tabla con el mismo valor; digamos cero. Dado que no se puede utilizar la cláusula VALUE con elementos que formen parte de una tabla, es preciso recurrir a la instrucción MOVE para la inicialización.

#### EJEMPLO 10.4 Definida una tabla de 50 elementos mediante

```
01 CIFRAS-VENTAS-POR-ESTADOS.
  05 VENTAS-ESTADO          OCCURS 50 TIMES.
  10 ID-ESTADO              PIC X(2).
  10 IMPORTE-VENTAS        PIC S9(5)V99  COMP-3.
```

indicaremos un modo incorrecto y otro correcto de inicializar la tabla con ceros.

- MOVE ZEROS TO CIFRAS-VENTAS-POR-ESTADOS

Válido, puesto que CIFRAS-VENTAS-POR-ESTADOS no es un elemento de la tabla (no se describe con una cláusula OCCURS ni tampoco está subordinado a una cláusula OCCURS) y no necesita subíndice. Pero como se trata de un elemento compuesto, en la tabla entrarán ceros en formato PIC X DISPLAY; en particular, en los 50 elementos IMPORTE-VENTAS, que está en formato COMP-3. Por tanto, tendrá lugar un error lógico.

- PERFORM INICIALIZA-IMPORTE-VENTAS  
VARYING SUB-VENTAS-ESTADO  
FROM 1 BY 1  
UNTIL SUB-VENTAS-ESTADO GREATER THAN 50

```
INICIALIZA-IMPORTE-VENTAS.  
MOVE ZERO TO IMPORTE-VENTAS (SUB-VENTAS-ESTADO)  
MOVE SPACES TO ID-ESTADO (SUB-VENTAS-ESTADO)
```

El procedimiento correcto consiste en definir un subíndice (SUB-VENTAS-ESTADO) mediante el cual se inicialicen todos los campos de la tabla individualmente en un párrafo que se ejecutará mediante PERFORM.

#### EJEMPLO 10.5

```
01 TABLA-DATOS-EMPLEADO.
  05 INFORMACION-EMPLEADOS OCCURS 500 TIMES.
  10 NUM-SEG-SOCIAL      PIC 9(9).
  10 SALARIO-EMPLEADO    PIC S9(3)V99.
```

Como todos los campos de la tabla están en formato DISPLAY, se puede inicializar la tabla con una única instrucción:

```
MOVE ZEROS TO TABLA-DATOS-EMPLEADOS
```

#### Inicialización mediante la cláusula VALUE

Si cada elemento de la tabla se tiene que inicializar con un valor distinto, se puede utilizar la cláusula VALUE con una serie de elementos individuales que después se redefinen (REDEFINED) como una tabla. El método indirecto se debe a que no se puede utilizar la cláusula VALUE con los elementos de una tabla. Este procedimiento es válido sólo con elementos del almacenamiento de trabajo.

#### EJEMPLO 10.6

```
01 TABLA-VALORES-COD-ESTADO.
  05 VALORES-VALIDO-ESTADO.
    10 FILLER           PIC X(3)  VALUE "A07".
    10 FILLER           PIC X(3)  VALUE "A2K".
    10 FILLER           PIC X(3)  VALUE "B03".
    10 FILLER           PIC X(3)  VALUE "C99".
  05 CODIGO-ESTADO      REDEFINES VALORES-VALIDO-ESTADO
                        PIC X(3)
                        OCCURS 4 TIMES.
```

VALORES-VALIDOS-ESTADO contiene cuatro elementos simples, en formato PIC X(3), que se inicializan mediante la cláusula VALUE. Obsérvese que no se dan nombres a estos elementos, en su lugar se utiliza la palabra reservada FILLER. A continuación se redefine como tabla VALORES-VALIDOS-ESTADO con el nombre CODIGO-ESTADO, de manera que los cuatro elementos PIC X(3) de VALORES-VALIDOS-ESTADO se identifican (ocupan las mismas posiciones de memoria). En otras palabras, CODIGO-ESTADO (1) tiene el valor inicial "A07", CODIGO-ESTADO (2) tiene el valor "A2K", etc. Las instrucciones que procesen la tabla deberán referirse a CODIGO-ESTADO y no a VALORES-VALIDOS-ESTADO.

**EJEMPLO 10.7** El Ejemplo 10.6 utiliza la cláusula VALUE para inicializar los elementos de una tabla; ahora vamos a inicializar una tabla con elementos compuestos.

```

01 TABLA-INSTRUCCIONES-ENVIO.
05 VALORES-CODIGO-ENVIO.
  10 FILLER          PIC X(2)      VALUE "A3".
  10 FILLER          PIC X(20)     VALUE "U.S. MAIL".
  10 FILLER          PIC X(2)      VALUE "A8".
  10 FILLER          PIC X(20)     VALUE "UPS".
  10 FILLER          PIC X(2)      VALUE "C3".
  10 FILLER          PIC X(20)     VALUE "FEDERAL EXPRESS".
05 CODIGOS-E-INSTRUCCIONES  REDEFINES VALORES-CODIGO-ENVIO
                           OCCURS 3 TIMES.
  10 CODIGO-ENVIO    PIC X(2).
  10 INSTRUCCIONES-ENVIO  PIC X(20).

```

De nuevo, los elementos de la tabla se definen primero como elementos simples con el nombre FILLER, los formatos PIC y USAGE apropiados y mediante cláusulas VALUE. Observe que VALORES-CODIGO-ENVIO consiste en un elemento PIC X(2) seguido por un elemento PIC X(20) y por otro elemento PIC X(2), etc. Después CODIGOS-E-INSTRUCCIONES redefine VALORES-CODIGO-ENVIO como una tabla de 3 elementos, cada uno de los cuales consiste en un CODIGO-ENVIO en formato PIC X(2) seguido de INSTRUCCIONES-ENVIO en formato PIC X(20). Como consecuencia de las cláusulas VALUE, CODIGO-ENVIO (1) toma el valor inicial "A3" e INSTRUCCIONES-ENVIO (1) toma el valor inicial "U.S. MAIL". Del mismo modo, CODIGO-ENVIO (2) toma el valor inicial "A8" e INSTRUCCIONES-ENVIO (2) toma el valor inicial "UPS"; etcétera. En el Ejemplo 10.20 se aplica esta técnica a una tabla de elementos numéricos.

#### Carga mediante la instrucción READ

Hay dos inconvenientes en la técnica de inicializar las tablas de almacenamiento temporal aplicando REDEFINE a elementos que disponen de cláusula VALUE: (1) conforme las tablas son mayores y/o más complicadas, este método llega a ser pesado y requiere una porción importante de la división de datos; (2) si se tuvieran que cambiar los valores iniciales de la tabla, se tiene que cambiar y recomilar todo el programa COBOL. Una forma de evitar esto consiste en conservar los valores iniciales en un fichero secuencial en cinta o disco. Cuando se comienza la ejecución del programa, se pueden leer estos valores mediante READ y así transferirlos a las áreas dedicadas a la tabla. Si se tuvieran que cambiar los valores iniciales, sólo se tendría que retocar este fichero.

**EJEMPLO 10.8** La tabla de códigos de estado del Ejemplo 10.6 se puede inicializar desde un fichero del siguiente modo:

```

FD FICHERO-CODIGO-ESTADO...
01 REGISTRO-CODIGO-ESTADO.
  05 VALOR-CODIGO-ESTADO  PIC X(3).
  .....
WORKING-STORAGE SECTION.
  .....
01 TABLA-CODIGOS-ESTADO.
  05 CODIGO-ESTADO        PIC X(3)      OCCURS 400 TIMES.
  05 NUMERO-DE-CODIGOS   PIC S9(3)    COMP SYNC.
  .....

```

```

PROCEDURE DIVISION.
  OPEN INPUT FICHERO-CODIGO-ESTADO
  MOVE "NO" TO INTER-FIN-FICHERO-ESTADO
  PERFORM CARGA-TABLA-ESTADO
    VARYING NUMERO-DE-CODIGOS FROM 1 BY 1
    UNTIL     INTER-FIN-FICHERO-ESTADO EQUAL "SI"
              OR      NUMERO-DE-CODIGOS GREATER THAN 400
    IF NUMERO-DE-CODIGOS GREATER THAN 400
      MOVE 400 TO NUMERO-DE-CODIGOS
      DISPLAY "TABLA MUY LARGA-RECOMPIRAR"
      UPON DISPOSITIVO-MENSAJE-OPERADOR
    ELSE
      SUBTRACT 2 FROM NUMERO-DE-CODIGOS
    CLOSE FICHERO-CODIGO-ESTADO
    .....
  CARGA-TABLA-ESTADO.
  READ FICHERO-CODIGO-ESTADO
  AT END
    MOVE "SI" TO INTER-FIN-FICHERO-ESTADO
  IF INTER-FIN-FICHERO-ESTADO EQUAL "NO"
    MOVE VALOR-CODIGO-ESTADO TO CODIGO-ESTADO (NUMERO-DE-CODIGOS)

```

Observe cómo se utiliza NUMERO-DE-CODIGOS para contar el número de elementos de una tabla que se dan entrada desde el fichero. También dése cuenta de que el bucle PERFORM...VARYING que da entrada a los valores del código de entráda se detiene el alcanzar el final del fichero o cuando NUMERO-DE-CODIGOS supera el tamaño de la tabla tal y como se define en la cláusula OCCURS (400 en este caso). A continuación de PERFORM...VARYING... una instrucción IF comprueba NUMERO-DE-CODIGOS, y si se ha superado el tamaño de la tablá, se escribe un mensaje para el operador en el que se indica que debe modificarse la cláusula OCCURS y recompilar el programa. Observe que si se alcanza el fin de fichero, el último valor de NUMERO-DE-CODIGOS corresponde al final del fichero y no a VALOR-CODIGO-ESTADO. Por ello, NUMERO-DE-CODIGOS debe disminuirse en dos unidades al concluir PERFORM...VARYING... En todo caso, NUMERO-DE-CODIGOS indica correctamente el número de elementos de la tabla cargados durante la ejecución del programa.

El mantenimiento de un contador del número de elementos de una tabla es una técnica muy corriente.

### Suma de elementos de una tabla

Además del Ejemplo 10.3 daremos otro más.

**EJEMPLO 10.9** Suponga que desea acumular la suma de los salarios de los empleados en la tabla del Ejemplo 10.5:

```

01 TABLA-DATOS-EMPLEADOS.
  05 INFORMACION-EMPLEADOS          OCCURS 500 TIMES.
    10 NUM-SEG-SOCIAL             PIC 9(9).
    10 SALARIO-EMPLEADO           PIC S9(3)V99   COMP-3.
  05 NUMERO-EMPLEADOS-ACTIVOS    PIC S9(3)       COMP.
  .....
  MOVE ZERO TO TOTAL-SALARIOS
  PERFORM TOTAL-SALARIOS-EMPLEADOS
    VARYING SUB-EMPLEADO
    FROM 1 BY 1
    UNTIL SUB-EMPLEADO GREATER THAN NUMERO-EMPLEADOS-ACTIVOS
  .....
  TOTAL-SALARIOS-EMPLEADOS.
  ADD SALARIO-EMPLEADOS (SUB-EMPLEADO) TO TOTAL-SALARIOS

```

No hay necesidad de sumar los 500 elementos, porque aunque la tabla está preparada para acomodar 500 empleados, es posible que contenga menos elementos. El elemento NUMERO-EMPLEADOS-ACTIVOS (que no forma parte de tabla alguna) se utiliza para contar el número de empleados de la tabla. Se asume que las rutinas que añaden o borran elementos de la tabla modifican convenientemente este contador. El campo NUMERO-EMPLEADOS-ACTIVOS es el valor de parada de la instrucción PERFORM en la que se suman los salarios de los empleados.

### Examen de tablas con PERFORM

En una operación de *examen* o *búsqueda*, se comparan los elementos de una tabla, de forma sistemática, con un valor hasta que se encuentra el elemento que coincide con el valor dado o hasta que se concluye que el valor buscado no está en la tabla. El algoritmo más simple de búsqueda se denomina *búsqueda secuencial* o *lineal* y en él se examina el primer elemento, después el segundo, a continuación el tercero, etc., hasta que se encuentre el elemento deseado o se alcanza el final de la tabla. La mayor parte de las versiones de COBOL disponen de la instrucción SEARCH para la búsqueda secuencial (véase la Sección 10.8). Sin embargo, también podría hacerse con un bucle PERFORM.

**EJEMPLO 10.10** Dada la tabla del Ejemplo 10.6 y un valor determinado de CODIGO-ESTADO-ENTRADA PIC X(3), indicar si es válido, examinando para ello la TABLA-CODIGOS-ESTADO.

```

01 TABLA-CODIGOS-ESTADO.
   05 CODIGO-ESTADO          PIC X(3)
                                OCCURS 40 TIMES.

CODIGO-ESTADO-COMP.
   PERFORM EXAMEN-ESTADO
   IF INTER-ESTADO-VALIDO EQUAL "SI"
      ...(procesamiento código estado válido)
   ELSE
      ...(manejo código estado inválido)
   .....

EXAMEN-ESTADO.
   MOVE "NO" TO INTER-ESTADO-VALIDO
   PERFORM VALIDACION-ENTRADA
      VARYING SUB-CODIGO-ESTADO FROM 1 BY 1
      UNTIL   SUB-CODIGO-ESTADO GREATER THAN 40
              OR INTER-ESTADO-VALIDO EQUAL "SI"

VALIDACION-ENTRADA.
   IF CODIGO-ESTADO (SUB-CODIGO-ESTADO) EQUAL CODIGO-ESTADO-ENTRADA
      MOVE "SI" TO INTER-ESTADO-VALIDO

```

El párrafo EXAMEN-ESTADO inicializa INTER-ESTADO-VALIDO con "NO". Después se ejecuta VALIDACION-ENTRADA haciendo variar el subíndice SUB-CODIGO-ESTADO [PIC S9(5) COMP SYNC] desde 1 hasta el número de elementos de la tabla. Si en algún instante el elemento correspondiente a SUB-CODIGO-ESTADO es igual a CODIGO-ESTADO-ENTRADA, el interruptor INTER-ESTADO-VALIDO toma el valor "SI" y así se detiene la ejecución de PERFORM...UNTIL... Si ningún elemento de la tabla resulta ser igual que CODIGO-ESTADO-ENTRADA, entonces INTER-ESTADO-VALIDO continúa valiendo "NO".

Observe que si, por ejemplo, el sexto elemento es igual que CODIGO-ESTADO-ENTRADA, la instrucción PERFORM... se detiene siendo SUB-CODIGO-ESTADO igual a *siete* (véase Fig. 7-14). Esto puede remediarse del modo siguiente:

```

EXAMEN-ESTADO.
   MOVE 1 TO SUB-CODIGO-ESTADO
   MOVE "SI" TO INTER-ESTADO-VALIDO
   PERFORM VALIDACION-ENTRADA
      UNTIL INTER-ESTADO-VALIDO EQUAL "NO"
              OR CODIGO-ESTADO (SUB-CODIGO-ESTADO) EQUAL
                  CODIGO-ESTADO-ENTRADA

```

```

VALIDACION-ENTRADA.
IF SUB-CODIGO-ESTADO EQUAL 40
    MOVE "NO" TO INTER-ESTADO-VALIDO
ELSE
    ADD 1 TO SUB-CODIGO-ESTADO

```

Ahora si en algún momento SUB-CODIGO-ESTADO apunta hacia un elemento igual a CODIGO-ESTADO-ENTRADA, la condición de UNTIL llega a ser cierta y la ejecución de PERFORM se detiene de inmediato sin incrementar SUB-CODIGO-ESTADO, que continúa señalando el elemento de la tabla deseado.

**EJEMPLO 10.11** Una razón común para la búsqueda en una tabla es la *decodificación* de campos de entrada codificados cuando se están generando informes impresos. Vamos a ilustrar esto con una tabla similar a la del Ejemplo 10.7:

```

01 TABLA-INSTRUCCIONES-ENVIO.
05 CODIGOS-E-INSTRUCCIONES      OCCURS 18 TIMES.
    10 CODIGO-ENVIO              PIC X(2).
    10 INSTRUCCIONES-ENVIO       PIC X(30).

MOVE "SI" TO INTER-CODIGO-ENCONTRADO
MOVE 1 TO SUB-ENVIO
PERFORM VALIDACION-ENTRADA
    UNTIL      CODIGO-ENVIO (SUB-ENVIO) EQUAL CODIGO-ENTRADA
               OR INTER-CODIGO-ENCONTRADO EQUAL "NO"
IF INTER-CODIGO-ENCONTRADO EQUAL "SI"
    MOVE INSTRUCCIONES-ENVIO (SUB-ENVIO) TO
        MENSAJE-ENVIO-IMPRESO
ELSE
    MOVE "**** CODIGO ENVIO INVALIDO ****" TO
        MENSAJE-ENVIO-IMPRESO
;

VALIDACION-ENTRADA.
IF SUB-ENVIO EQUAL 18
    MOVE "NO" TO INTER-CODIGO-ENCONTRADO
ELSE
    ADD 1 TO SUB-ENVIO

```

Si se encuentra CODIGO-ENTRADA en TABLA-INSTRUCCIONES-ENVIO, se traslada al área de salida el campo INSTRUCCIONES-ENVIO correspondiente al CODIGO-ENVIO que coincidió con CODIGO-ENTRADA. Si CODIGO-ENTRADA no estuviera en la tabla, se imprimiría un mensaje de error. Mediante esta técnica, los registros de entrada contienen tan sólo un código de 2 bytes que se transforma, sin embargo, en unas instrucciones de envío de 30 bytes en el informe impreso.

#### 10.4 TABLAS BIDIMENSIONALES

Cuando un elemento que contiene en su definición una cláusula OCCURS tiene subordinado otro elemento que contiene otra cláusula OCCURS, se dice que la tabla subordinada es *bidimensional*.

**EJEMPLO 10.12**

```

01 TABLA-FLETES.
05 RANGO-PESOS      OCCURS 3 TIMES.
    10 LIMITE-PESO     PIC S9(3)          COMP-3.
    10 DESTINO         OCCURS 3 TIMES.
    15 FLETE          PIC S9(3)V99     COMP-3.

```

RANGO-PESOS es una tabla con 3 elementos, cada uno de los cuales contiene los campos LIMITE-PESO y DESTINO. A su vez, DESTINO es una tabla con 3 elementos (cada uno de los cuales contiene un valor de FLETE). De este modo la tabla DESTINO contiene  $3 \times 3 = 9$  elementos.

RANGO-PESOS (1)	LIMITE-PESO (1)	2 bytes, S9(3)	COMP-3
	FLETE (1, 1)	3 bytes, S9(3)V99	COMP-3
	FLETE (1, 2)	3 bytes, S9(3)V99	COMP-3
	FLETE (1, 3)	3 bytes, S9(3)V99	COMP-3
RANGO-PESOS (2)	LIMITE-PESO (2)	2 bytes, S9(3)	COMP-3
	FLETE (2, 1)	3 bytes, S9(3)V99	COMP-3
	FLETE (2, 2)	3 bytes, S9(3)V99	COMP-3
	FLETE (2, 3)	3 bytes, S9(3)V99	COMP-3
RANGO-PESOS (3)	LIMITE-PESO (3)	2 bytes, S9(3)	COMP-3
	FLETE (3, 1)	3 bytes, S9(3)V99	COMP-3
	FLETE (3, 2)	3 bytes, S9(3)V99	COMP-3
	FLETE (3, 3)	3 bytes, S9(3)V99	COMP-3

Fig. 10-1

En la Figura 10.1 se observa que el número de subíndices precisos para referirse a un elemento de la tabla es igual al número de cláusulas OCCURS que se aplican al elemento. El primer índice se refiere a la cláusula OCCURS de mayor nivel, el segundo a la del siguiente nivel, etc. En esta tabla bidimensional, los dos subíndices tienen el mismo rango de valores, caso particular que no tiene porque suceder. En la Figura 10.1 los subíndices múltiples se separan por una coma y un espacio; la coma es opcional.

**EJEMPLO 10.13** Si el registro EXPEDIENTE-ESTUDIANTE, definido abajo, corresponde al registro lógico de un fichero maestro de estudiantes, ¿cuál es la longitud del registro lógico?

```

01 EXPEDIENTE-ESTUDIANTE.
  05 ID-ESTUDIANTE          PIC X(9).
  05 NOMBRE-ESTUDIANTE      PIC X(25).
  05 SEMESTRES              PIC S9          COMP-3.
  05 INFORMACION-SEMESTRE   OCCURS 8 TIMES.
    10 NOTA-MEDIA            PIC S9V99        COMP-3.
    10 NUMERO-DE-CURSOS     PIC S9          COMP-3.
    10 INFO-CURSO            OCCURS 6 TIMES.
      15 ID-CURSO             PIC X(5).
      15 NUMERO-ASIGNATURAS  PIC S9V9         COMP-3.
      15 GRADO                 PIC S9          COMP-3.

```

ID-CURSO tiene 5 bytes, NUMERO-ASIGNATURAS tiene 2 bytes (Sección 5.18) y GRADO tiene 1 byte; por tanto, cada elemento de INFO-CURSO tiene 8 bytes. Dado INFO-CURSO OCCURS 6 TIMES, la tabla INFO-CURSO completa tiene 48 bytes. El campo INFORMACION-SEMESTRE contiene una tabla INFO-CURSO completa junto con NUMERO-DE-CURSOS (1 byte) y NOTA-MEDIA (2 bytes); por tanto, un elemento de INFORMACION-SEMESTRE tiene 51 bytes. Puesto que hay 8 elementos, la tabla completa de INFORMACION-SEMESTRE contiene 408 bytes. Ahora hay que sumar el espacio de ID-ESTUDIANTE (9 bytes), el de NOMBRE-ESTUDIANTE (25 bytes) y SEMESTRES (1 byte), todo ello da un total de 443 bytes.

Debe tenerse en cuenta que la definición de EXPEDIENTE-ESTUDIANTE presupone una carga de la tabla tal que INFO-CURSO (i, j) se refiera al curso j-ésimo *realizado por el estudiante* durante el i-ésimo semestre.

(Si la tabla apareciera en forma matricial, los datos del último semestre aparecerían en la última fila.) El número de semestres realizados por el estudiante se almacenará en el contador SEMESTRES y el contador NUMERO-DE-CURSOS (k) almacenará el número de cursos realizados por el estudiante durante el  $k$ -ésimo semestre. Estos contadores indican cuáles de las  $8 \times 6 = 48$  posiciones de memoria de INFO-CURSO contienen datos reales.

**EJEMPLO 10.14** Calcule el rendimiento medio semestral de un estudiante cuya información aparezca en el registro lógico EXPEDIENTE-ESTUDIANTE del Ejemplo 10.13.

```

MOVE ZERO TO NOTA-TOTAL
PERFORM MEDIAS-SEMESTRALES
    VARYING SUB-SEMESTRE FROM 1 BY 1
    UNTIL SUB-SEMESTRE GREATER THAN SEMESTRES
    DIVIDE SEMESTRES INTO NOTA-TOTAL
    GIVING MEDIA-SEMESTRAL
.
.
.
MEDIAS-SEMESTRALES
ADD NOTA-MEDIA (SUB-SEMESTRE) TO NOTA-TOTAL

```

## 10.5 MANIPULACION DE TABLAS BIDIMENSIONALES

### Inicialización a cero

Como hay dos subíndices se utiliza la instrucción PERFORM...VARYING...AFTER...

**EJEMPLO 10.15** En lo que sigue, se asume que los seis elementos de una tabla son activos.

```

01 IMPORTES-VENTAS.
    05 TOTALES-REGIONES          OCCURS 2 TIMES.
    10 TOTALES-DEPARTAMENTO      OCCURS 3 TIMES.
    15 VENTAS                   PIC S9(5)V99 COMP.
.
.
.
01 :SUBINDICES.
    05 NUMERO-REGION            PIC S9(5) COMP SYNC.
    05 NUMERO-DEPARTAMENTO     PIC S9(5) COMP SYNC.
.
.
.
PERFORM INICIA-IMPORTES-A-CERO
    VARYING NUMERO-REGION FROM 1 BY 1
    UNTIL NUMERO-REGION GREATER THAN 2
    AFTER NUMERO-DEPARTAMENTO FROM 1 BY 1
    UNTIL NUMERO-DEPARTAMENTO GREATER THAN 3
.
.
.
INICIA-IMPORTES-A-CERO
MOVE ZERO TO VENTAS (NUMERO-REGION NUMERO-DEPARTAMENTO)

```

### Suma de elementos

**EJEMPLO 10.16** Calcule el número total de asignaturas aprobadas por el estudiante del Ejemplo 10.13.

```

.
.
.
05 TOTAL-ASIGNATURAS   PIC S9(3)V9   COMP-3.
.
.
MOVE ZERO TO TOTAL-ASIGNATURAS
PERFORM ACUMULA-ASIGNATURAS
    VARYING NUMERO-DE-SEMESTRE FROM 1 BY 1
    UNTIL NUMERO-DE-SEMESTRE GREATER THAN SEMESTRES
    AFTER NUMERO-DE-CURSO FROM 1 BY 1
    UNTIL NUMERO-DE-CURSO GREATER THAN
        NUMERO-DE-CURSOS (NUMERO-DE-SEMESTRE)
.
.
.
```

ACUMULA-ASIGNATURAS.

ADD NUMERO-ASIGNATURAS (NUMERO-DE-SEMESTRE NUMERO-DE-CURSO)  
TO TOTAL-ASIGNATURAS

donde los subíndices se definen así:

```
01 SUBINDICES.  
  05 NUMERO-DE-SEMESTRE    PIC S9(5) COMP SYNC.  
  05 NUMERO-DE-CURSO       PIC S9(5) COMP SYNC.
```

Observe especialmente el subíndice de NUMERO-DE-CURSOS en la instrucción PERFORM. Dado que NUMERO-DE-CURSOS OCCURS 8 TIMES forma parte de la tabla INFORMACION-SEMESTRE, debe contar con un subíndice; cada vez que se incrementa NUMERO-DE-SEMESTRE, se modifica consiguientemente el valor de NUMERO-DE-CURSOS (NUMERO-DE-SEMESTRE).

**EJEMPLO 10.17** En los Ejemplos 10.13 y 10.14 el contenido de NOTA-MEDIA, parte de la INFORMACION-SEMESTRE, se trató como si fuera un dato de entrada. Supongamos que es preciso calcular NOTA-MEDIA a partir de la tabla INFO-CURSO: esto puede hacerse para cada semestre sumando los grados (calificaciones) de las asignaturas del semestre y dividiendo el resultado por el número de asignaturas del semestre. El programa en COBOL incluirá una estructura PERFORM...VARYING... anidada, como se ve más abajo, con los subíndices definidos en el Ejemplo 10.16 y donde NOTA-MEDIA y TOTAL-ASIGNATURAS son elementos numéricos del almacenamiento temporal.

```
PERFORM CALCULA-MEDIA-SEMESTRE  
      VARYING NUMERO-DE-SEMESTRE FROM 1 BY 1  
      UNTIL NUMERO-DE-SEMESTRE GREATER THAN SEMESTRES  
  
.....  
  
CALCULA-MEDIA-SEMESTRE.  
  MOVE ZERO TO NOTA-TOTAL  
  TOTAL-ASIGNATURAS  
  PERFORM TOTAL-CURSOS-SEMESTRE FROM 1 BY 1  
    UNTIL NUMERO-DE-CURSO GREATER THAN  
      NUMERO-DE-CURSOS (NUMERO-DE-SEMESTRE)  
  DIVIDE TOTAL-ASIGNATURAS INTO NOTA-TOTAL  
    GIVING NOTA-MEDIA (NUMERO-DE-SEMESTRE)  
  
TOTAL-CURSOS-SEMESTRE.  
  ADD NUMERO-ASIGNATURAS (NUMERO-DE-SEMESTRE NUMERO-DE-CURSO)  
    TO TOTAL-ASIGNATURAS  
  COMPUTE NOTA-TOTAL = NOTA-TOTAL +  
    GRADO (NUMERO-DE-SEMESTRE NUMERO-DE-CURSO) *  
    NUMERO-ASIGNATURAS (NUMERO-DE-SEMESTRE NUMERO-DE-CURSO)
```

### Conversión de datos de entrada en subíndices

Para las tablas consideradas hasta el momento, los subíndices se han generado mediante PERFORM...VARYING... Algunas veces, sin embargo, se desean obtener los subíndices mediante conversión de datos de entrada.

La conversión más simple que puede hacerse es no efectuar conversión —caso en que los datos de entrada están apropiadamente diseñados.

**EJEMPLO 10.18** Suponga que el personal de la entrada de datos ha tecleado los siguientes campos (DISPLAY):

```
FD FICHERO-ENTRADA-VENTAS...  
01 REGISTRO-VENTAS.  
  05 ID-REGION          PIC 9(1).  
  05 ID-DEPARTAMENTO   PIC 9(2).  
  05 IMPORTE-VENTAS    PIC S9(4)V99.
```

Utilizando ID-REGION e ID-DEPARTAMENTO como subíndices, crear una tabla de las ventas totales por regiones y departamentos:

```
01 TABLA-IMPORTES-TOTAL-VENTAS.
  05 REGION          OCCURS 7 TIMES.
    10 DEPARTAMENTO   OCCURS 12 TIMES.
      15 IMPORTE-TOTAL-VENTAS  PIC S9(5)V99 COMP.
  05 LIMITE-REGION   PIC S9(2)  COMP SYNC VALUE+7.
  05 LIMITE-DEPARTAMENTO  PIC S9(2)  COMP SYNC VALUE+12.
```

El programa siguiente llevaría a cabo esta tarea (se asume que los elementos de IMPORTE-TOTAL-VENTAS se han inicializado a cero con anterioridad):

```
.....  

OPEN INPUT FICHERO-ENTRADA-VENTAS
MOVE "NO" TO INTER-FIN-FICHERO
PERFORM CAPTURA-REGISTRO-VENTAS
PERFORM ACUMULA-TOTALES-VENTAS
  UNTIL INTER-FIN-FICHERO EQUAL "SI"
CLOSE FICHERO-ENTRADA-VENTAS  

.....  

CAPTURA-REGISTRO-VENTAS
  READ FICHERO-ENTRADA-VENTAS
  AT END
    MOVE "SI" TO INTER-FIN-FICHERO  

ACUMULA-TOTALES-VENTAS.
  IF ID-REGION NOT NUMERIC OR ID-DEPARTAMENTO NOT NUMERIC OR
    IMPORTE-VENTAS NOT NUMERIC
      PERFORM RUTINA-NUMERO-INVALIDO
    ELSE IF ID-REGION LESS THAN 1 OR GREATER THAN LIMITE-REGION
      PERFORM RUTINA-REGION-FUERA-RANGO
    ELSE IF ID-DEPARTAMENTO LESS THAN 1 OR GREATER THAN
      LIMITE-DEPARTAMENTO
      PERFORM RUTINA-DPTO-FUERA-RANGO
    ELSE
      ADD IMPORTE-VENTAS TO
        IMPORTE-TOTAL-VENTAS (ID-REGION ID-DEPARTAMENTO)
  PERFORM CAPTURA-REGISTRO-VENTAS
```

Observe las comprobaciones de los elementos de entrada que se utilizan como subíndices: en la mayor parte de las versiones de COBOL no existen *mensajes automáticos de aviso* cuando un subíndice cae fuera de la tabla.

La conversión real de datos de entrada puede llevarse a cabo con una instrucción IF lineal si el número de valores es relativamente pequeño o mediante el examen de una tabla. En el Ejemplo 10.19 se ilustran ambas técnicas.

**EJEMPLO 10.19** Vuelva a resolver el Ejemplo 10.18 si la tabla tiene que ser  $3 \times 15$  (en lugar de  $7 \times 12$ ) y si los campos alfanuméricos vienen dados por

```
01 REGISTRO-VENTAS.
  05 ID-REGION      PIC X(3).
  05 ID-DEPARTAMENTO  PIC X(5).
  05 IMPORTE-VENTAS  PIC S9(4)V99.
```

Se puede utilizar una tabla unidimensional para convertir ID-DEPARTAMENTO al subíndice NUMERO-DEPARTAMENTO:

```
01 TABLA-ID-DEPARTAMENTOS
  05 ID-DEPARTAMENTO-VALIDO  OCCURS 15 TIMES
                                PIC X(5).
```

El primer elemento de la TABLA-ID-DEPARTAMENTOS se convertirá al valor 1 de NUMERO-DEPARTAMENTO, el segundo elemento al valor 2, y así sucesivamente. De este modo se puede obtener NUMERO-DEPARTAMENTO examinando la tabla ID-DEPARTAMENTO. Este examen, junto con la estructura IF lineal que convierte ID-REGION en el subíndice NUMERO-REGION, está contenido en la siguiente versión revisada del problema:

```

ACUMULA-TOTAL-VENTAS.
  MOVE "NO" TO INTER-ERROR
  IF ID-REGION EQUAL "XYZ"
    MOVE 1 TO NUMERO-REGION
  ELSE IF ID-REGION EQUAL "ABC"
    MOVE 2 TO NUMERO-REGION
  ELSE IF ID-REGION EQUAL "GHF"
    MOVE 3 TO NUMERO-REGION
  ELSE
    MOVE "SI" TO INTER-ERROR

  MOVE 1 TO NUMERO-DEPARTAMENTO
  PERFORM EXAMEN-DEPARTAMENTO
    UNTIL INTER-ERROR EQUAL "SI" OR
      ID-DEPARTAMENTO EQUAL
        ID-DEPARTAMENTO-VALIDO (NUMERO-DEPARTAMENTO)
  IF IMPORTE-VENTAS NOT NUMERIC
    MOVE "SI" TO INTER-ERROR

  IF INTER-ERROR EQUAL "SI"
    PERFORM RUTINA-ENTRADA-INVALIDA
  ELSE
    ADD IMPORTE-VENTAS TO
      IMPORTE-TOTAL-VENTAS (NUM-REGION NUM-DEPARTAMENTO)

  PERFORM CAPTURA-REGISTRO-VENTAS

  .....
EXAMEN-DEPARTAMENTO.
  IF NUMERO-DEPARTAMENTO EQUAL LIMITE-DEPARTAMENTO
    MOVE "SI" TO INTER-ERROR
  ELSE
    ADD 1 TO NUMERO-DEPARTAMENTO

```

En un programa más realista, los valores de ID-REGION no se escribirán como constantes, sino que serán elementos del almacenamiento auxiliar con cláusulas VALUE que las inicialicen adecuadamente:

```
05 ID-REGION-UNO  PIC X(3)  VALUE "XYZ".
```

En caso de que no se reconozca ID-REGION, se activa un interruptor de error. De forma similar, si ID-DEPARTAMENTO no se encontrara en la tabla de ID válidos, se activaría el mismo error. El interruptor de error también se activa si IMPORTE-VENTAS no es numérico.

#### Carga con la cláusula VALUE

Al igual que sucedía con las tablas unidimensionales (véase la Sección 10.3), se puede inicializar una tabla bidimensional redefiniendo un área cuya estructura coincida con la de la tabla. De nuevo no se recomienda esta técnica si es probable que tenga que cambiarse la tabla durante la vida del programa.

**EJEMPLO 10.20** Una realización de la tabla del Ejemplo 10.12:

```
01 AREA-FLETES.
  05 VALORES-FLETES.
```

```

10 FILLER.
  15 FILLER  PIC S9(3)    COMP-3   VALUE +50.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +1.50.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +2.75.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +2.25.

10 FILLER.
  15 FILLER  PIC S9(3)    COMP-3   VALUE +100.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +3.00.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +4.23.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +4.16.

10 FILLER.
  15 FILLER  PIC S9(3)    COMP-3   VALUE +999.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +3.50.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +5.45.
  15 FILLER  PIC S9(3)V99  COMP-3   VALUE +5.20.

05 TABLA-FLETES REDEFINES VALORES-FLETES.
10 RANGO-PESO      OCCURS 3 TIMES.
  15 LIMITE-PESO    PIC S9(3)    COMP-3.
  15 DESTINO        OCCURS 3 TIMES.
    20 FLETE         PIC S9(3)V99  COMP-3.

```

#### Carga con la instrucción READ

Este método y sus ventajas son análogas a los correspondientes en las tablas unidimensionales (véase la Sección 10.3).

**EJEMPLO 10.21** La tabla del Ejemplo 10.12 va a cargarse a partir de un fichero de entrada.

```

FD FICHERO-TABLA-VALORES...
01 REGISTRO-TABLA-VALORES.
  05 VALOR-PESO      PIC S9(3)    COMP-3.
  05 VALOR-FLETE    PIC S9(3)V99  COMP-3.
                           OCCURS 3 TIMES.

```

Cada REGISTRO-TABLA-VALORES va a colocarse en RANGO-PESO, de modo que VALOR-PESO se sitúe sobre LIMITE-PESO y los tres elementos de VALOR-FLETE se coloquen sobre los tres elementos de FLETE.

Con la siguiente porción de programa se realiza esta tarea.

```

.....
OPEN INPUT FICHERO-TABLA-VALORES
MOVE "NO" TO INTER-FIN-FICHERO
MOVE 1 TO NUMERO-ELEM-PESO
PERFORM CAPTURA-REGISTRO-TABLA
PERFORM CARGA-TABLA
  UNTIL INTER-FIN-FICHERO EQUAL "SI" OR NUMERO-ELEM-PESO
    GREATER THAN TALLA-TABLA-PESO
IF INTER-FIN-FICHERO EQUAL "NO"
  DISPLAY "TABLA DE FLETES PEQUEÑA: MODIFICAR PROGRAMA"
  UPON DISPOSITIVO-MENSAJES-OPERADOR

SUBTRACT 1 FROM NUMERO-ELEM-PESO
CLOSE FICHERO-TABLA-VALORES
.....

CAPTURA-REGISTRO-TABLA
READ FICHERO-TABLA-VALORES
AT END
  MOVE "SI" TO INTER-FIN-FICHERO

```

```

CARGA-TABLA.
MOVE VALOR-PESO TO LIMITE-PESO (NUMERO-ELEM-PESO)
PERFORM MUEVE-FLETES
    VARYING NUMERO-FLETES FROM 1 BY 1
        UNTIL NUMERO-FLETES GREATER THAN
            TALLA-TABLA-FLETE
    PERFORM CAPTURA-REGISTRO-TABLA
    ADD 1 TO NUMERO-ELEM-PESO

MUEVE-FLETES.
MOVE VALOR-FLETE (NUMERO-FLETES) TO
    FLETE (NUMERO-ELEM-PESO NUMERO-FLETES)

```

Los subíndices y sus límites se definen en el almacenamiento temporal:

01 TABLA-DIMENSIONES.			
05 NUMERO-ELEM-PESO	PIC S9(5)	COMP SYNC.	
05 NUMERO-FLETES	PIC S9(5)	COMP SYNC.	
05 TALLA-TABLA-PESO	PIC S9(5)	COMP SYNC	VALUE +3.
05 TALLA-TABLA-FLETE	PIC S9(5)	COMP SYNC	VALUE +3.

Observe que no se han regateado esfuerzos para hacer el programa tan flexible y fácil de modificar como ha sido posible. De este modo se calcula el número de elementos de peso (como número de registros lógicos del fichero de entrada) para que el programa adapte entre 1 y 3 elementos en RANGO-PESO. Además TALLA-TABLA-PESO y TALLA-TABLA-FLETE son variables en el programa, de forma que en caso de necesidad de ampliación sólo habría que modificar las correspondientes cláusulas OCCURS y VALUE. (Véase el Problema 10.78 para una solución más eficiente.)

## 10.6 TABLAS DE LONGITUD VARIABLE: CLAUSULA OCCURS...DEPENDING...

Hasta el momento se han visto y definido tablas con un número *fijo* de elementos; por lo general, estas tablas contenían espacio suficiente para albergar el *máximo* número de elementos que pudiera darse. Por ello era necesario disponer de un *contador* que contenía el número de elementos activos de la tabla. No hay forma en COBOL de ahorrar el espacio de memoria perdido cuando una tabla no ocupa toda la dimensión de que dispone. Sin embargo, al dar salida a tablas hacia disco o cinta, COBOL dispone de un mecanismo por el cual pasan a formar parte del registro lógico sólo los elementos activos; así se ahorra espacio en el disco o cinta.

**EJEMPLO 10.22** Considere el fichero

```

FD FICHERO-MAESTRO-PRODUCTOS...
01 REGISTRO-MAESTRO-PRODUCTOS.
    05 ID-PRODUCTO          PIC X(5).
    05 DESCRIPCION-PRODUCTO PIC X(20).
    05 NUMERO-PROVEEDORES   PIC S9(2)  COMP.
    05 INFORMACION-PROVEEDORES
        OCCURS 1 TO 50 TIMES
        DEPENDING ON NUMERO-PROVEEDORES.
        10 ID-PROVEEDORES     PIC X(4).
        10 NOMBRE-PROVEEDOR   PIC X(20).
        10 PRECIO-PROVEEDOR   PIC S9(3)V99  COMP-3.
        10 TERMINOS-PROVEEDOR PIC X(2).

```

La cláusula OCCURS...DEPENDING... indica al compilador que aunque el tamaño máximo de la tabla es de 50 elementos, el tamaño real, es decir, el ocupado por los elementos activos de la tabla (que será el que se refleje en disco o cinta), está determinado por el contenido de NUMERO-PROVEEDORES, que a su vez está comprendido entre 1 y 50.

Cuando una *tabla de longitud variable* (definida por una cláusula OCCURS...DEPENDING...) es parte de un registro lógico, el fichero correspondiente es de registros de longitud variable (Capítulo 5).

La sintaxis de la cláusula OCCURS... DEPENDING... para definir tablas de longitud variable es:

OCCURS entero-1 TO entero-2 TIMES  
DEPENDING ON nombre-de-dato

donde “entero-1” debe ser mayor o igual que 1 y menor que “entero-2”. El “nombre-de-datos” debe ser un elemento numérico (con cualquier formato USAGE) que, durante la ejecución, pueda tomar los valores enteros comprendidos entre entero-1 y entero-2, ambos inclusive. Con el valor en un instante concreto de “nombre-de-dato” queda determinado el número de elementos activos de la tabla (y por tanto el espacio necesario para su almacenamiento en disco o cinta). Observe que una tabla de longitud variable debe tener siempre por lo menos un elemento (véase el Apéndice C sobre el COBOL de 1980).

Si un registro contiene una tabla de longitud variable, es necesario que sea el *último* campo del registro. En el caso de tablas anidadas, sólo la tabla de *nivel superior* puede ser de longitud variable; en otras palabras, una cláusula OCCURS...DEPENDING... no puede estar subordinada a una cláusula OCCURS.

#### Copias de tablas de longitud variable

Si se desea copiar una tabla de longitud variable desde un área a otra, los valores de los elementos de DEPENDING en las dos cláusulas deben igualarse *antes de ejecutar la instrucción MOVE*. (Véase el Apéndice C sobre las modificaciones al COBOL de 1980.)

**EJEMPLO 10.23** A continuación se muestra el modo correcto de copiar en el almacenamiento temporal un registro lógico que contenga una tabla de longitud variable:

```

FD FICHERO-ENSAMBLAJE...
01 REGISTRO-ENSAMBLAJE.
  05 ID-PRODUCTO          PIC X(6).
  05 NUMERO-SUBENSAMBLES  PIC S9(2)    COMP.
  05 INFO-SUBENSAMBLES   OCCURS 1 TO 25 TIMES
                        DEPENDING ON
                        NUMERO-SUBENSAMBLES.

  10 ID-COMPONENTE        PIC X(6).
  10 LOCALIDAD-ALMACEN    PIC XX.
  10 EXISTENCIAS          PIC S9(5)    COMP-3.

.
.
.

WORKING-STORAGE SECTION.
01 WS-REGISTRO-ENSAMBLAJE.
  05 WS-ID-PRODUCTO      PIC X(6).
  05 WS-NUMERO-SUBENSAMBLES  PIC S9(2)    COMP.
  05 WS-INFO-SUBENSAMBLES OCCURS 1 TO 25 TIMES
                        DEPENDING ON
                        WS-NUMERO-SUBENSAMBLES.

  10 WS-ID-COMPONENTE    PIC X(6).
  10 WS-LOCALIDAD-ALMACEN PIC XX.
  10 WS-EXISTENCIAS      PIC S9(5)    COMP-3.

.
.
.

READ FICHERO-ENSAMBLAJE AT END...
MOVE NUMERO-SUBENSAMBLES      TO WS-NUMERO-SUBENSAMBLES
MOVE REGISTRO-ENSAMBLAJE      TO WS-REGISTRO-ENSAMBLAJE

```

Tenga en cuenta que “READ FICHERO-ENSAMBLAJE INTO WS-REGISTRO-ENSAMBLAJE” no habría surtido el mismo efecto, puesto que sería equivalente a

```

READ FICHERO-ENSAMBLAJE
MOVE REGISTRO-ENSAMBLAJE TO WS-REGISTRO-ENSAMBLAJE

```

que *no* iguala WS-NUMERO-SUBENSAMBLES a NUMERO-SUBENSAMBLES antes de que se ejecute la instrucción MOVE. El resultado sería probablemente el truncamiento o relleno de WS-REGISTRO-ENSAMBLAJE, dependiendo del contenido de WS-NUMERO-SUBENSAMBLES.

#### Expansión o contracción de tablas

En una tabla de longitud variable se pueden añadir nuevos elementos o borrar los ya existentes, siempre que no se violen los límites de las cláusulas TIMES y OCCURS. Si se va a añadir un elemento, el valor de la variable de DEPENDING debe incrementarse en 1; si se va a borrar un elemento, debe restarse uno al valor de la variable DEPENDING.

Los cambios en una tabla de longitud variable que afecten a su longitud no deben hacerse en el buffer del registro lógico (donde podrían interferir con otros registros lógicos del mismo bloque). Estos cambios pueden hacerse en la copia en el almacenamiento de trabajo del registro de entrada o en el buffer del registro lógico de salida.

**EJEMPLO 10.24** Para añadir información procedente de

FD FICHERO-TRANSACCIONES...	
01 REGISTRO-TRANSACCIONES.	
05 ID-PRODUCTO-ENTRADA	PIC X(6).
05 ID-COMPONENTE-ENTRADA	PIC X(6).
05 LOCALIDAD-ALMACEN-ENTRADA	PIC XX.
05 EXISTENCIAS-ENTRADA	PIC S9(5).

en el WS-REGISTRO-ENSAMBLAJE del Ejemplo 10.23, se podría hacer lo siguiente:

```

ADD 1 TO WS-NUMERO-SUBENSAMBLES
MOVE ID-COMPONENTE-ENTRADA TO
    WS-ID-COMPONENTE (WS-NUMERO-SUBENSAMBLES)
MOVE LOCALIDAD-ALMACEN-ENTRADA TO
    WS-LOCALIDAD-ALMACEN (WS-NUMERO-SUBENSAMBLES)
MOVE EXISTENCIAS-ENTRADA TO
    WS-EXISTENCIAS (WS-NUMERO-SUBENSAMBLES)

```

Observe que la variable de DEPENDING se incrementa en 1 en primer lugar, así se expande la tabla para dar cabida al nuevo elemento. A continuación, la variable de DEPENDING sirve como subíndice para localizar el nuevo elemento al final de la tabla (el lugar más simple para su colocación; véase el Ejemplo 10.37).

**EJEMPLO 10.25** Suponga que ID-COMPONENTE-ENTRADA identifica un componente que debe eliminarse de WS-REGISTRO-ENSAMBLAJE en el Ejemplo 10.23.

- (a) En un *borrado lógico* se desplazan espacios (o cualquier otro *código de borrado*) hacia los elementos de la tabla que casan con ID-COMPONENTE-ENTRADA; los espacios señalan la inoperatividad del elemento a cualquier programa que procese la tabla. Sin embargo, no se decrementa WS-NUMERO-SUBENSAMBLES, puesto que el elemento no ha desaparecido *físicamente*.

```

MOVE "SI" TO ELEMENTOS-EN-LA-TABLA
MOVE 1 TO SUBINDICE-TABLA
PERFORM LOCALIZA-COMPONENTE
    UNTIL ELEMENTOS-EN-LA-TABLA EQUAL "NO"
    OR ID-COMPONENTE-ENTRADA
        WS-ID-COMPONENTE (SUBINDICE-TABLA)
IF ELEMENTOS-EN-LA-TABLA EQUAL "NO"
    PERFORM RUTINA-ERROR
ELSE
    MOVE SPACES TO WS-ID-COMPONENTE (SUBINDICE-TABLA)

```

---

```

LOCALIZA-COMPONENTE.
IF SUBINDICE-TABLA EQUAL WS-NUMERO-SUBENSAMBLES
    MOVE "NO" ELEMENTOS-EN-LA-TABLA
ELSE
    ADD 1 TO SUBINDICE-TABLA

```

Con los espacios en WS-ID-COMPONENTE (j), cualquier programa que procese a tabla ignorará también WS-LOCALIDAD-ALMACEN (j) y WS-EXISTENCIAS (j).

- (b) En un *borrado físico* se trasladan todos los elementos que siguen al que coincide con ID-COMPONENTE-ENTRADA un lugar hacia arriba en la tabla. La variable de DEPENDING se decremente en uno para materializar el borrado físico.

```

MOVE "SI" TO ELEMENTOS-EN-LA-TABLA
MOVE 1 TO SUBINDICE-TABLA
PERFORM LOCALIZA-COMPONENTE
  UNTIL ELEMENTOS-EN-LA-TABLA EQUAL "NO"
    OR ID-COMPONENTE-ENTRADA EQUAL
      WS-ID-COMPONENTE (SUBINDICE-TABLA)
IF ELEMENTOS-EN-LA-TABLA EQUAL "NO"
  PERFORM RUTINA-ERROR
ELSE
  PERFORM TRASLADO-ARRIBA
    VARYING SUBINDICE-MOVIL FROM SUBINDICE-TABLA BY 1
      UNTIL SUBINDICE-MOVIL EQUAL WS-NUMERO-SUBENSAMBLES
    SUBTRACT 1 FROM WS-NUMERO-SUBENSAMBLES

.....
LOCALIZA-COMPONENTE.
IF SUBINDICE-TABLA EQUAL WS-NUMERO-SUBENSAMBLES
  MOVE "NO" TO ELEMENTOS-EN-LA-TABLA
ELSE
  ADD 1 TO SUBINDICE-TABLA

TRASLADO-ARRIBA.
  ADD 1 SUBINDICE-MOVIL GIVING SUBINDICE-MOVIL-MAS-UNO
  MOVE WS-INFO-SUBENSAMBLE (SUBINDICE-MOVIL-MAS-UNO) TO
    WS-INFO-SUBENSAMBLE (SUBINDICE-MOVIL)

```

Fíjese en que PERFORM-COMPONENTE UNTIL... inicializa SUBINDICE-TABLA en el número de posición del elemento a borrar (si es que existe). Todos los elementos comprendidos entre SUBINDICE-TABLA y el final de la tabla se tienen que desplazar una posición hacia arriba para eliminar el espacio que ocupaba el elemento borrado. Esta tarea se lleva a cabo en TRASLADO-ARRIBA, en donde el elemento correspondiente a SUBINDICE-MOVIL-MAS-UNO se traslada hacia el elemento correspondiente a SUBINDICE-MOVIL. SUBINDICE-MOVIL varía desde SUBINDICE-TABLA a WS-NUMERO-SUBENSAMBLES menos 1 (recuerde que cuando se emplea EQUAL en lugar de GREATER THAN en una instrucción PERFORM... VARYING... UNTIL..., el último valor del elemento de VARYING para el que se ejecuta el párrafo es una unidad inferior al valor de parada indicado). Esto es necesario porque las posiciones de los elementos a trasladarse se especifican con SUBINDICE-MOVIL-MAS-UNO. Como un subíndice no puede formar parte de una expresión, es necesario definir un elemento extra SUBINDICE-MOVIL-MAS-UNO cuyo contenido es obvio. Después de que se hayan trasladado todos los elementos necesarios un lugar hacia arriba, la variable de DEPENDING se disminuye en 1 para indicar que la tabla tiene un elemento activo menos. El borrado físico es preferible al borrado lógico en las tablas de longitud variable, puesto que con él se disminuye el espacio ocupado en el almacenamiento auxiliar por la tabla.

## 10.7 INDICES

Cuando se utilizan subíndices para identificar los elementos de una tabla, el compilador tiene que generar instrucciones en código máquina para convertir el valor del subíndice en la dirección real en memoria del elemento deseado. Los *índices* son variables de COBOL que pueden utilizarse en lugar de subíndices y hacer más eficiente el cálculo de las correspondientes direcciones en memoria (véanse los Problemas 10.74 y 10.75). Un valor del índice significa un desplazamiento en la tabla; es decir, un número que sumado a la dirección inicial de la tabla da la dirección del elemento deseado. Los índices son más eficientes que los subíndices y deben utilizarse siempre que sea posible.

Los índices deben definirse en la cláusula OCCURS donde se define la tabla; no se requiere

definición por separado (a diferencia de los subíndices). La sintaxis de la cláusula OCCURS al definir índices es la que se muestra en la Figura 10-2.

OCCURS entero-1 [TO entero-2] TIMES  
[DEPENDING ON nombre-dato-1]  
[INDEXED BY nombre-índice-1 [nombre-índice-2...]]

Fig. 10-2

“Nombre-índice-1” y “nombre-índice-2” se pueden utilizar en lugar de subíndices para acceder a elementos de la tabla.

#### EJEMPLO 10.26

```

01 CIFRAS-VENTAS-POR-ESTADOS.
  05 VENTAS-ESTADO          OCCURS 50 TIMES
                                INDEXED BY NUMERO-ESTADO.
  10 ID-ESTADO              PIC X(2).
  10 IMPORTE-VENTAS        PIC S9(5)V99 COMP-3.

```

Aquí se define la tabla del Ejemplo 10.4 con un índice. Puede utilizarse el índice NUMERO-ESTADO en lugar de SUB-VENTAS-ESTADO. Observe que un índice está asociado con una *tabla particular*; NUMERO-ESTADO no debe usarse para acceder a otra tabla a menos que tenga exactamente la misma estructura (número de elementos, niveles, USAGE, PICTURE, etc.) que VENTAS-ESTADO. Asimismo, es de remarcar que no se necesita definición adicional (tampoco se permite) de NUMERO-ESTADO.

Los índices *no se pueden* manipular del mismo modo que los subíndices. Un índice se puede inicializar o cambiar sólo con (1) una instrucción SET, (2) una instrucción PERFORM, (3) una instrucción SEARCH (véase la Sección 10.8).

La instrucción SET se puede utilizar para inicializar un índice con el valor de otro índice, un literal numérico o una variable numérica. Recíprocamente, la instrucción SET puede utilizarse para hacer que una variable numérica tome el valor de un índice. La sintaxis es:

SET {nombre-índice-1} [{nombre-índice-2}] ... TO {nombre-índice-3}  
{nombre-dato-1} [{nombre-dato-2}] ... {identificador-3}  
{literal}

#### EJEMPLO 10.27

- SET NUMERO-ESTADO TO 3

La instrucción SET convierte el valor “3” en el desplazamiento hasta el tercer elemento de VENTAS-ESTADO (véase el Ejemplo 10.26).

- SET NUMERO-ESTADO TO OTRO-INDICE

Si OTRO-INDICE apunta hacia el elemento *k*-ésimo de OTRA-TABLA, NUMERO-ESTADO apuntarán hacia el elemento *k*-ésimo de VENTAS-ESTADO.

- SET NUMERO-ESTADO TO UN-SUBINDICE

Hace que NUMERO-ESTADO apunte al elemento de la tabla cuya posición corresponde al contenido de UN-SUBINDICE. UN SUBINDICE podría definirse así:

05 UN-SUBINDICE PIC S9(5) COMP SYNC.

- SET UN-SUBINDICE TO NUMERO-ESTADO

La posición correspondiente al desplazamiento almacenado en NUMERO-ESTADO se copia en UN-SUBINDICE.

También debe utilizarse la instrucción SET para incrementar o decrementar un índice:

SET nombre-índice-1 [nombre-índice-2] ... {UP BY} {nombre-dato}  
{DOWN BY} {literal}

(Con los índices *no se pueden* utilizar MOVE..., ADD... y SUBTRACT.)

**EJEMPLO 10.28**

- SET NUMERO-ESTADO DOWN BY 1

El desplazamiento contenido en NUMERO-ESTADO se decremente de forma que apunte hacia el elemento anterior en la tabla.

- SET NUMERO-ESTADO UP BY UN-SUBINDICE

Se incrementa NUMERO-ESTADO con el valor del desplazamiento que hace que apunte hacia un elemento que corresponda a un número de posiciones hacia delante igual al contenido de UN-SUBINDICE (por ejemplo, si NUMERO-ESTADO está apuntando hacia el tercer elemento de la tabla y UN-SUBINDICE contiene el valor 5, entonces NUMERO-ESTADO acabará apuntando hacia el octavo elemento de la tabla).

Los índices se pueden utilizar en las instrucciones PERFORM e IF como cualquier otro dato entero.

**EJEMPLO 10.29** Calcule el importe total de las ventas del Ejemplo 10.26.

```

01 CIFRAS-VENTAS-POR-ESTADOS.
    05 VENTAS-ESTADO          OCCURS 50 TIMES INDEXED BY
                                NUMERO-ESTADO.
        10 ID-ESTADO           PIC X(2).
        10 IMPORTE-VENTAS     PIC S9(5)V99   COMP-3.

MOVE ZERO TO TOTAL-VENTAS
PERFORM SUMA-VENTAS-ESTADOS
    VARYING NUMERO-ESTADO FROM 1 BY 1
    UNTIL NUMERO-ESTADO GREATER THAN 50

SUMA-VENTAS-ESTADOS.
    ADD IMPORTE-VENTAS (NUMERO-ESTADO) TO TOTAL-VENTAS

```

Observe que NUMERO-ESTADO se utiliza *exactamente igual que un subíndice*. De hecho, sólo examinando la cláusula OCCURS se puede saber que no es un subíndice. No obstante, el índice NUMERO-ESTADO genera un código máquina más eficiente.

**EJEMPLO 10.30** En el Ejemplo 10.10 vuelva a escribirse el párrafo EXAMEN-ESTADO con un índice en lugar de subíndice.

```

EXAMEN-ESTADO.
    SET INDICE-TABLA-ESTADO TO 1
    MOVE "SI" TO INTER-ESTADO-VALIDO
    PERFORM VALIDACION-ENTRADA
        UNTIL INTER-ESTADO-VALIDO EQUAL "NO"
            OR CODIGO-ESTADO (INDICE-TABLA-ESTADO) EQUAL CODIGO-ESTADO-
                ENTRADA

VALIDACION-ENTRADA.
    IF INDICE-TABLA-ESTADO EQUAL 40
        MOVE "NO" TO INTER-ESTADO-VALIDO
    ELSE
        SET INDICE-TABLA-ESTADO

```

Con el índice debe utilizarse la instrucción SET, a diferencia de con los subíndices con los que se puede utilizar MOVE y ADD. Este ejemplo es más eficiente que el Ejemplo 10.10.

**EJEMPLO 10.31** Igual que el Ejemplo 10.18, pero con índices en lugar de subíndices:

```

FD FICHERO-ENTRADA-VENTAS...
01 TABLA-IMPORTES-TOTAL-VENTAS.
    05 REGION          OCCURS 7 TIMES
                        INDEXED BY INDICE-REGION.

```

```

10 DEPARTAMENTO      OCCURS 12 TIMES
                     INDEXED BY INDICE-DEPARTAMENTO.
15 IMPORTE-TOTAL-VENTAS PIC S9(5)V99 COMP.

ELSE
  SET INDICE-REGION TO ID-REGION
  SET INDICE-DEPARTAMENTO TO ID-DEPARTAMENTO
  ADD IMPORTE-VENTAS TO TOTAL-VENTAS (INDICE-REGION INDICE-DEPARTAMENTO)

PERFORM CAPTURA-REGISTRO-VENTAS

```

### Indexación relativa

A un índice se le puede sumar o restar una constante numérica (y *sólo* una constante numérica) para que apunte a un elemento a partir de aquel al que está apuntando. Esto puede ser útil al trasladar los elementos dentro de una tabla.

**EJEMPLO 10.32** Suponga que se desea mover el elemento que ocupa la posición K a la posición K-5. Con subíndices, se definirían:

```

01 SUBINDICE-TABLA.
  05 K          PIC S9(5)  COMP SYNC.
  05 K-MENOS-CINCO PIC S9(5)  COMP SYNC.

SUBTRACT 5 FROM K GIVING K-MENOS-CINCO
MOVE ELEMENTO-TABLA (K) TO ELEMENTO-TABLA (K-MENOS-CINCO)

```

Con índices sería mucho más simple:

```

SET ELEMENTO-TABLA TO K
MOVE ELEMENTO-TABLA (ELEMENTO-TABLA) TO ELEMENTO-TABLA (ELEMENTO-TABLA-5)

```

## 10.8 BUSQUEDA SECUENCIAL EN UNA TABLA Y EL VERBO SEARCH

En las secciones precedentes se ilustró el modo de programar una búsqueda en una tabla mediante un bucle PERFORM. Sin embargo, el COBOL norma ANSI posee un verbo especial, SEARCH, cuyo uso realiza este cometido de forma más sencilla y más eficiente. Si una tabla tiene  $n$  elementos, una búsqueda secuencial los examina en el orden  $k, k + 1, k + 2, \dots, n - 1, n$ . (Por lo general,  $k = 1$ .) La instrucción SEARCH de búsqueda secuencial tiene la sintaxis que se refleja en la Figura 10-3. “Identificador-1” debe ser el nombre de la tabla en la que se va a buscar (elemento con la cláusula OCCURS). La cláusula AT END especifica lo que ha de hacerse en caso de “fallo” en la búsqueda, es decir, si ninguna de las condiciones WHEN es verdadera durante el proceso de búsqueda.

```

SEARCH identificador-1 [ VARYING { identificador-2
                                { nombre-índice-1 } } ]
  [AT END instrucción-imperativa(s)-1]
    WHEN condición-1 { instrucción imperativa(s)-2
                        { NEXT SENTENCE } }
    [ WHEN condición-2 { instrucción imperativa(s)-3
                        { NEXT SENTENCE } } ] ...

```

Fig. 10-3

**EJEMPLO 10.33** Resuelva de nuevo el Ejemplo 10.10, utilizando SEARCH... en vez de PERFORM...

```

01 TABLA-CODIGOS-ESTADO.
  05 CODIGO-ESTADO      PIC X(3)
                           OCCURS 40 TIMES
                           INDEXED BY INDICE-CODIGO-VALIDO.

EXAMEN-ESTADO.
  SET INDICE-CODIGO-VALIDO TO 1
  SEARCH CODIGO-ESTADO
    AT END
      MOVE "NO" TO INTER-CODIGO-OK
    WHEN CODIGO-ESTADO-ENTRADA EQUAL CODIGO-ESTADO (INDICE-CODIGO-
      VALIDO)
      MOVE "SI" TO INTER-CODIGO-OK

```

Observe que para que tenga aplicación correcta el verbo SEARCH, la tabla debe definirse *con un índice*. El programador, mediante SET, debe prefijar el índice con el valor a partir del cual debe comenzarse la búsqueda (por lo general 1). La instrucción SEARCH de arriba funciona del modo siguiente:

- (1) Como no se especifica la opción VARYING, SEARCH hace uso del *primer* índice definido para la tabla CODIGO-ESTADO. En este caso, el primero y único índice es INDICE-CODIGO-VALIDO.
- (2) El valor del índice de SEARCH se compara con el número de elementos de la tabla (aquí 40). Si fuera superior, se ejecutarían las instrucciones de la cláusula AT END y después la ejecución proseguiría a partir del punto que marca el final de la instrucción SEARCH.
- (3) Si el índice no excede el tamaño de la tabla, se evalúan las condiciones WHEN (en el orden en que están escritas). La primera condición de WHEN es verdadera y por ello se ejecutan las instrucciones asociadas. Después la ejecución continúa con la primera instrucción que sigue al punto que marca el final de SEARCH.
- (4) Si ninguna condición de WHEN es verdadera, el índice de SEARCH se incrementa en uno y la ejecución de SEARCH continúa a partir del paso (2) de arriba.

**EJEMPLO 10.34** Por medio de SEARCH... VARYING... se pueden examinar *tablas paralelas*, tablas cuyos elementos correspondientes contienen datos de la misma entidad. Suponga que tenemos las siguientes tablas paralelas:

```

01 TABLA-CODIGO-DEDUCCION.
  05 CODIGO-DEDUCCION      PIC X
                           OCCURS 30 TIMES
                           INDEXED BY INDICE-CODIGO.

01 TABLA-INFO-DEDUCCIONES.
  05 DEDUCCIONES-NOMINA    OCCURS 30 TIMES
                           INDEXED BY INDICE-INFO.
  10 DESCRIPCION-DEDUCCION  PIC X(25).
  10 IMPORTE-DEDUCCION     PIC S99V99  COMP-3.

```

donde CODIGO-DEDUCCION (k) corresponde a DEDUCCIONES-NOMINA (k). Dado un CODIGO-NOMINA, procesamos las deducciones de la nómina con SEARCH... VARYING...:

```

SET INDICE-CODIGO TO 1
SET INDICE-INFO TO 1
SEARCH CODIGO-DEDUCCION
  VARYING INDICE-INFO
  AT END
    MOVE "SI" TO INTER-ERROR-NOMINA
    WHEN CODIGO-DEDUCCION (INDICE-CODIGO) EQUAL CODIGO-NOMINA
      MOVE DESCRIPCION-DEDUCCION (INDICE-INFO) TO AREA-SALIDA
      SUBTRACT IMPORTE-DEDUCCION (INDICE-INFO) FROM SALARIO-NETO

```

Como las dos tablas tienen estructuras diferentes, se necesitan dos índices. Cuando la opción VARYING especifica un índice que pertenece a una tabla distinta de la tabla de búsqueda: (i) el *primer* índice especificado

para la tabla de búsqueda (INDICE-CODIGO) se incrementa durante la ejecución de SEARCH, (ii) cada vez que el primer índice se incrementa, también lo hace el índice de VARYING (INDICE-INFO). Así el índice de VARYING va junto con el índice de SEARCH, de este modo apuntan siempre hacia elementos correspondientes si es que mediante SET se han inicializado en valores correspondientes antes de la ejecución de SEARCH.

Si una tabla tiene más de un índice y la cláusula VARYING... nombra un índice distinto del primero, dicho índice se utiliza en la búsqueda.

Como SEARCH... sólo manipula una cláusula OCCURS cada vez, la búsqueda en una tabla bidimensional es especial. Por lo general es suficiente incluir una instrucción SEARCH en un párrafo que se ejecutará con PERFORM; dentro del párrafo se varía el primer índice (el segundo se varía en la instrucción SEARCH).

#### EJEMPLO 10.35 Para localizar CURSO-ENTRADA PIC X(5) en la tabla

01	EXPEDIENTE-ESTUDIANTE.	
05	ID-ESTUDIANTE	PIC X(9).
05	SEMESTRES	PIC S9      COMP.
05	INFORMACION-SEMESTRE	OCCURS 1 TO 8 TIMES DEPENDING ON SEMESTRES INDEXED BY INDICE-SEMESTRE.
10	FECHA-SEMESTRE	PIC X(6).
10	NUMERO-DE-CURSOS	PIC S9      COMP.
10	INFO-CURSO	OCCURS 6 TIMES INDEXED BY INDICE-CURSO.
15	ID-CURSO	PIC X(5).
15	NUMERO-ASIGNATURAS	PIC S9V9    COMP-3.
15	GRADO	PIC S9      COMP-3.

se necesita lo siguiente:

```

MOVE "NO" INTER-ENCONTRADO
PERFORM LOCALIZA-CURSO
    VARYING INDICE-SEMESTRE FROM 1 BY 1
        UNTIL INDICE-SEMESTRE GREATER THAN SEMESTRES
            OR INTER-ENCONTRADO EQUAL "SI"
        IF INTER-ENCONTRADO EQUAL "SI"
            SET INDICE-SEMESTRE DOWN BY 1
            PERFORM PROCESO-INFO-CURSO
        ELSE
            PERFORM ERROR-CURSO-NO-ENCONTRADO

LOCALIZA-CURSO.
    SET INDICE-CURSO TO 1
    SEARCH INFO-CURSO
        WHEN INDICE-CURSO GREATER THAN
            NUMERO-DE-CURSOS (INDICE-SEMESTRE)
            NEX SENTENCE
        WHEN ID-CURSO (INDICE-SEMESTRE INDICE-CURSO)
            EQUAL CURSO-ENTRADA
            MOVE "SI" TO INTER-ENCONTRADO

```

Comentarios:

- (1) La instrucción SEARCH se ejecuta una y otra vez mientras que INDICE-SEMESTRE varía de 1 hasta SEMESTRES, su rango completo.
- (2) INDICE-CURSO, mediante SET, toma el valor 1 cada vez que se ejecuta LOCALIZA-CURSO; aquí se examina toda la tabla INFO-CURSO para cada valor de INDICE-SEMESTRE.
- (3) INFO-CURSO no tiene subíndices ni índices que afecten a la instrucción SEARCH, aunque es un elemento de la tabla INFORMACION-SEMESTRE y por tanto estará indexada.
- (4) La falta de la cláusula AT END no representa un problema en este caso. Si se alcanza el final de la tabla INFO-CURSO para un valor particular de INDICE-SEMESTRE, la ejecución continúa con la

instrucción que sigue a SEARCH, que en este caso es el fin del párrafo. Pero el llegar al final de LOCALIZA-CURSO hace que la instrucción PERFORM incremente el INDICE-SEMESTRE y se ejecute el párrafo de nuevo, así se busca INFO-CURSO en el siguiente semestre como se desea.

- (5) La primera cláusula WHEN comprueba si se ha alcanzado el *final lógico* de la tabla INFO-CURSO. Si la condición es verdadera, la cláusula NEXT SENTENCE hace que la ejecución proceda hasta el final del párrafo, siendo incrementado como en (4) INDICE-SEMESTRE.
- (6) Si la segunda condición WHEN llega a ser cierta, INTER-ENCONTRADO toma el valor "SI". Sin embargo, PERFORM... VARYING... incrementa *en primer lugar* INDICE-SEMESTRE y *después* evalúa la condición UNTIL (INTER-ENCONTRADO EQUAL "SI").

SET INDICE-SEMESTRE DOWN BY 1  
deshace el incremento.

## 10.9 BUSQUEDA BINARIA EN UNA TABLA Y EL VERBO SEARCH ALL

El algoritmo de *búsqueda binaria* es más eficiente que la búsqueda secuencial en tablas *relativamente grandes y clasificadas en orden ascendente o descendente* por algún *campo* de la tabla denominado *clave*. Cuanto mayor es la tabla, mayor es la superioridad de la búsqueda binaria sobre la búsqueda secuencial. Una búsqueda binaria comienza examinando el elemento del centro de la tabla (uno de los dos centrales si el número de elementos es par). Si este es el elemento buscado, todo ha acabado. Si el elemento buscado es mayor (tiene una clave mayor) que el elemento central, debe estar en la mitad inferior (para una tabla clasificada por orden ascendente); por el contrario, si el elemento deseado es menor que el elemento central, debe caer en la mitad superior de la tabla. En cualquier caso, se ha examinado la mitad de la tabla.

La búsqueda continua examinando el elemento central de la mitad de la tabla que queda y se aplica el mismo procedimiento reduciendo así la tabla a la mitad de la mitad. Al final todo se reduce a un elemento que o bien es el que se busca o si no dicho elemento *no está en la tabla*.

La búsqueda binaria se programa en COBOL por medio de la instrucción SEARCH ALL (Figura 10-4). "Identificador-1" debe ser el nombre de la tabla (definido con una cláusula OCCURS) en la que se va a efectuar la búsqueda; no debe estar indexada en la propia cláusula SEARCH ALL. La cláusula AT END funciona como lo haría en una búsqueda secuencial, ejecutándose sólo si la búsqueda binaria falla. Si la condición WHEN llega a ser cierta durante la búsqueda, sus instrucciones asociadas se ejecutan y la búsqueda finaliza. Observe que la condición WHEN debe tener la forma de la Figura 10-4.

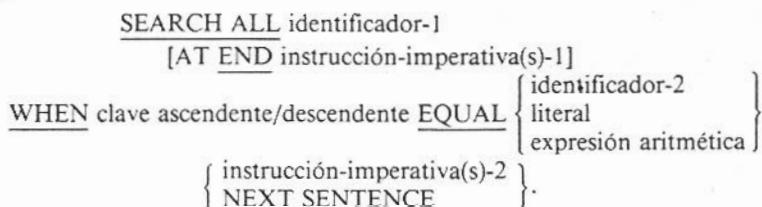


Fig. 10-4

Después de que se ejecute la cláusula AT END o la cláusula WHEN (o después de que la condición AT END tenga lugar aunque dicha cláusula no esté especificada), la ejecución continua con la instrucción inmediatamente siguiente al punto que marca el final de SEARCH ALL.

Cuando se va a procesar una tabla con SEARCH ALL, su cláusula OCCURS debe contener la subcláusula ASCENDING/DESCENDING KEY (Fig. 10-5). Esta cláusula no clasifica la tabla, sino que se limita a indicar el campo por el que está clasificada la tabla.

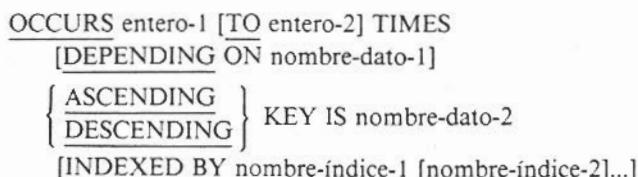


Fig. 10-5

**EJEMPLO 10.36** Supongamos que se desea recuperar la descripción de trabajo correspondiente a CODIGO-TRABAJO-ENTRADA PIC X(3) en la siguiente tabla:

```

01 TABLA-CODIGOS-TRABAJO.
  05 NUMERO-CODIGOS-TRABAJO  PIC S9(5)    COMP SYNC.
  05 SIGNIFICADO-CODIGO      OCCURS 1 TO 100 TIMES
                                DEPENDING ON NUMERO-CODIGOS-TRABAJO
                                ASCENDING KEY IS CODIGO-TRABAJO
                                INDEXED BY INDICE-CODIGO-TRABAJO.
  10 CODIGO-TRABAJO          PIC X(3).
  10 DESCRIPCION-TRABAJO     PIC X(35).

```

Asumiendo que la tabla está ya clasificada en orden ascendente por CODIGO-TRABAJO, se puede utilizar una búsqueda binaria:

```

SEARCH ALL SIGNIFICADO-CODIGO
  AT END
    MOVE "*** CODIGO INVALIDO **" TO SALIDA-DESCRIPCION
    WHEN CODIGO-TRABAJO (INDICE-CODIGO-TRABAJO) EQUAL CODIGO-TRABAJO-
      ENTRADA
      MOVE DESCRIPCION-TRABAJO (INDICE-CODIGO-TRABAJO) TO SALIDA-
        DESCRIPCION

```

Comentarios:

- (1) El nombre de la tabla, como se define en la cláusula OCCURS, debe figurar en SEARCH ALL sin índices ni subíndices.
- (2) La cláusula VARYING no es válida con SEARCH ALL: SEARCH ALL siempre varía el *primer índice* definido para la tabla en la que se efectúa la búsqueda. No es necesario inicializar el índice cuando se utiliza SEARCH ALL.
- (3) Para que SEARCH ALL funcione correctamente, la tabla debe estar clasificada del modo en que se indique en la cláusula KEY. Si la tabla no estuviera clasificada convenientemente, sin producir mensajes de error, los resultados serían erróneos. En particular, si la tabla tiene *longitud fija*, faltando la cláusula DEPENDING, algunos de los 100 elementos serán inactivos (basura) y la tabla estará totalmente sin clasificar. Véanse los Problemas 10.76 y 10.77 para ver cómo se puede programar la clasificación de una tabla.
- (4) El orden de las subcláusulas dentro de OCCURS es invariable en el COBOL del sistema IBM OS/VS (deben escribirse en el orden de la Figura 10-5).

**EJEMPLO 10.37** Los datos de CODIGO-TRABAJO-ENTRADA PIC X(3) y de DESCRIPCION-TRABAJO-ENTRADA PIC X(35) deben insertarse en un nuevo elemento de la tabla del Ejemplo 10.36. Para mantener la tabla en orden ascendente de CODIGO-TRABAJO, debe buscarse en la tabla el primer elemento que sea mayor (en términos de su clave) que el nuevo, a continuación el nuevo elemento se sitúa delante. Si no existe ningún elemento *mayor que el nuevo*, éste ocupará el final de la tabla. Si se encuentra en la tabla un elemento igual que el que se pretende insertar, se produce un error y debe tenerse en cuenta.

Deseamos comprobar CODIGO-TRABAJO GREATER THAN..., pero la cláusula WHEN de SEARCH ALL... sólo proporciona CODIGO-TRABAJO EQUAL... Por tanto, nos vemos forzados a utilizar SEARCH... en lugar de SEARCH ALL..., aunque la tabla está clasificada:

```

SET INDICE-CODIGO-TRABAJO TO 1
SEARCH SIGNIFICADO-CODIGO
  AT END
    ADD 1 TO NUMERO-CODIGOS-TRABAJO
    SET INDICE-CODIGO-TRABAJO TO NUMERO-CODIGOS-TRABAJO
    MOVE CODIGO-TRABAJO-ENTRADA TO CODIGO-TRABAJO
      (INDICE-CODIGO-TRABAJO)
    MOVE DESCRIPCION-TRABAJO-ENTRADA TO
      DESCRIPCION-TRABAJO (INDICE-CODIGO-TRABAJO)
WHEN
  CODIGO-TRABAJO (INDICE-CODIGO-TRABAJO) EQUAL CODIGO-TRABAJO-
    ENTRADA
    PERFORM ERROR-CODIGO-DUPPLICADO

```

```

WHEN
  CODIGO-TRABAJO (INDICE-CODIGO-TRABAJO) GREATER THAN CODIGO-
    TRABAJO-ENTRADA
    ADD 1 TO NUMERO-CODIGOS-TRABAJO
    PERFORM ELEMENTOS-ARRIBA
      VARYING INDICE-MOVIDA
        FROM NUMEROS-CODIGOS-ENTRADA BY -1
          UNTIL INDICE-MOVIDA EQUAL INDICE-CODIGO-TRABAJO
    MOVE CODIGO-TRABAJO-ENTRADA TO CODIGO-TRABAJO
      (INDICE-CODIGO-TRABAJO)
    MOVE DESCRIPCION-TRABAJO-ENTRADA TO
      DESCRIPCION-TRABAJO (INDICE-CODIGO-TRABAJO)

.
.
.
ELEMENTOS-ARRIBA.
MOVE SIGNIFICADO-CODIGO (INDICE-MOVIDA -1) TO
  SIGNIFICADO-CODIGO (INDICE-MOVIDA)

```

Comentarios:

- (1) La subcláusula INDEXED del Ejemplo 10.36 se transforma en
 

```

INDEXED BY INDICE-CODIGO-TRABAJO
INDICE-MOVIL.

```
- (2) Si como consecuencia de SEARCH tiene lugar la condición AT END, sabemos que el nuevo elemento es mayor que todos los existentes. Por ello, se añade al final de la tabla mediante: (i) incrementándose la variable de DEPENDING, (ii) copiando la información en el nuevo espacio creado por (i).
- (3) Si SEARCH encuentra un elemento igual al nuevo, se ejecuta una rutina de error.
- (4) Si SEARCH determina un valor de INDICE-CODIGO-TRABAJO para el cual CODIGO-TRABAJO supera CODIGO-TRABAJO-ENTRADA, se crea una nueva posición al final de la tabla incrementando NUMERO-CODIGOS-TRABAJO para ello (véase Fig. 10-6). El último elemento de la tabla, que está ahora en NUMERO-CODIGOS-TRABAJO -1, se desplaza a la nueva posición; su lugar es ocupado por el elemento de NUMERO-CODIGOS-TRABAJO -2; ...; el elemento apuntado por INDICE-CODIGO-TRABAJO se traslada a INDICE-CODIGO-TRABAJO +1; y acabada la ejecución de PERFORM, los datos de entrada se trasladan a INDICE-CODIGO-TRABAJO. Observe que es esencial efectuar los movimientos en orden inverso: si se intentara empezar moviendo el dato colocado en INDICE-CODIGO-TRABAJO hacia adelante, el resultado sería que dichos datos se copiarían en el resto de la tabla.

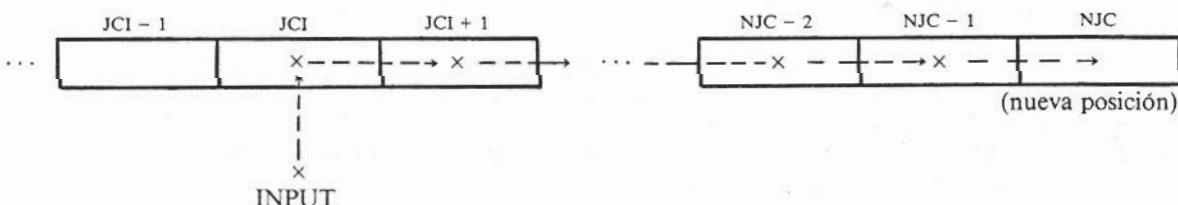


Fig. 10-6

### Preguntas de repaso

- 10.1 Dé varios ejemplos de tablas en los negocios.
- 10.2 Indique los factores a tener en cuenta para determinar el tamaño de una tabla.
- 10.3 Comente el uso de tablas en el procesamiento de información *codificada*.

- 10.4 ¿Qué se entiende por “traducción de un código mediante examen de tabla”?
- 10.5 ¿Qué es un subíndice? ¿Cómo se definen?
- 10.6 ¿En qué lugar de la división de datos se pueden definir tablas?
- 10.7 Comente el uso de la cláusula VALUE al inicializar tablas del almacenamiento de trabajo.
- 10.8 ¿En qué condiciones funcionaría adecuadamente la instrucción “MOVE ZEROS TO TABLA-COMPLETA”?
- 10.9 ¿En qué casos debe utilizarse la instrucción MOVE en un párrafo ejecutado con PERFORM para inicializar los elementos de una tabla?
- 10.10 Comente la importancia de llevar un contador del número de elementos activos en la mayor parte de las tabla. ¿Cuándo es imprescindible?
- 10.11 ¿Por qué es a veces mejor inicializar tablas desde un fichero que mediante VALUE y REDEFINES?
- 10.12 ¿Cómo se definen las tablas bidimensionales?
- 10.13 Explique las reglas del uso de subíndices e índices en tablas bidimensionales.
- 10.14 Comente el uso de PERFORM...VARYING...AFTER... con tablas bidimensionales.
- 10.15 Explique cómo pueden servir datos de entrada como subíndice. ¿En qué circunstancias podría ser de utilidad?
- 10.16 ¿Por qué es importante la validación de los datos de entrada que van a utilizarse como subíndices?
- 10.17 Indique cómo se pueden transformar campos alfanuméricos de entrada en subíndices.
- 10.18 Explique lo que se entiende por “cláusulas OCCURS anidadas”. ¿Cuál es el límite de anidamiento de cláusulas OCCURS en COBOL?
- 10.19 ¿Por qué no se ahorra memoria con el uso de tablas de longitud variable? ¿Se puede ahorrar espacio en algún otro sitio?
- 10.20 ¿Qué relación existe entre registros y tablas de longitud variable?
- 10.21 Indique y comente la cláusula DEPENDING y su posición dentro de un registro lógico. ¿Puede ir después de una tabla de longitud variable?
- 10.22 Indique las restricciones en las cláusulas OCCURS anidadas cuando se trabaja con tablas de longitud variable.
- 10.23 ¿Por qué puede definirse una tabla de longitud variable en el almacenamiento de trabajo? ¿Se ahorra así memoria?
- 10.24 Comente los aspectos a tener en cuenta al copiar tablas de longitud variable.
- 10.25 ¿Por qué no se puede utilizar READ...INTO... y WRITE...FROM... con registros que contengan tablas de longitud variable?
- 10.26 Comente los dos métodos de borrar un elemento de una tabla.
- 10.27 Explique la función del *código de borrado* en el borrado lógico de elementos de una tabla. ¿Qué implicaciones tiene el borrado lógico para los otros programas que procesan la tabla?
- 10.28 Comente el algoritmo de contracción de una tabla en el borrado físico.
- 10.29 ¿Qué es un índice? ¿Cuándo debe utilizarse? ¿Cómo se define?

- 10.30 Compare los contenidos de un índice y un subíndice.
- 10.31 ¿En qué consiste el *cálculo de la dirección*?
- 10.32 ¿Con qué tabla debe usarse un índice?
- 10.33 Indique las tres maneras de inicializar o modificar un índice.
- 10.34 Comente el uso de la instrucción SET. ¿Cuál es su sintaxis?
- 10.35 Comente cómo se pueden utilizar índices en las instrucciones PERFORM e IF.
- 10.36 ¿Qué es la *indexación relativa*? Comente su uso.
- 10.37 ¿Existe equivalencia mediante subíndices de la indexación relativa?
- 10.38 Explique el algoritmo de búsqueda secuencial.
- 10.39 Comente la instrucción SEARCH. ¿Cuál es su sintaxis? Explique (a) VARYING..., (b) AT END..., (c) WHEN..., (d) inicialización del índice de SEARCH.
- 10.40 Comente la importancia del orden en que se escriban las cláusulas de WHEN en la instrucción SEARCH...
- 10.41 ¿Qué son *tablas paralelas*? ¿Cómo se relacionan con la opción VARYING de SEARCH?
- 10.42 Indique cómo se pueden aplicar SEARCH... y SEARCH ALL... a tablas bidimensionales.
- 10.43 Explique el algoritmo de búsqueda binaria. ¿Qué es condición precedente para que la búsqueda binaria funcione?
- 10.44 Compare las eficiencias de las búsquedas secuencial y binaria. ¿Cuándo debe usarse cada una?
- 10.45 Comente la instrucción SEARCH ALL. ¿En qué se diferencia de SEARCH?
- 10.46 Explique el uso de ASCENDING/DESCENDING KEY...
- 10.47 SEARCH ALL... no debe utilizarse con tablas de longitud fija a menos que estén "llenas". ¿Por qué?
- 10.48 Describa un algoritmo para insertar un elemento en una tabla clasificada.
- 10.49 Describa un algoritmo para borrar físicamente un elemento de una tabla.

### Problemas resueltos

- 10.50 ¿Qué está equivocado en las siguientes definiciones de tablas?
- |                          |                          |
|--------------------------|--------------------------|
| (a) 01 BARAJA-DE-CARTAS  | PIC X(2)                 |
|                          | OCCURS 52 TIMES.         |
| (b) 01 BARAJA-DE-CARTAS. | OCCURS 1 TO 10 TIMES     |
| 05 MANO-POKER            | DEPENDING ON NUMERO-DE-  |
|                          | JUGADORES.               |
| 10 CARTAS                | OCCURS 5 TIMES.          |
| 15 VALOR                 | PIC XX.                  |
| 15 PALO                  | PIC X.                   |
| 05 NUMERO-DE-JUGADORES   | PIC S9(5)     COMP SYNC. |

(c) 01 MANO-POKER.  
 05 CARTA  
 10 VALOR  
 10 PALO

OCCURS 5 TIMES  
 INDEXED BY INDICE-CARTAS  
 ASCENDING KEY IS RANK.  
 PIC XX.  
 PIC X.

(d) 01 MESA-POKER.  
 05 NUMERO-DE-JUGADORES  
 05 MANO-POKER  
 10 NUMERO-ACORDADO  
 10 CARTAS  
 15 VALOR  
 15 PALO

PIC S9(4) COMP SYNC.  
 OCCURS 1 TO 10 TIMES  
 DEPENDING ON NUMERO-DE-  
 JUGADORES  
 INDEXED BY INDICE-MANO.  
 PIC S9(4) COMP SYNC.  
 OCCURS 1 TO 7 TIMES  
 DEPENDING ON NUMERO-  
 ACORDADO  
 INDEXED BY INDICE-CARTAS.  
 PIC XX.  
 PIC X.

- (a) No se puede utilizar la cláusula OCCURS al nivel 01.
- (b) Hay un campo (NUMERO-DE-JUGADORES) a continuación de una tabla de longitud variable.
- (c) A continuación de ASCENDING KEY... debe ir INDEXED BY... (en el COBOL del sistema IBM OS/VS).
- (d) Hay una cláusula OCCURS...DEPENDING... subordinada a otra cláusula OCCURS... Sería correcto utilizar:

10 CARTAS      OCCURS 7 TIMES  
 INDEXED BY INDICE-CARTAS.

una *tabla de longitud fija* puede estar subordinada a una *tabla de longitud variable* (o fija).

**10.51** Comente la tabla siguiente:

01 EQUIPO-BEISBOL		
05 JUGADOR		OCCURS 9 TIMES
		INDEXED BY INDICE-JUGADOR.
10 NOMBRE		PIC X(25).
10 POSICION		PIC X(5).
10 PROMEDIO-BATEO		PIC SV999 COMP-3.

Esta tabla contendrá siempre nueve elementos activos. Por lo general, el número de elementos activos de una tabla cambia, bien durante la ejecución de un programa que la procese o bien en algún momento durante la vida útil del programa.

**10.52** Defina una tabla que contenga el número de líneas COBOL producidas a la semana por cada uno de los programadores de una instalación. Actualmente la compañía cuenta con 55 programadores. El año próximo se tiene planificado contar con 15 más y a partir de entonces un 10% de aumento de personal, por año, dedicado a la programación. La vida útil del programa que procesa la tabla se estima en cuatro años.

01 PRODUCTIVIDAD-PROGRAMADOR.		
05 NUMERO-PROGRAMADORES		PIC S9(5) COMP SYNC.
05 DATOS PROGRAMADOR		OCCURS 1 TO 94 TIMES
		DEPENDING ON
		NUMERO-PROGRAMADORES
		INDEXED BY INDICE-PROGRAMADOR.
10 ID-PROGRAMADOR		PIC X(2)
10 LINEAS-COBOL		PIC S9(3) COMP-3.

El primer año que se use el programa, la compañía tendrá  $55+15=70$  programadores; el segundo año,  $70 \times 1.10 = 77$ ; el tercer año,  $77 \times 1.10 = 85$ ; el cuarto año,  $85 \times 1.10 = 94$ . Por ello, se utiliza una tabla de longitud variable con un máximo de 94 elementos.

- 10.53** Utilizando la tabla del Problema 10.51, escriba un programa que visualice con DISPLAY la posición con el índice medio de bateo más bajo.

Para disponer de un segundo índice se puede modificar la subcláusula INDEXED para que quede: INDEXED BY INDICE-JUGADOR INDICE-MAS-BAJO. Entonces:

```

SET INDICE-MAS-BAJO TO 1
PERFORM ENCUENTRA-PROMEDIO-MENOR
    VARYING INDICE-JUGADOR FROM 2 BY 1
        UNTIL INDICE-JUGADOR GREATER THAN 9
DISPLAY POSICION (INDICE-MAS-BAJO)
.
.
.
ENCUENTRA-PROMEDIO-MENOR.
IF PROMEDIO-BATEO (INDICE-JUGADOR) LESS THAN
    PROMEDIO-BATEO (INDICE-MAS-BAJO)
        SET INDICE-MAS-BAJO TO INDICE-JUGADOR

```

- 10.54** Si en el Problema 10.53 hubiera un empate en el promedio menor, ¿se visualizaría el primero o el último que se encontrase?

Como se usa el operador relacional "LESS THAN", se visualizaría con DISPLAY el primero que se encontrase.

- 10.55** Modifique el Problema 10.53 para que se visualice el último encontrado en caso de empate por el promedio menor.

Usando "NOT GREATER THAN" en vez de "LESS THAN" en el párrafo ENCUENTRA-PROMEDIO-MENOR.

- 10.56** Decodifique CODIGO-LOCALIDAD-ENTRADA PIC X(3) dando SALIDA-LOCALIDAD PIC X(30) y utilizando la información de las tablas siguientes:

01 TABLA-CODIGOS-LOCALIDAD.	
05 CODIGO-LOCALIDAD	PIC X(3)
	OCCURS 80 TIMES
	ASCENDING KEY
	CODIGO-LOCALIDAD
	INDEXED BY INDICE-CODIGO.
01 TABLA-DESCRIPCIONES-LOCALIDAD.	
05 DESCRIPCION-LOCALIDAD	PIC X(30)
	OCCURS 80 TIMES
	INDEXED BY
	INDICE-DESCRIPCION.

Puesto que se corresponden los elementos de ambas tablas, se puede utilizar (asumiendo que TABLA-CODIGOS-LOCALIDAD está realmente clasificada por CODIGO-LOCALIDAD) lo siguiente:

```

SEARCH ALL CODIGO-LOCALIDAD
    AT END
        MOVE "*** LOCALIDAD DESCONOCIDA **" TO SALIDA-LOCALIDAD
    WHEN
        CODIGO-LOCALIDAD (INDICE-CODIGO) EQUAL CODIGO-LOCALIDAD-
            ENTRADA
            SET INDICE-DESCRIPCION TO INDICE-CODIGO
            MOVE DESCRIPCION-LOCALIDAD (INDICE-DESCRIPCION)
                TO SALIDA-LOCALIDAD

```

- 10.57 Vuelva a resolver el Problema 10.56 si la tabla de códigos de localidad no está clasificada.

Se tiene que realizar una búsqueda secuencial:

```

SET INDICE-CODIGO
INDICE-DESCRIPCION TO 1
SEARCH CODIGO-LOCALIDAD
VARYING INDICE-DESCRIPCION
AT END
MOVE "*** LOCALIDAD-DESCONOCIDA **" TO SALIDA-LOCALIDAD
WHEN
CODIGO-LOCALIDAD (INDICE-CODIGO) EQUAL CODIGO-LOCALIDAD-
ENTRADA
MOVE DESCRIPCION-LOCALIDAD (INDICE-DESCRIPCION)
TO SALIDA-LOCALIDAD

```

Como el elemento de VARYING pertenece a otra tabla, se incrementa junto con el índice de SEARCH (primer índice de la tabla de SEARCH); de este modo INDICE-CODIGO e INDICE-DESCRIPCION siempre apuntan hacia elementos correspondientes.

- 10.58 ¿Qué sucede en COBOL si un índice o subíndice cae por debajo de 1 o supera la dimensión de la tabla?

Se pueden violar las áreas de memoria “anteriores” o “posteriores” a la tabla. Si existe la posibilidad de que un subíndice o índice tome valores inválidos, el programa debe efectuar la comprobación antes de acceder a la tabla.

- 10.59 Critique lo siguiente:

01 ASIGNACION-PROYECTOS.		
05 VALORES-PROYECTO.		
10 FILLER PIC X(11)	VALUE "MARYX250100".	
10 FILLER PIC X(11)	VALUE "MIKE2A70553".	
10 FILLER PIC X(11)	VALUE "JOHNR20050".	
05 INFO-PROGRAMADOR	REDEFINES VALORES-PROYECTO	
	OCCURS 3 TIMES	
	INDEXED BY INDICE-	
	PROGRAMADOR.	
10 NOMBRE	PIC X(4).	
10 CODIGO-PROYECTO	PIC X(3).	
10 HORAS-TRABAJADAS	PIC S9(3)V9 COMP-3.	

- (1) Las cláusulas VALUE no casan con la descripción de un elemento de la tabla: HORAS-TRABAJADAS es un campo COMP-3, pero su cláusula VALUE contiene un valor en formato DISPLAY.
- (2) Como la información de esta tabla cambiará con frecuencia, el uso de VALUE y REDEFINES para inicializar la tabla no es lo mejor que puede hacerse. Sería mucho mejor guardar la información de la tabla en un fichero en disco, donde se puede modificar con facilidad y cargar la tabla cuando sea necesario.
- (3) Puesto que el número de programadores no va a ser constante, debería haber un contador asociado a la tabla que indicase el número de elementos activos. Dicho contador se debería definir al nivel 05 debajo de ASIGNACION-PROYECTOS.

- 10.60 Corrija VALORES-PROYECTO del Problema 10.59 para subsanar la objeción (1).

05 VALORES-PROYECTO.		
10 FILLER PIC X(7)	VALUE "MARYX25".	
10 FILLER PIC S9(3)V9	COMP-3 VALUE +10.0.	
10 FILLER PIC X(7)	VALUE "MIKE2A7".	
10 FILLER PIC S9(3)V9	COMP-3 VALUE +55.3.	

```
10 FILLER PIC X(7)           VALUE "JOHNRS2".
10 FILLER PIC S9(3)V9      COMP-3  VALUE +5.0.
```

- 10.61** (a) ¿Funcionaría lo siguiente con la tabla del Problema 10.56?

```
MOVE SPACES TO TABLA-CODIGOS-LOCALIDAD
TABLA-DESCRIPCIONES-LOCALIDAD
```

- (b) ¿Funcionaría lo siguiente con la tabla del Problema 10.52?

```
MOVE SPACES TO DATOS-PROGRAMADOR
```

- (a) Sí, puesto que todos los elementos son PIC X(m) y DISPLAY.
- (b) No. En primer lugar DATOS-PROGRAMADOR requiere un subíndice o índice cuando se utiliza en una instrucción MOVE. En segundo lugar los campos tabulados son PIC X(2) Y PIC S9(3) COMP-3, por tanto como MOVE no se pueden inicializar los dos a la vez.

- 10.62** Escriba una porción de programa que inicialice ID-PROGRAMADOR y LINEAS-COBOL (Problema 10.52) con espacios y ceros, respectivamente.

```
PERFORM BORRA-DATOS-PROGRAMADOR
VARYING INDICE-PROGRAMADOR FROM 1 BY 1
UNTIL INDICE-PROGRAMADOR GREATER THAN
NUMERO-PROGRAMADORES
```

```
.....
```

BORRA-DATOS-PROGRAMADOR
MOVE SPACES TO ID-PROGRAMADOR (INDICE-PROGRAMADOR)
MOVE ZEROS TO LINEAS-COBOL (INDICE-PROGRAMADOR)

- 10.63** Defina una tabla que pueda contener hasta 100 elementos ID-ARTICULO PIC X(5) y CANTIDAD PIC S9(4) COMP para cada uno de los hasta 20 elementos ALMACEN PIC X(3).

01 ALMACEN.		
05 NUMERO-LOCALIDADES	PIC S9(5) COMP SYNC.	
05 DATOS-LOCALIDAD	OCCURS 1 TO 20 TIMES	
	DEPENDING ON NUMERO-LOCALIDADES	
	INDEXED BY INDICE-LOCALIDAD.	
10 LOCALIDAD-ALMACEN	PIC X(3).	
10 NUMERO-ARTICULOS	PIC S9(5) COMP.	
10 DATOS-ARTICULOS	OCCURS 100 TIMES	
	INDEXED BY INDICE-ARTICULO.	
15 ID-ARTICULO	PIC X(5).	
15 CANTIDAD	PIC S9(4) COMP.	

Alguien podría desear definir DATOS-ARTICULOS así: "OCCURS 1 TO 100 TIMES DEPENDING ON NUMERO-ARTICULOS". Sin embargo, una tabla de longitud variable no puede formar parte de ninguna otra tabla.

- 10.64** ¿Qué nombres de datos del Problema 10.63 podrían utilizarse en (a) una instrucción SEARCH ALL, (b) una instrucción SEARCH?

(a) Ninguno, puesto que no hay cláusula ASCENDING DESCENDING KEY. (b) DATOS-LOCALIDAD y DATOS-ARTICULOS (observe que la tabla debe estar indexada para que puedan utilizarse las instrucciones SEARCH o SEARCH ALL).

- 10.65** Escriba cada nombre de datos del Problema 10.63 como debería aparecer en una instrucción MOVE.

ALMACEN; NUMERO-LOCALIDADES; DATOS-LOCALIDAD (INDICE-LOCALIDAD); LOCALIDAD-ALMACEN (INDICE-LOCALIDAD); NUMERO-ARTICULOS (INDICE-LOCALI-

DAD); DATOS-ARTICULOS (INDICE-LOCALIDAD INDICE-ARTICULO); ID-ARTICULO (INDICE-LOCALIDAD INDICE-ARTICULO); CANTIDAD (INDICE-LOCALIDAD INDICE-ARTICULO).

- 10.66** Muestre cómo incrementar un 10% a la CANTIDAD de cada elemento de la tabla del Problema 10.63.

```

PERFORM SUMA-DIEZ-POR-CIENTO
VARYING INDICE-LOCALIDAD FROM 1 BY 1
UNTIL INDICE-LOCALIDAD GREATER THAN NUMERO-
LOCALIDADES
AFTER INDICE-ARTICULO FROM 1 BY 1
UNTIL INDICE-ARTICULO GREATER THAN
NUMERO-ARTICULOS (INDICE-LOCALIDAD)

.....
SUMA-DIEZ-POR-CIENTO
COMPUTE CANTIDAD (INDICE-LOCALIDAD INDICE-ARTICULO) =
CANTIDAD (INDICE-LOCALIDAD INDICE-ARTICULO) * 1.10

```

Observe que NUMERO-ARTICULOS debe estar indexado al ser un elemento de la tabla.

- 10.67** Escriba una porción de programa que visualice con DISPLAY *todas* las localidades de los almacenes que contengan el artículo identificado por ID-ARTICULO-ENTRADA PIC X(5). Utilice la tabla del Programa 10.63.

```

PERFORM ENCUENTRA-ARTICULO
VARYING INDICE-LOCALIDAD FROM 1 BY 1
UNTIL INDICE-LOCALIDAD GREATER THAN NUMERO-LOCALIDADES

.....
ENCUENTRA-ARTICULO.
SET INDICE-ARTICULO TO 1
SEARCH DATOS-ARTICULOS
AT END
NEXT SENTENCE
WHEN
ID-ARTICULO-ENTRADA EQUAL
ID-ARTICULO (INDICE-LOCALIDAD INDICE-ARTICULO)
DISPLAY LOCALIDAD-ALMACEN (INDICE-LOCALIDAD)

```

- 10.68** Se dan los siguientes registros lógicos:

FD FICHERO-ENVIO...	
01 REGISTRO-ENVIO.	
05 ID-CLIENTE	PIC X(6).
05 ID-PRODUCTO	PIC X(4).
05 CLASE-ENVIO	PIC S9(2).
05 CLASE-PESO	PIC S9(2).
05 DIRECCION	PIC X(50).

Suponga que se ha definido y cargado una tabla de costes de envío en el almacenamiento de trabajo:

01 TABLA-COSTE-ENVIO.	
05 NUMERO-CLASES-ENVIO	PIC S9(5) COMP SYNC.
05 CLASES-ENVIO	OCCURS 30 TIMES INDEXED BY INDICE-ENVIO.
10 NUMERO-CLASES-PESO	PIC S9(5) COMP.
10 CLASES-PESO	OCCURS 15 TIMES INDEXED BY INDICE-PESO.
15 COSTE-ENVIO	PIC S9(3)V99 COMP-3.

Asumiendo que CLASE-ENVIO se define como un entero comprendido entre 1 y 30, y que CLASE-PESO es otro entero entre 1 y 15, escriba una porción de programa que dé a CARGO-ENVIO-CLIENTE el valor apropiado. Si no se puede localizar el coste, se debe dar a CARGO-ENVIO-CLIENTE el valor cero.

```

IF CLASE-ENVIO LESS THAN 1 OR GREATER THAN
    NUMERO-CLASES-ENVIO
        MOVE ZERO TO CARGO-ENVIO-CLIENTE
    ELSE IF CLASE-PESO LESS THAN 1 OR GREATER THAN
        NUMERO-CLASES-PESO (CLASE-ENVIO)
            MOVE ZERO TO CARGO-ENVIO-CLIENTE
    ELSE
        MOVE COSTE-ENVIO (CLASE-ENVIO CLASE-PESO)
            TO CARGO-ENVIO-CLIENTE

```

- 10.69 Revise el Problema 10.68 para que utilice índices en lugar de subíndices.

```

IF CLASE-ENVIO LESS THAN 1 OR GREATER THAN
    NUMERO-CLASES-ENVIO
        MOVE ZERO TO CARGO-ENVIO-CLIENTE
    ELSE
        SET INDICE-ENVIO TO CLASE-ENVIO
        IF CLASE-PESO LESS THAN 1 OR GREATER THAN
            NUMERO-CLASES-PESO (INDICE-ENVIO)
                MOVE ZERO TO CARGO-ENVIO-CLIENTE
        ELSE
            SET INDICE-PESO TO CLASE-PESO
            MOVE COSTE-ENVIO (INDICE-ENVIO INDICE-PESO)
                TO CARGO-ENVIO-CLIENTE

```

- 10.70 ¿Qué hay equivocado y cómo se puede corregir lo siguiente?

```

SET INDICE-UNA-TABLA TO 10
SEARCH UNA-TABLA
    AT END NEXT SENTENCE
    WHEN ELEMENTO-UNA-TABLA (INDICE-UNA-TABLA) EQUAL
        SPACES
    SEARCH ALL OTRA-TABLA
        WHEN OTRA-CLAVE (OTRO-INDICE) EQUAL ZERO
            PERFORM LLUPI

```

Se puede comenzar una búsqueda secuencial a partir de cualquier elemento de una tabla; por tanto, suponiendo que UNA-TABLA tiene por lo menos 10 elementos, la instrucción SET es correcta. Sin embargo, la cláusula WHEN debe contener sólo instrucciones *imperativas* y SEARCH ALL se considera instrucción condicional. El programa apropiado es:

```

SET INDICE-UNA-TABLA TO 10
SEARCH UNA-TABLA
    AT END NEXT SENTENCE
    WHEN ELEMENTO-UNA-TABLA (INDICE-UNA-TABLA) EQUAL SPACES
        PERFORM BUSCA-OTRO
    .....
    BUSCA-OTRO.
    SEARCH ALL OTRA-TABLA
        WHEN OTRA-CLAVE (OTRO-INDICE) EQUAL ZERO
            PERFORM LLUPI

```

- 10.71 El Problema 10.68 se cambia de modo que CLASE-ENVIO es PIC X(4), CLASE-PESO es PIC X(3) y TABLA-COSTE-ENVIO contiene un elemento de nivel 10 ID-CLASE-ENVIO PIC X(4) y un elemento de nivel 15 ID-CLASE-PESO PIC X(3). Vuelva a escribir el programa para convertir la entrada alfanumérica a los índices apropiados.

```

SET INDICE-ENVIO TO 1
SEARCH CLASES-ENVIO
AT END
MOVE ZERO TO CARGO-ENVIO-CLIENTE
WHEN
ID-CLASE-ENVIO (INDICE-ENVIO) EQUAL
CLASE-ENVIO
PERFORM LOCALIZA-CLASE-PESO

.....
LOCALIZA-CLASE-PESO.
SET INDICE-PESO TO 1
SEARCH CLASES-PESO
AT END
MOVE ZERO TO CARGO-ENVIO-CLIENTE
WHEN
ID-CLASE-PESO (INDICE-ENVIO INDICE-PESO)
EQUAL CLASE-PESO
MOVE COSTE-ENVIO (INDICE-ENVIO INDICE-PESO)
TO CARGO-ENVIO-CLIENTE

```

- 10.72 Dada la tabla (Problema 10.68)

01 TABLA-ENVIO.		
05 NUMERO-CLASES	PIC S9(5)	COMP SYNC.
05 ELEMENTO-CLASE	OCCURS 1 TO 80 TIMES	
	DEPENDING ON NUMERO-CLASES	
	ASCENDING KEY ID-CLASE	
	INDEXED BY INDICE-CLASE.	
10 ID-CLASE	PIC X(4).	
10 NUMERO-PESOS	PIC S9(5)	COMP.
10 ELEMENTO-PESO	OCCURS 15 TIMES	
	DESCENDING KEY ID-PESO	
	INDEXED BY INDICE-PESO.	
15 ID-PESO	PIC X(3).	
15 COSTE	PIC S9(3)V99	COMP-3.

revise el Problema 10.71 para que se utilice búsqueda binaria cuando se pueda y detenga la búsqueda secuencial cuando el índice de búsqueda exceda el número de elementos de la tabla.

```

SEARCH ALL ELEMENTO-CLASE
AT END
MOVE ZERO TO CARGO-ENVIO-CLIENTE
WHEN
ID-CLASE (INDICE-CLASE) EQUAL CLASE-ENVIO
PERFORM LOCALIZA-CLASE-ENVIO

.....

```

```

LOCALIZA-CLASE-ENVIO
SET INDICE-PESO TO 1
SEARCH ELEMENTO-PESO
AT END
MOVE ZERO TO CARGO-ENVIO-CLIENTE

```

```

WHEN
    INDICE-PESO GREATER THAN NUMERO-PESOS (INDICE-CLASE)
        MOVE ZERO TO CARGO-ENVIO-CLIENTE
WHEN
    ID-PESO (INDICE-CLASE INDICE-PESO) EQUAL CLASE-PESO
        MOVE COSTE (INDICE-CLASE INDICE-PESO)
            TO CARGO-ENVIO-CLIENTE

```

Observe que esta solución es más eficiente que la del Problema 10.71.

- 10.73** ¿Por qué no se utiliza SEARCH ALL... para la búsqueda de ELEMENTO-PESO en el Problema 10.72?

Porque es una tabla de longitud fija que no siempre estará llena; NUMERO-PESOS no es siempre 15. Por tanto, la tabla completa no está clasificada.

- 10.74** Dadas las tablas del Problema 10.56, (a) ¿cuál es el contenido de INDICE-DESCRIPCION después de "SET INDICE-DESCRIPCIÓN TO 1"? (b) ¿qué se suma a INDICE-DESCRIPCION en "SET INDICE-DESCRIPCION UP BY 1"?

- (a) Cero: Un índice representa el *desplazamiento* necesario en la tabla para alcanzar el elemento deseado. El primer elemento está cero bytes hacia delante desde el comienzo de la tabla.
- (b) Treinta: Los índices se aumentan o disminuyen en cantidades que son múltiplos de la longitud de un elemento de la tabla (tantas veces la longitud como posiciones haya que saltar).

- 10.75** Contraste los cálculos de la dirección en lenguaje máquina precisos con los subíndices, con la indexación en tablas unidimensionales.

Si K es un subíndice, es el número de posición de un elemento:

$$\text{dirección de ELEMENTO (K)} = \text{dirección-comienzo-tabla} + (K - 1) * \text{longitud-elemento}$$

Si K es un índice, es el desplazamiento desde el principio de la tabla:

$$\text{dirección de ELEMENTO (K)} = \text{dirección-comienzo-tabla} + K$$

- 10.76** ¿Qué ocurre si se produce una búsqueda binaria en una tabla que no está clasificada? Una solución, si la tabla se carga desde un fichero, consiste en realizar una *clasificación por inserción* en la que cada elemento se coloca en su posición según se les va dando entrada desde el fichero. Aplique dicha técnica a:

FD	FICHERO-ENTRADA...	
01	REGISTRO-ENTRADA.	
	05 CODIGO-ENTRADA	PIC X(3).
	05 DESCRIPCION-ENTRADA	PIC X(35).
01	TABLA-CODIGOS-TRABAJO.	
	05 NUMERO-CODIGOS	PIC S9(5) COMP SYNC.
	05 ELEMENTO-CODIGO	OCCURS 1 TO 100 TIMES DEPENDING ON NUMERO- CODIGOS ASCENDING KEY IS CODIGO- TRABAJO INDEXED BY INDICE-CODIGO INDICE MOVIL.
10	CODIGO-TRABAJO	PIC X(3).
10	DESCRIPCION-TRABAJO	PIC X(35).

```

CONST-TABLA.
  OPEN INPUT FICHERO-ENTRADA
  MOVE "NO" TO INTER-FIN-FICHERO
  PERFORM CAPTURA-SIGUIENTE-REGISTRO
  IF INTER-FIN-FICHERO EQUAL "NO"
    MOVE 1 TO NUMERO-CODIGOS
    MOVE CODIGO-ENTRADA      TO CODIGO-TRABAJO (1)
    MOVE ENTRADA-DESCRIPCION TO DESCRIPCION-TRABAJO (1)
    PERFORM CAPTURA-SIGUIENTE-REGISTRO
    PERFORM INSERTA-EN-ORDEN
      UNTIL INTER-FIN-FICHERO EQUAL "SI" OR NUMERO-CODIGOS
      EQUAL 100
  ELSE
    PERFORM ERROR-FICHERO-VACIO

  IF INTER-FIN-FICHERO NOT EQUAL "SI"
    PERFORM ERROR-TABLA-PEQUEÑA

  CLOSE FICHERO-ENTRADA
  .....
INSERTA-EN-ORDEN.
  (rutina como la del Ejemplo 10.37)

```

- 10.77** Otra forma (ineficiente) de clasificar una tabla es la *clasificación por burbujas*, en la cual los elementos adyacentes se comparan y se intercambian si no estuvieran en orden. Obviamente, después de un adecuado número de pasadas, la tabla quedará clasificada. Escriba un programa que clasifique mediante este algoritmo la tabla del Problema 10.76.

Se define un índice más para la tabla INDICE-PARADA, así como dos elementos INTERCAMBIO-NECESARIO y SUJETA-ELEMENTO, que son PIC X(3) y PIC X(38), respectivamente.

```

SET INDICE-PARADA TO NUMERO-CODIGOS
SET INDICE-PARADA DOWN BY 1
MOVE "SI" TO SUJETA-ELEMENTO
PERFORM PASADA
  UNTIL INDICE-PARADA NOT GREATER THAN ZERO
  OR SUJETA-ELEMENTO EQUAL "NO"
  .....

```

```

PASADA.
  MOVE "NO" TO SUJETA-ELEMENTO
  PERFORM COMPARA-E-INTERCAMBIA
  VARYING INDICE-CODIGO FROM 1 BY 1
  UNTIL INDICE-CODIGO GREATER THAN INDICE-PARADA
  SET INDICE-PARADA DOWN BY 1
  .....

```

```

COMPARA-E-INTERCAMBIA.
  IF CODIGO-TRABAJO (INDICE-CODIGO) GREATER THAN
    CODIGO-TRABAJO (INDICE-CODIGO + 1)
    MOVE "SI" TO SUJETA-ELEMENTO
    MOVE ELEMENTO-CODIGO (INDICE-CODIGO) TO SUJETA-ELEMENTO
    MOVE ELEMENTO-CODIGO (INDICE-CODIGO + 1) TO
      ELEMENTO-CODIGO (INDICE-CODIGO)
    MOVE SUJETA-ELEMENTO TO ELEMENTO-CODIGO (INDICE-
      CODIGO + 1)

```

- 10.78 Dé una forma más eficiente para cargar la tabla del Ejemplo 10.21.

Como REGISTRO-TABLA-VALORES tiene *exactamente* la misma estructura que RANGO-PESO, en la TABLA-FLETES se puede hacer uso de una instrucción MOVE para copiar todo el elemento. Así CARGA-TABLA se podría escribir como sigue (eliminando la necesidad de MUEVE-FLETES):

```
CARGA-TABLA.  
MOVE REGISTRO-TABLA-VALORES TO  
      RANGO-PESO (NUMERO-ELEM-PESO)  
ADD 1 TO NUMERO-ELEM-PESO  
PERFORM CAPTURA-REGISTRO-TABLA
```

# Capítulo 11

## Procesamiento secuencial de ficheros

A estas alturas ya estamos suficientemente familiarizados con (1) la creación y (2) la recuperación de registros lógicos de un fichero cuya organización es secuencial; se conoce también (3) la utilización del área FILE STATUS a la hora de detectar errores en el procesamiento de ficheros secuenciales. A continuación veamos un ejemplo de esto último:

**EJEMPLO 11.1** En los sistemas IBM, el código de estado “00” indica un acceso con éxito, “10” indica fin de fichero, “30” significa un error permanente en el dispositivo I/O y “92” indica la existencia de un error lógico de programa. De modo que si la cláusula FILE STATUS de la instrucción SELECT nombra al elemento del WORKING-STORAGE AREA-CODIGO-ESTADO PIC XX, es posible que tengamos en la división de procedimientos (PROCEDURE DIVISION):

```
OPEN INPUT UN-FICHERO-SECUENCIAL
IF AREA-COD-ESTADO NOT EQUAL "00"
  DISPLAY "NO SE PUEDE ABRIR"
  MOVE "SI" TO INTER-ABORTO

.....
READ UN-FICHERO-SECUENCIAL
IF AREA-COD-ESTADO NOT EQUAL "00"
  IF AREA-COD-ESTADO EQUAL "10"
    MOVE "SI" TO INTER-FIN-FICHERO
  ELSE IF AREA-COD-ESTADO EQUAL "30"
    DISPLAY "ERROR LECTURA FICHERO"
    MOVE "SI" TO INTER-IGNORA-REGISTRO

.....
CLOSE UN-FICHERO-SECUENCIAL
IF AREA-COD-ESTADO NOT EQUAL "00"
  DISPLAY "FALLO EN EL CIERRE"
```

### 11.1 ACTUALIZACION DE FICHEROS SECUENCIALES

Los ficheros maestros organizados de manera secuencial se conservan generalmente clasificados por un campo clave (Sección 10.9), hecho que es vital para la actualización del fichero. Los registros lógicos contenido información para ser incorporada o eliminada del fichero maestro están generalmente almacenados en un fichero secuencial aparte, denominado *fichero de transacciones*. Cada registro de un fichero de transacciones especifica la clave de un registro del fichero maestro, que se verá ampliado, modificado o suprimido; el fichero de transacciones está clasificado por esta misma clave. Una típica actualización de ficheros secuenciales se ve reflejada en el diagrama de flujo de la Figura 11-1 y supone los siguientes pasos:

1. *O bien* los documentos fuente se recopilan a lo largo de un período de tiempo hasta que todo el lote se introduce en el fichero secuencial de transacciones en el disco; *o* las transacciones son almacenadas a medida que van produciéndose por medio de terminales en linea, pero el programa que comunica con los terminales, en lugar de procesar dichos datos, simplemente los recoge agrupados en el fichero secuencial de transacciones en disco.

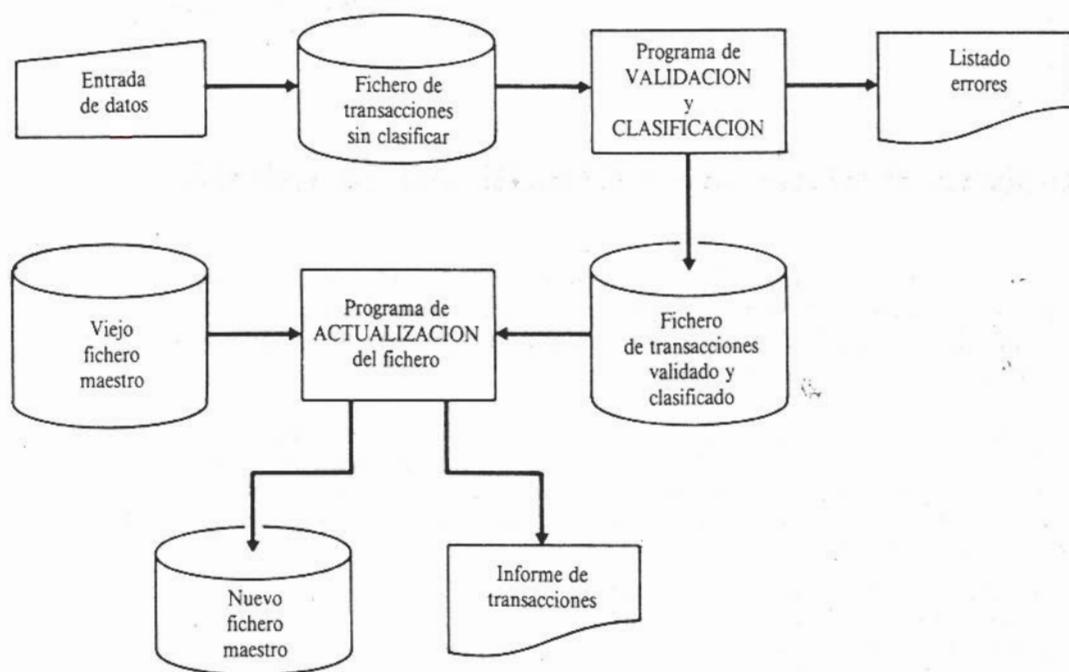


Fig. 11-1

2. En cualquier caso, tenemos un fichero de transacciones secuencial, que está validado y clasificado en el mismo orden que el fichero maestro, que habrá de ser actualizado. Las transacciones con errores deberán corregirse y reciclarse de nuevo al paso 1.
3. Las transacciones validadas y clasificadas entran ahora en un programa de actualización de ficheros secuenciales. Este programa además accede a los registros existentes en el fichero maestro, realiza los cambios precisos y crea un *nuevo fichero maestro* actualizado. En este proceso se ha conservado el *viejo fichero maestro*.
4. El antiguo fichero maestro y el fichero de transacciones se conservan por motivos de seguridad (backup). Si ocurriera algo al nuevo fichero maestro, podría volver a crearse simplemente con ejecutar el programa de actualización de nuevo. Esto se conoce como el método de backup *abuelo, padre, hijo*, ya que pueden conservarse tantas "generaciones" de transacciones de antiguos ficheros maestros como se desee.
5. El programa de actualización de ficheros secuenciales ejecutado en el paso 3, generalmente produce un informe impreso, denominado *informe de transacciones*, el cual lista los resultados de aplicar cada transacción al fichero maestro. Téngase en cuenta que aunque la transacción sea válida en sí misma podría estar en discordancia con el fichero maestro (por ejemplo, es posible que tengamos una transacción a suprimir correctamente especificada para un registro de fichero maestro que no existe). Tales errores deben ser corregidos y reciclados de nuevo al paso 1.

## 11.2 ALGORITMO "MAESTRO-TRANSACCIONES"

La Figura 11-2 define, en pseudocódigo COBOL, uno de los métodos más eficaces para casar los registros de un fichero de transacciones ordenado con los registros de un fichero maestro ordenado, lo que conocemos por *algoritmo "maestro-transacciones"*. El algoritmo maneja adecuadamente más de un registro de transacciones por registro maestro, una circunstancia común cuando las transacciones corresponden a un largo período de tiempo. Deberá tenerse en cuenta que todos los detalles del procesamiento de los campos de registro han sido dejados para módulos de menor nivel. De ahí que los módulos de alto nivel puedan ser utilizados en múltiples programas diferentes para distintas aplicaciones.

```

MAESTRO-TRANSACCIONES.
MOVE "NO" TO MAESTRO-LISTO-PARA-SALIDA
OPEN INPUT      FICHERO-TRANSACCIONES
          VIEJO-FICHERO-MAESTRO
OUTPUT      NUEVO-FICHERO-MAESTRO
          INFORME-ERRORES
PERFORM CONSIGUE-NUEVA-TRANSACCION
PERFORM CONSIGUE-SIGUIENTE-MAESTRO
PERFORM DETERMINA-CLAVE-MENOR
PERFORM ACTUALIZACION-FICHERO-MAESTRO
    UNTIL CLAVE-MENOR EQUAL HIGH-VALUES
CLOSE FICHERO-TRANSACCIONES
VIEJO-FICHERO-MAESTRO
NUEVO-FICHERO-MAESTRO
INFORME-ERRORES
STOP RUN

ACTUALIZACION-FICHERO-MAESTRO.
IF CLAVE-MENOR EQUAL CLAVE-VIEJO-MAESTRO
    MOVE REGISTRO-VIEJO-MAESTRO TO WS-REGISTRO-NUEVO-MAESTRO
    MOVE "SI" TO MAESTRO-LISTO-PARA-SALIDA
    PERFORM CONSIGUE-SIGUIENTE-MAESTRO

IF CLAVE-MENOR EQUAL CLAVE-TRANSACCION
    PERFORM PROCESA-UNA-TRANSACCION
        UNTIL CLAVE-MENOR NOT EQUAL CLAVE-TRANSACCION

IF MAESTRO-LISTO-PARA-SALIDA EQUAL "SI"
    WRITE REGISTRO-NUEVO-MAESTRO FROM WS-REGISTRO-NUEVO-
        MAESTRO
    MOVE "NO" TO MAESTRO-LISTO-PARA-SALIDA

    PERFORM DETERMINA-CLAVE-INFERIOR

PROCESA-UNA-TRANSACCION
    IF COD-TRANS-INSERTA
        IF MAESTRO-LISTO-PARA-SALIDA EQUAL "SI"
            PERFORM CLAVE-DUPPLICADA
        ELSE
            PERFORM COPIA-TRANS-EN-NUEVO-MAESTRO
            MOVE "SI" TO MAESTRO-LISTO-PARA-SALIDA
    ELSE IF COD-TRANS-CAMBIA
        IF MAESTRO-LISTO-PARA-SALIDA EQUAL "NO"
            PERFORM ERROR-NO-HAY-REG-MAESTRO
        ELSE
            PERFORM APLICA-TRANS-NUEVO-MAESTRO
    ELSE IF COD-TRANS-BORRA
        IF MAESTRO-LISTO-PARA-SALIDA EQUAL "NO"
            PERFORM ERROR-NO-SE-PUEDE-BORRAR
        ELSE
            MOVE "NO" TO MAESTRO-LISTO-PARA-SALIDA
    ELSE
        PERFORM ERROR-COD-TRANS-INVALIDO

    PERFORM CONSIGUE-NUEVA-TRANSACCION

```

Fig. 11-2

```

DETERMINA-CLAVE-MENOR
IF CLAVE-TRANSACCION IS LESS THAN CLAVE-VIEJO-MAESTRO
    MOVE CLAVE-TRANSACCION TO CLAVE-MENOR
ELSE
    MOVE CLAVE-VIEJO-MAESTRO TO CLAVE-MENOR

CONSIGUE-NUEVA-TRANSACCION.
READ FICHERO-TRANSACCIONES
AT END
    MOVE HIGH-VALUES TO CLAVE-TRANSACCION

CONSIGUE-SIGUIENTE-MAESTRO.
READ VIEJO-FICHERO-MAESTRO
AT END
    MOVE HIGH-VALUES TO CLAVE-VIEJO-MAESTRO

```

Fig. 11-2 (cont.)

El corazón de dicho algoritmo se halla en el párrafo ACTUALIZACION-FICHERO-MAESTRO. CLAVE-MENOR se igualará siempre al menor de CLAVE-TRANSACCION y de CLAVE-VIEJO-MAESTRO. Si la clave de REGISTRO-VIEJO-MAESTRO es menor o igual a CLAVE-TRANSACCION ("CLAVE-MENOR EQUAL CLAVE-VIEJO-MAESTRO"), entonces REGISTRO-VIEJO-MAESTRO se desplaza a WS-REGISTRO-NUEVO-MAESTRO, que es el área del WORKING-STORAGE donde los registros del nuevo fichero maestro son construidos. MAESTRO-LISTO-PARA-SALIDA es un interruptor del programa que nos indica si hay datos activos en esta área de construcción del registro. El lector debería verificar que es imposible que "MOVE REGISTRO-VIEJO-MAESTRO TO WS-REGISTRO-NUEVO-MAESTRO" se ejecute cuando existe ya un registro activo construyéndose en el área WS-REGISTRO-NUEVO-MAESTRO. Tan pronto como el REGISTRO-VIEJO-MAESTRO se traslade al área de construcción del nuevo registro maestro, se dará entrada al buffer de entrada del VIEJO-FICHERO-MAESTRO a un nuevo registro.

El paso siguiente es determinar si existen transacciones que deban aplicarse al contenido de WS-REGISTRO-NUEVO-MAESTRO. Mientras permanezca la CLAVE-TRANSACCION igual al valor presente de la CLAVE-MENOR, los registros de transacciones se aplicarán al mismo maestro y se les dará entrada y serán procesados. Téngase en cuenta que es la instrucción PERFORM PROCESA-UNA-TRANSACCION UNTIL... la que permite al algoritmo manejar más de una transacción por registro maestro.

Todas las transacciones apropiadas se han aplicado y el algoritmo da salida al registro de WS-REGISTRO-NUEVO-MAESTRO hacia el NUEVO-FICHERO-MAESTRO (en el supuesto de que exista un registro activo en el área de construcción). Entonces el interruptor MAESTRO-LISTO-PARA-SALIDA vuelve a desconectarse.

Finalmente, tras la posible entrada de un REGISTRO-VIEJO-MAESTRO y/o uno o más nuevos REGISTRO-TRANSACCIONES, se ejecutará DETERMINA CLAVE-MENOR de nuevo antes de que se repita ACTUALIZACION-FICHERO-MAESTRO.

**EJEMPLO 11.2** La Tabla 11-1 recoge la aplicación del algoritmo "MAESTRO-TRANSACCIONES" para los ficheros:

FICHERO-TRANSACCIONES: 15, 20, 20, 60  
VIEJO-FICHERO-MAESTRO: 10, 20, 30, 60, 70

Anotaciones paso a paso:

1. El algoritmo comienza leyendo el primer registro de cada fichero y determina la clave menor; el área de elaboración de nuevos registros maestros está vacía.
2. "CLAVE-MENOR EQUAL CLAVE-VIEJO-MAESTRO" es cierto; así que el registro maestro antiguo se copiará en el área de construcción del nuevo maestro, el interruptor se activa con el fin de indicar que el área de trabajo contiene un registro y se da entrada al *siguiente* registro del antiguo maestro.

Tabla 11-1

Ejecución de ACTUALIZACION FICHERO-MAESTRO	CLAVE-MENOR	Valor de REGISTRO-TRANSACCIONES	Valor de REGISTRO-VIEJO-MAESTRO	NUEVO-MAESTRO en el área de trabajo	Interruptor
1.	primero	10	15	10	NO
2.	primero	10	15	vacio 10	SI
3.	primero	10	15	vacio	NO
4.	primero	15	15	vacio	NO
5.	segundo	15	15	vacio	SI
6.	segundo	15	20 (1st)	15	SI
7.	segundo	15	20 (1st)	15	NO
8.	segundo	20	20 (1st)	vacio	NO
9.	tercero	20	20 (1st)	vacio	NO
10.	tercero	20	20 (1st)	20	SI
11.	tercero	20	20 (1st)	30	SI
12.	tercero	20	20 (2nd)	30	SI
13.	tercero	20	60	30	SI
14.	tercero	20	60	30	SI
15.	tercero	30	60	30	NO
16.	cuarto	30	60	30	SI
17.	cuarto	30	60	vacio	NO
18.	cuarto	60	60	vacio	NO
19.	quinto	60	60	60	SI
20.	quinto	60	60	60	NO
21.	quinto	60	HIGH-VALUES	60	NO
22.	quinto	70	HIGH-VALUES	70	NO
23.	sexto	70	HIGH-VALUES	70	SI
24.	sexto	70	HIGH-VALUES	vacio	NO
25.	sexto	HIGH-VALUES	HIGH-VALUES	vacio	NO

3. "CLAVE-MENOR EQUAL CLAVE-TRANSACCION" es falso; así que ninguna transacción será procesada. Como MAESTRO-LISTO-PARA-SALIDA EQUAL "SI", el contenido del área de trabajo (10) se escribirá en el nuevo fichero maestro y el interruptor del maestro preparado se desactivará.
4. DETERMINA-CLAVE-ERROR comparará 15 con 20, seleccionando el 15.
5. En la segunda ejecución del ACTUALIZACION-FICHERO-MAESTRO, CLAVE-MENOR no es igual a CLAVE-VIEJO-MAESTRO; luego no se da entrada a ningún registro del antiguo maestro. Como CLAVE-MENOR es igual a CLAVE-TRANSACCION, se ejecuta PERFORM PROCESA-UNA-TRANSACCION procesando la transacción número 15. Tenga presente que si no se trata de una transacción a insertar se tratará como un error, ya que el interruptor MAESTRO-LISTO contiene "NO". Suponiendo que es una inserción, la información de la transacción se desplazará al área de trabajo y se activará el interruptor MAESTRO-LISTO.
6. PROCESA-UNA-TRANSACCION da entrada al siguiente registro de transacciones.
7. Como CLAVE-MENOR ≠ CLAVE-TRANSACCION, no se ejecutará de nuevo PROCESA-UNA-TRANSACCION. En su lugar el algoritmo comprueba el interruptor MAESTRO-LISTO que está activado; de forma que el 15 se escribe en el nuevo fichero maestro y se desactiva el interruptor.
8. DETERMINA-CLAVE-MENOR comparará 20 con 20, eligiendo el 20.
9. Al iniciar la tercera ejecución de ACTUALIZACION-FICHERO-MAESTRO, CLAVE-MENOR es igual a CLAVE-VIEJO-MAESTRO; así pues, el registro maestro antiguo se desplazará al área de trabajo, el interruptor se activará y se dará entrada al siguiente registro del maestro antiguo.
10. Como CLAVE-MENOR también es igual a CLAVE-TRANSACCION, se ejecuta PROCESA-UNA-TRANSACCION dando lugar al procesamiento de la primera transacción 20. Tenga en cuenta que si se tratara de una inserción se tendría un valor duplicado, ya que MAESTRO-LISTO está activado; suponiendo que fuese una modificación, aplicaríamos el cambio al registro 20 del maestro antiguo, que se haya en el área de trabajo.
11. Se da entrada al siguiente registro de transacciones.
12. Como CLAVE-MENOR sigue siendo igual a la nueva CLAVE-TRANSACCION, se volverá a ejecutar PROCESA-UNA-TRANSACCION, originando la modificación por segunda vez del registro maestro antiguo 20 en el área de trabajo (suponiendo que se trata de una modificación).
13. Se dará entrada al siguiente registro de transacciones.
14. Como CLAVE-MENOR ya no es igual a CLAVE-TRANSACCION, el algoritmo pasa a comprobar MAESTRO-LISTO. Como el interruptor está activado, el registro modificado del área de trabajo (20) se escribirá en el nuevo fichero maestro, desactivándose entonces el interruptor.
15. DETERMINA-CLAVE-MENOR comparará 60 con 30, seleccionando el 30.
16. Al iniciar la cuarta ejecución de ACTUALIZACION-FICHERO-MAESTRO, CLAVE-MENOR es igual a CLAVE-VIEJO-MAESTRO; así pues, el registro del antiguo maestro será desplazado al área de trabajo, se activará el interruptor y se dará entrada al siguiente registro del antiguo maestro.
17. Como CLAVE MENOR ≠ CLAVE TRANSACCION, no se procesará ninguna transacción. Como MAESTRO-LISTO está activado, el contenido del área de trabajo (registro 30) se escribirá en el nuevo fichero maestro y MAESTRO-LISTO se desactivará.
18. DETERMINA-CLAVE-MENOR comparará 60 con 60, seleccionando 60.
19. Al iniciar la quinta ejecución de ACTUALIZACION-FICHERO-MAESTRO, CLAVE-MENOR es igual a CLAVE-VIEJO-MAESTRO; por tanto, el registro del antiguo maestro será desplazado al área de trabajo y el interruptor activado. El siguiente registro del maestro antiguo será introducido.
20. Como CLAVE-MENOR = CLAVE-TRANSACCION, dicha transacción se procesará. Suponiendo que se tratase de una supresión, MAESTRO-LISTO se desactivará.
21. Se da entrada a la siguiente transacción; así se alcanza el fin del fichero, lo que sitúa a la CLAVE-TRANSACCION en HIGH-VALUES. Como CLAVE-MENOR ≠ CLAVE-TRANSACCION, PROCESA-UNA-TRANSACCION no se llevará a cabo nuevamente. Como MAESTRO-LISTO fue desactivado durante el borrado, el contenido del área de trabajo no se escribe en el nuevo fichero maestro.
22. DETERMINA-CLAVE-MENOR comparará HIGH-VALUES con 70, seleccionando 70.
23. Al iniciar la sexta ejecución de ACTUALIZACION-FICHERO-MAESTRO, CLAVE-MENOR = CLAVE-VIEJO-MAESTRO; así pues, el registro del maestro antiguo será desplazado al área de trabajo, activándose

MAESTRO-LISTO y dándose entrada al siguiente registro del maestro antiguo. Así se alcanza el fin del fichero y CLAVE-VIEJO-MAESTRO toma el valor HIGH-VALUES.

24. Como CLAVE-MENOR ≠ CLAVE-TRANSACCION, no se llevará a cabo PROCESA-UNA-TRANSACION. Puesto que MAESTRO-LISTO vale "SI", el contenido del área de trabajo (registro 70) se escribirá en el nuevo fichero maestro, desconectándose MAESTRO-LISTO.
25. DETERMINA-CLAVE-MENOR comparará HIGH-VALUES con HIGH-VALUES, seleccionando HIGH-VALUES. Como CLAVE-MENOR ahora es igual a HIGH-VALUES, no se realizarán más iteraciones de ACTUALIZACION-FICHERO-MAESTRO. Los resultados son:

NUEVO-FICHERO-MAESTRO: 10, 15(insertado), 20(modificado dos veces), 30, 70

Observe que el registro 60 se ha eliminado.

**EJEMPLO 11.3** Utilizando el algoritmo "MAESTRO-TRANSACCIONES", escriba un programa de actualización del fichero secuencial con los datos obtenidos a partir del siguiente registro maestro:

```

01 REGISTRO-MAESTRO-CLIENTES.
  05 ID-CLIENTE          PIC X(4).
  05 NOMBRE-CLIENTE      PIC X(20).
  05 NUMERO-FACTURAS    PIC S9(2)      COMP-3.
  05 DATOS-FACTURA      OCCURS 1 TO 15 TIMES
                                DEPENDING ON NUMERO-FACTURAS
                                ASCENDING KEY IS FECHA-FACTURA.
    10 ID-FACTURA         PIC X(5).
    10 FECHA-FACTURA.
      15 AÑO-FACTURA     PIC 99.
      15 MES-FACTURA     PIC 99.
      15 DIA-FACTURA     PIC 99.
    10 IMPORTE-FACTURA   PIC S9(5)V99  COMP-3.

```

Tanto el fichero maestro como el fichero de transacciones están clasificados por el ID del cliente. Existen tres tipos de registros de transacción:

**Borrado** de un registro de cliente del fichero maestro.

columnas 1-4 : ID del cliente  
 columna 5 : código de transacción "1"  
 columnas 6-80: sin utilizar

**Inserción** del registro de un cliente al fichero maestro.

columnas 1-4 : ID del cliente  
 columna 5 : código de transacción "2"  
 columnas 6-25 : nombre del cliente  
 columnas 26-30: factura ID  
 columnas 31-36: fecha de factura (aammdd)  
 columnas 37-43: cantidad (DISPLAY)  
 columnas 44-80: sin utilizar

**Modificación** de los datos de una factura.

columnas 1-4 : ID del cliente  
 columna 5 : código de transacción "3"  
 columna 6 : código de factura ("I", "P" o "A")  
 columnas 7-11 : ID de la factura  
 columnas 12-17: fecha de la factura (aammd)  
 columnas 18-24: cantidad [S9(5)V99 DISPLAY]  
 columnas 25-80: sin utilizar

El código "I" indica una nueva inserción en la tabla de los datos de factura (conservando la tabla clasificada por fecha de facturación). El código "A" supone la sustitución de una cantidad en la tabla por el importe del registro de transacciones. El código "P" señala la aplicación de un pago al importe de una factura. Si el valor

del importe de una factura es *cero* una vez restado el pago, deben suprimirse todos los datos de la factura; si fuera *negativo*, realice la supresión y escriba además un registro en un FICHERO-DE-CREDITO secuencial que contiene los campos ID-CLIENTE, NOMBRE-CLIENTE, ID-FACTURA (aammdd) y SALDO-CREDITO [S9(5)V99 COMP-3]. Si un registro maestro se quedase sin facturas, suprimale del fichero. Junto al NUEVO-FICHERO-MAESTRO-CLIENTES y al FICHERO-DE-CREDITOS, obtenga un INFORME-DE-ERRORES que incluya cabeceras de página (con número de página y fecha) para cada página, todos los campos de las transacciones para cada registro erróneo y una frase describiendo el error. Los siguientes errores deberán estar contemplados: (1) código de transacción "1" e ID del cliente que no se encuentren en el fichero; (2) código de transacción "2" e ID del cliente que ya se encuentren en el fichero; (3) código "3" e ID del cliente que no se encuentren en el fichero; (4) código "3I" y la factura ID se hayan en la tabla de facturas; (5) código "3A" y la factura ID no se hayan en la tabla de facturas; (6) código "3P" y factura ID no se encuentren en la tabla de facturas o que la fecha de factura de transacción no concuerde con la fecha de factura maestra; (7) que no sean numéricas la fecha de factura de la transacción o el importe de la factura.

Una solución viene dada en la Figura 11-3.

```

00001      IDENTIFICATION DIVISION.
00002
00003      PROGRAM-ID. SEQUPDAT.
00004
00005      AUTHOR. LARRY NEWCOMER.
00006      INSTALLATION. PENN STATE UNIVERSITY, YORK CAMPUS.
00007
00008      DATE-WRITTEN. MAY 1983.
00009      DATE-COMPILED. MAY 9,1983.
00011      SECURITY. NONE.
00012
00013      * ACTUALIZACION FICHERO SECUENCIAL CON EL ALGORITMO
00014      * MAESTRO-TRANS. ENTRADA: VIEJO FICHERO MAESTRO
00015      *                      FICHERO TRANSACCIONES
00016      *                      SALIDA: NUEVO FICHERO MAESTRO
00017      *                      INFORME ERRORES
00018      *                      FICHERO CREDITOS
00020      ENVIRONMENT DIVISION.
00022
00023      CONFIGURATION SECTION.
00024      SOURCE-COMPUTER. IBM-3081.
00025
00026      OBJECT-COMPUTER. IBM-3081.
00027
00028      INPUT-OUTPUT SECTION.
00029
00030      FILE-CONTROL.
00031      SELECT VIEJO-FICHERO-MAESTRO
00032          ASSIGN TO OLDMAST
00033          ORGANIZATION IS SEQUENTIAL
00034          ACCESS IS SEQUENTIAL
00035
00036      SELECT FICHERO-TRANSACCIONES
00037          ASSIGN TO TRANS
00038          ORGANIZATION IS SEQUENTIAL
00039          ACCESS IS SEQUENTIAL
00040
00041      SELECT NUEVO-FICHERO-MAESTRO
00042          ASSIGN TO NEWMAST
00043          ORGANIZATION IS SEQUENTIAL
00044          ACCESS IS SEQUENTIAL
00045
00046
00047      SELECT FICHERO-CREDITOS

```

Fig. 11-3

```
00049      ASSIGN TO CREDITS
00050      ORGANIZATION IS SEQUENTIAL
00051      ACCESS IS SEQUENTIAL
00052
00053
00054      SELECT INFORME-ERRORES
00055          ASSIGN TO ERRORLOG
00056          ORGANIZATION IS SEQUENTIAL
00057          ACCESS IS SEQUENTIAL
00058
00059

00061      DATA DIVISION.

00063      FILE SECTION.
00064
00065      *
00066      *
00067      *
00068      *     USO DE REGISTROS DE LONGITUD VARIABLE PARA AHORRAR ESPACIO
00069      *
00070      *
00071      *
00072      FD VIEJO-FICHERO-MAESTRO
00073          BLOCK CONTAINS 0 RECORDS
00074          RECORD CONTAINS 41 TO 251 CHARACTERS
00075          LABEL RECORDS ARE STANDARD
00076
00077
00078      01 OM-AR-CUSTOMER-RECORD.
00079          05 OM-AR-CUSTOMER-ID      PIC X(4).
00080          05 OM-AR-CUSTOMER-NAME    PIC X(20).
00081          05 OM-AR-NUMBER-INVOICES  PIC S9(2)    COMP-3.
00082      *
00083      *
00084      *
00085      *     TABLA LONGITUD VARIABLE DENTRO DEL REGISTRO
00086      *
00087      *
00088      *
00089          05 OM-AR-INVOICE-DATA      OCCURS 1 TO 15 TIMES
00090                  DEPENDING ON
00091                  OM-AR-NUMBER-INVOICES
00092
00093          10 OM-INVOICE-ID        PIC X(5).
00094          10 OM-INFOICE-DATE.
00095              15 OM-INVOICE-YY      PIC 99.
00096              15 OM-INVOICE-MM      PIC 99.
00097              15 OM-INVOICE-DD      PIC 99.
00098          10 OM-INVOICE-AMOUNT    PIC S9(5)V99  COMP-3.
00099
00100      FD FICHERO-TRANSACTIONS
00101          RECORD CONTAINS 80 CHARACTERS
00102          LABEL RECORDS ARE OMITTED
00103
00104
00105      01 TRANSACTION-RECORD.
00106          05 TRANS-SORT-KEY.
00107              10 TRANS-CUSTOMER-ID  PIC X(4).
00108              10 TRANS-CODE       PIC X.
00109                  88 DELETION-TRANSACTION  VALUE "1".
00110                  88 ADD-TRANSACTION    VALUE "2".
00111                  88 CHANGE-TRANSACTION  VALUE "3".
00112          05 TRANS-VARIABLE-AREA  PIC X(75).
```

Fig. 11-3 (cont.)

```

00113      *
00114      *
00115      *
00116      *   USO DE REDEFINES PARA DESCRIBIR LAS DIVERSAS
00117      *   FORMAS DE REGISTROS DE TRANSACCIONES
00118      *
00119      *
00120      *
00121      05 TRANS-ADD-AREA      REDEFINES TRANS-VARIABLE-AREA.
00122          10 ADD-CUSTOMER-NAME  PIC X(20).
00123          10 ADD-INVOICE-ID   PIC X(5).
00124          10 ADD-INVOICE-DATE  PIC 9(6).
00125          10 ADD-AMOUNT       PIC S9(5)V99.
00126          10 FILLER         PIC X(37).
00127      05 TRANS-CHANGE-AREA  REDEFINES TRANS-VARIABLE-AREA.
00128          10 TRANS-INVOICE-CODE PIC X.
00129              88 INSERTION-CODE  VALUE "I".
00130              88 ADJUSTMENT-CODE VALUE "A".
00131              88 PAYMENT-CODE    VALUE "P".
00132          10 CHANGE-INVOICE-ID  PIC X(5).
00133          10 CHANGE-INVOICE-DATE PIC X(6).
00134          10 CHANGE-AMOUNT     PIC S9(5)V99.
00135          10 CHANGE-AMOUNT-X   REDEFINES CHANGE-AMOUNT
00136                           PIC X(7).
00137          10 FILLER         PIC X(56).
00138
00139
00140      FD NUEVO-FICHERO-MAESTRO
00141          BLOCK CONTAINS 0 RECORDS
00142          RECORD CONTAINS 41 TO 251 CHARACTERS
00143          LABEL RECORDS ARE STANDARD
00144
00145
00146      01 NM-AR-CUSTOMER-RECORD.
00147          05 NM-AR-CUSTOMER-ID  PIC X(4).
00148          05 NM-AR-CUSTOMER-NAME PIC X(20).
00149          05 NM-AR-NUMBER-INVOICES PIC S9(2)      COMP-3.
00150          05 NM-AR-INVOICE-DATA  OCCURS 1 TO 15 TIMES
00151                           DEPENDING ON
00152                           NM-AR-NUMBER-INVOICES
00153
00154          10 NM-INVOICE-ID    PIC X(5).
00155          10 NM-INVOICE-DATE.
00156              15 NM-INVOICE-YY   PIC 99.
00157              15 NM-INVOICE-MM   PIC 99.
00158              15 NM-INVOICE-DD   PIC 99.
00159          10 NM-INVOICE-AMOUNT PIC S9(5)V99  COMP-3.
00160
00161      FD FICHERO-CREDITOS
00162          BLOCK CONTAINS 0 RECORDS
00163          RECORD CONTAINS 39 CHARACTERS
00164          LABEL RECORDS ARE STANDARD
00165
00166
00167      01 CREDIT-BALANCE-RECORD.
00168          05 CREDIT-CUSTOMER-ID  PIC X(4).
00169          05 CREDIT-CUSTOMER-NAME PIC X(20).
00170          05 CREDIT-INVOICE-ID   PIC X(5).
00171          05 CREDIT-INVOICE-DATE PIC 9(6).
00172          05 CREDIT-BALANCE     PIC 9(5)V99  COMP-3.
00173
00174      FD INFORME-ERRORES
00175          RECORD CONTAINS 132 CHARACTERS
00176          LABEL RECORDS ARE OMITTED
00177          LINAGE IS 60

```

Fig. 11-3 (cont.)

```

00178          WITH FOOTING AT 59
00179          LINES AT TOP 3
00180          LINES AT BOTTOM 3
00181
00182
00183      01 ERROR-REPORT-LINE      PIC X(132).
00185
WORKING-STORAGE SECTION.

00187      01 WS-SWITCHES-AND-WORK-AREAS.
00188          05 WS-PAGE-NUMBER      PIC S9(3)      COMP-3.
00189          05 LOW-KEY          PIC X(4).
00190          05 MAXIMUM-TABLE-SIZE  PIC S9(3)      COMP-3
00191                  VALUE +15.
00192          05 MASTER-READY      PIC X(3).
00193                  88 MASTER-RECORD-PRESENT  VALUE "YES".
00194                  88 NO-MASTER-RECORD    VALUE "NO".
00195          05 SYSTEM-DATE.
00196                  10 SYSTEM-YY      PIC 99.
00197                  10 SYSTEM-MM      PIC 99.
00198                  10 SYSTEM-DD      PIC 99.
00199          05 WS-DIFFERENCE    PIC S9(5)V99  COMP-3.
00200
00201      *
00202      *
00203      *
00204      * MENSAGE DE ERROR DEFINIDO COMO CONSTANTE DEL
00205      * PROGRAMA Y NO COMO LITERAL NO-NUMERICO
00206      *
00207      *
00208      *
00209      01 WS-ERROR-MESSAGE-AREAS.
00210          05 ADD-ERROR-MESSAGE    PIC X(42)
00211                  VALUE "DUPLICATE KEY--NO ADD".
00212          05 CHANGE-ERROR-MESSAGE  PIC X(42)
00213                  VALUE "KEY NOT FOUND--NO CHANGE".
00214          05 DELETION-ERROR-MESSAGE  PIC X(42)
00215                  VALUE "KEY NOT FOUND--NO DELETE".
00216          05 INVALID-TRANS-CODE-MESSAGE
00217                  PIC X(42)
00218                  VALUE "INVALID CODE--NO ACTION".
00219          05 INVALID-CHANGE-CODE-MESSAGE
00220                  PIC X(42)
00221                  VALUE "INVALID CHANGE--IGNORED".
00222          05 ADJUST-ERROR-MESSAGE   PIC X(42)
00223                  VALUE "INVOICE NOT FOUND".
00224          05 INVALID-FIELDS-MESSAGE  PIC X(42)
00225                  VALUE "INVALID AMOUNT/DATE".
00226          05 NO-PAYMENT-INVOICE-MESSAGE
00227                  PIC X(42)
00228                  VALUE "NO" INVOICE--PAYMENT NOT CREDITED".
00229          05 DUPLICATE-INVOICE-MESSAGE
00230                  PIC X(42)
00231                  VALUE "DUPLICATE INVOICE--IGNORED".
00232          05 TABLE-OVERFLOW-MESSAGE  PIC X(42)
00233                  VALUE "> 15 INVOICES--IGNORED".
00234
00235      *
00236      *
00237      *
00238      * AREA DE TRABAJO USADA EN EL ALGORITMO PARA
00239      * CONSTRUIR EL REGISTRO DEL NUEVO MAESTRO CON TODAS
00240      * LAS TRANSACCIONES APLICABLES
00241
00242      *
00243      *

```

Fig. 11-3 (cont.)

```

00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308

      01 WS-AR-CUSTOMER-RECORD.
      05 WS-AR-CUSTOMER-ID          PIC X(4).
      05 WS-AR-CUSTOMER-NAME        PIC X(20).
      05 WS-AR-CUSTOMER-INVOICES   PIC S9(2)    COMP-3.
      05 WS-AR-INVOICE-DATA         OCCURS 1 TO 15 TIMES
                                    DEPENDING ON
                                    WS-AR-NUMBER-INVOICES
                                    ASCENDING KEY IS
                                    WS-INVOICE-DATE
                                    INDEXED BY WS-INDEX
                                    WS-INDEX-2

      10 WS-INVOICE-ID              PIC X(5).
      10 WS-INVOICE-DATE.
      15 WS-INVOICE-YY              PIC 99.
      15 WS-INVOICE-MM              PIC 99.
      15 WS-INVOICE-DD              PIC 99.
      10 WS-INVOICE-AMOUNT          PIC S9(5)V99  COMP-3.

      01 WS-ERROR-PAGE-TITLE.
      05 FILLER                   PIC X(20)     VALUE SPACES.
      05 FILLER                   PIC X(25)
                                    VALUE "TRANSACTION ERROR LOG".
      05 FILLER                   PIC X(5)      VALUE "PAGE".
      05 WS-ERROR-PAGE-NUMBER    PIC ZZ9.
      05 FILLER                   PIC XX       VALUE SPACES.
      05 WS-ERROR-PAGE-DATE.
      10 WS-ERROR-MM              PIC Z9.
      10 FILLER                   PIC X        VALUE "/".
      10 WS-ERROR-DD              PIC 99.
      10 FILLER                   PIC X        VALUE "/".
      10 WS-ERROR-YY              PIC 99.
      05 FILLER                   PIC X(69)     VALUE SPACES.

      01 WS-ERROR-HEADING.
      05 FILLER                   PIC X(9)      VALUE "CUSTOMER".
      05 FILLER                   PIC X(5)      VALUE "CODE".
      05 FILLER                   PIC X(118)
                                    VALUE "REST OF TRANSACTION".

      01 WS-ERROR-LINE.
      05 WS-ERROR-CUSTOMER-ID    PIC X(4).
      05 FILLER                   PIC X(5)     VALUE SPACES.
      05 WS-ERROR-CODE            PIC X.
      05 FILLER                   PIC X(4)     VALUE SPACES.
      05 WS-ERROR-REST-OF-TRAN   PIC X(75).
      05 FILLER                   PIC X(1)     VALUE SPACES.
      05 WS-ERROR-MESSAGE         PIC X(42).

      PROCEDURE DIVISION.

      *
      *
      *
      *   IMPLEMENTACION EN COBOL DEL ALGORITMO MAESTRO-
      *   TRANSACCIONES PARA ACTUALIZACION DE FICHEROS
      *
      *
      000-ACTUALIZACION-FICHERO-SEC.

      MOVE "NO" TO MASTER-READY
      MOVE ZERO TO WS-PAGE-NUMBER

```

Fig. 11-3 (cont.)

```

00309    ACCEPT SYSTEM-DATE FROM DATE
00310    MOVE SYSTEM-YY TO WS-ERROR-YY
00311    MOVE SYSTEM-MM TO WS-ERROR-MM
00312    MOVE SYSTEM-DD TO WS-ERROR-DD
00313        OPEN    INPUT          VIEJO-FICHERO-MAESTRO
00314                                FICHERO-TRANSACCIONES
00315        OUTPUT          NUEVO-FICHERO-MAESTRO
00316                                FICHERO-CREDITOS
00317                                INFORME-ERRORES
00318        PERFORM 170-IMPRIME-CABECERA-INFORME
00319        PERFORM 030-LEE-TRANSACCION
00320        PERFORM 040-LEE-MAESTRO
00321        PERFORM 050-DETERMINA-CLAVE-MENOR
00322        PERFORM 010-ACTUALIZACION-FICHERO-MAESTRO
00323            UNTIL LOW-KEY EQUAL HIGH-VALUES
00324
00325        CLOSE   VIEJO-FICHERO-MAESTRO
00326                                FICHERO-TRANSACCIONES
00327                                NUEVO-FICHERO-MAESTRO
00328                                FICHERO-CREDITOS
00329                                INFORME-ERRORES
00330        STOP RUN
00331
00332
00333    010-ACTUALIZACION-FICHERO-MAESTRO.
00334
00335        IF LOW-KEY EQUAL OM-AR-CUSTOMER-ID
00336        *
00337        *
00338        *
00339        *   ILUSTRA EL MANEJO DE TABLAS DE LONGITUD VARIABLE
00340        *
00341        *
00342        *
00343        MOVE OM-AR-NUMBER-INVOICES TO WS-AR-NUMBER-INVOICES
00344        MOVE OM-AR-CUSTOMER-RECORD TO WS-AR-CUSTOMER-RECORD
00345        MOVE "YES" TO MASTER-READY
00346        PERFORM 040-LEE-MAESTRO
00347
00348        IF LOW-KEY EQUAL TRANS-CUSTOMER-ID
00349            PERFORM 020-PROCESA-TRANSACCIONES
00350                UNTIL LOW-KEY NOT EQUAL TRANS-CUSTOMER-ID
00351
00352        IF MASTER-RECORD-PRESENT
00353            PERFORM 060-ESCRIBE-NUEVO-REG-MAESTRO
00354            MOVE "NO" TO MASTER-READY
00355
00356        PERFORM 050-DETERMINA-CLAVE-MENOR
00357
00358
00359    020-PROCESA-TRANSACCIONES.
00360
00361        *
00362        *
00363        *
00364        *   ESTA VERSION DEL ALGORITMO VALIDA LAS TRANSACCIONES
00365        *   ANTES DE APLICARLAS
00366        *
00367        *
00368        *
00369        IF ADD-TRANSACTION
00370            IF MASTER-RECORD-PRESENT
00371                MOVE ADD-ERROR-MESSAGE TO WS-ERROR-MESSAGE
00372                PERFORM 070-ESCRIBE-LINEA-ERROR
00373            ELSE

```

Fig. 11-3 (cont.)

```

00374      IF ADD-INVOICE-DATE NUMERIC AND ADD-AMOUNT NUMERIC
00375          PERFORM 080-COPIA-TRANS-MAESTRO
00376          MOVE "YES" TO MASTER-READY
00377      ELSE
00378          MOVE INVALID-FIELDS-MESSAGE TO WSSS-ERROR-MESSAGE
00379          PERFORM 070-ESCRIBE-LINEA-ERROR
00380      ELSE IF CHANGE-TRANSACTION
00381          IF NO-MASTER-RECORD
00382              MOVE CHANGE-ERROR-MESSAGE TO WS-ERROR-MESSAGE
00383              PERFORM 070-ESCRIBE-LINEA-ERROR
00384      ELSE
00385          IF     CHANGE-INVOICE-DATE NUMERIC
00386              AND CHANGE-AMOUNT NUMERIC
00387              PERFORM APPLY-TRANSACTION-TO-MASTER
00388      ELSE
00389          MOVE INVALID-FIELDS-MESSAGE TO WS-ERROR-MESSAGE
00390          PERFORM 070-ESCRIBE-LINEA-ERROR
00391      ELSE IF DELETION-TRANSACTION
00392          IF NO-MASTER-RECORD
00393              MOVE DELETION-ERROR-MESSAGE TO WS-ERROR-MESSAGE
00394              PERFORM 070-ESCRIBE-LINEA-ERROR
00395      ELSE
00396          MOVE "NO" TO MASTER-READY
00397      ELSE
00398          MOVE INVALID-TRANS-CODE-MESSAGE TO WS-ERROR-MESSAGE
00399          PERFORM 070-ESCRIBE-LINEA-ERROR
00400
00401      PERFORM 030-LEE-TRANSACCION
00402
00403
00404      030-LEE-TRANSACCION.
00405
00406      READ FICHERO-TRANSACCIONES
00407          AT END
00408          MOVE HIGH-VALUES TO TRANS-CUSTOMER-ID
00409
00410
00411      040-LEE-MAESTRO.
00412
00413      READ VIEJO-FICHERO-MAESTRO
00414          AT END
00415          MOVE HIGH-VALUES TO OM-AR-CUSTOMER-ID
00416
00417
00418      050-DETERMINA-CLAVE-MENOR.
00419
00420      IF TRANS-CUSTOMER-ID IS LESS THAN OM-AR-CUSTOMER-ID
00421          MOVE TRANS-CUSTOMER-ID TO LOW-KEY
00422      ELSE
00423          MOVE OM-AR-CUSTOMER-ID TO LOW-KEY
00424
00425
00426      060-ESCRIBE-NUEVO-REG-MAESTRO.
00427
00428      *
00429      *
00430      *
00431      *     COMPRUEBA LA LONGITUD VARIABLE ANTES DE ESCRIBIR
00432      *     EL REGISTRO LOGICO EN EL FICHERO
00433      *
00434      *
00435      *
00436      IF WS-AR-NUMBER-INVOICES POSITIVE
00437          MOVE WS-AR-NUMBER-INVOICES TO NM-AR-NUMBER-INVOICES
00438          MOVE WS-AR-CUSTOMER-RECORD TO NM-AR-CUSTOMER-RECORD
00439          WRITE NM-AR-CUSTOMER-RECORD

```

Fig. 11-3 (cont.)

```

00440
00441
00442      070-ESCRIBE-LINEA-ERROR.
00443
00444          MOVE TRANS-CUSTOMER-ID      TO WS-ERROR-CUSTOMER-ID
00445          MOVE TRANS-CODE        TO WS-ERROR-CODE
00446          MOVE TRANS-VARIABLE-AREA  TO WS-ERROR-REST-OF-TRAN
00447          WRITE ERROR-REPORT-LINE
00448              FROM WS-ERROR-LINE
00449              AFTER ADVANCING 2 LINES
00450              AT END-OF-PAGE
00451                  PERFORM 170-IMPRIME-CABECERA-INFORME
00452
00453
00454      080-COPIA-TRANS-MAESTRO.
00455
00456          MOVE TRANS-CUSTOMER-ID  TO WS-AR-CUSTOMER-ID
00457          MOVE ADD-CUSTOMER-NAME TO WS-AR-CUSTOMER-NAME
00458          MOVE 1                 TO WS-AR-NUMBER-INVOICES
00459          MOVE ADD-INVOICE-ID    TO WS-INVOICE-ID (1)
00460          MOVE ADD-INVOICE-DATE  TO WS-INVOICE-DATE (1)
00461          MOVE ADD-AMOUNT       TO WS-INVOICE-AMOUNT (1)
00462
00463
00464      APLICA-TRANS-A-MAESTRO.
00465
00466          IF INSERTION-CODE
00467              PERFORM 090-INSERTA-FACTURA
00468          ELSE IF ADJUSTMENT-CODE
00469              PERFORM 130-AJUSTA-FACTURA
00470          ELSE IF PAYMENT-CODE
00471              PERFORM 140-APLICA-PAGO
00472          ELSE
00473              MOVE INVALID-CHANGE-CODE-MESSAGE TO WS-ERROR-MESSAGE
00474              PERFORM 070-ESCRIBE-LINEA-ERROR
00475
00476
00477      090-INSERTA-FACTURA.
00478
00479      *
00480      *
00481      *
00482      *     COMPRUEBA SI HAY REBASAMIENTO ANTES DE INSERTAR
00483      *
00484      *
00485      *
00486          IF WS-AR-NUMBER-INVOICES EQUAL MAXIMUM-TABLE-SIZE
00487              MOVE TABLE-OVERFLOW-MESSAGE TO WS-ERROR-MESSAGE
00488              PERFORM 070-ESCRIBE-LINEA-ERROR
00489          ELSE
00490
00491
00492
00493      *     ILUSTRAS USO BUSQUEDA SECUENCIAL
00494
00495
00496      *
00497          SET WS-INDEX TO 1
00498          SEARCH WS-AR-INVOICE-DATA
00499              AT END
00500                  PERFORM 100-ENCUENTRA-Y-MUEVE
00501          WHEN WS-INVOICE-ID (WS-INDEX) EQUAL CHANGE-INVOICE-ID
00502              MOVE DUPLICATE-INVOICE-MESSAGE TO WS-ERROR-MESSAGE
00503              PERFORM 070-ESCRIBE-LINEA-ERROR
00504
00505

```

Fig. 11-3 (cont.)

```

00506      100-ENCUENTRA-Y-MUEVE.
00507
00508      SET WS-INDEX TO 1
00509      SEARCH WS-AR-INVOICE-DATA
00510      AT END
00511          ADD 1 TO WS-AR-NUMBER-INVOICES
00512          PERFORM 120-COPIA-NUEVA-ENTRADA
00513          WHEN WS-INVOICE-DATE (WS-INDEX) GREATER THAN
00514              CHANGE-INVOICE-DATE
00515                  ADD 1 TO WS-AR-NUMBER-INVOICES
00516                  PERFORM 110-MUEVE-HACIA-ABAJO
00517                      VARYING WS-INDEX-2 FROM WS-AR-NUMBER-INVOICES
00518                          BY -1
00519                          UNTIL WS-INDEX-2 EQUAL WS-INDEX
00520                  PERFORM 120-COPIA-NUEVA-ENTRADA
00521
00522
00523      110-MUEVE-HACIA-ABAJO.
00524
00525          MOVE WS-AR-INVOICE-DATA (WS-INDEX-2 - 1) TO
00526              WS-AR-INVOICE-DATA (WS-INDEX-2)
00527
00528
00529      120-COPIA-NUEVA-ENTRADA.
00530
00531          MOVE CHANGE-INVOICE-ID TO WS-INVOICE-ID (WS-INDEX)
00532          MOVE CHANGE-INVOICE-DATE TO WS-INVOICE-DATE (WS-INDEX)
00533          MOVE CHANGE-AMOUNT TO WS-INVOICE-AMOUNT (WS-INDEX)
00534
00535
00536      130-AJUSTA-FACTURA.
00537
00538          SET WS-INDEX TO 1
00539          SEARCH WS-AR-INVOICE-DATA
00540          AT END
00541              MOVE ADJUST-ERROR-MESSAGE TO WS-ERROR-MESSAGE
00542              PERFORM 070-ESCRIBE-LINEA-ERROR
00543              WHEN WS-INVOICE-ID (WS-INDEX) EQUAL CHANGE-INVOICE-ID
00544                  MOVE CHANGE-AMOUNT TO WS-INVOICE-AMOUNT (WS-INDEX)
00545
00546
00547      140-APLICA-PAGO.
00548
00549          SET WS-INDEX TO 1
00550          SEARCH WS-AR-INVOICE-DATA
00551          AT END
00552              MOVE NO-PAYMENT-INVOICE-MESSAGE TO WS-ERROR-MESSAGE
00553              PERFORM 070-ESCRIBE-LINEA-ERROR
00554              WHEN WS-INVOICE-ID (WS-INDEX) EQUAL CHANGE-INVOICE-ID
00555                  PERFORM 150-CALCULA-IMPORTE
00556
00557
00558      150-CALCULA-IMPORTE.
00559
00560          SUBTRACT CHANGE-AMOUNT FROM WS-INVOICE-AMOUNT (WS-INDEX)
00561              GIVING WS-INVOICE-AMOUNT (WS-INDEX)
00562                  WS-DIFFERENCE
00563
00564
00565
00566      *   SI EXcede, SE ESCRIBE EN EL FICHERO DE CREDITOS
00567
00568
00569
00570      *   IF WS-DIFFERENCE NEGATIVE

```

Fig. 11-3 (cont.)

```

00571      MOVE TRANS-CUSTOMER-ID      TO CREDIT-CUSTOMER-ID
00572      MOVE WS-AR-CUSTOMER-NAME   TO CREDIT-CUSTOMER-NAME
00573      MOVE CHANGE-INVOICE-ID    TO CREDIT-INVOICE-ID
00574      MOVE CHANGE-INVOICE-DATE   TO CREDIT-INVOICE-DATE
00575      MOVE WS-DIFFERENCE       TO CREDIT-BALANCE
00576      WRITE CREDIT-BALANCE-RECORD
00577
00578      *
00579      * -----
00580      *
00581      * SI ESTA PAGADA, SE ELIMINA LA FACTURA DE LA TABLA
00582      *
00583      *
00584      *
00585      IF WS-DIFFERENCE NOT POSITIVE
00586          PERFORM 160-BORRA-FACTURA
00587              VARYING WS-INDEX FROM WS-INDEX BY 1
00588                  UNTIL WS-INDEX GREATER THAN WS-AR-NUMBER-INVOICES
00589          SUBTRACT 1 FROM WS-AR-NUMBER-INVOICES
00590
00591
00592      160-BORRA-FACTURA.
00593
00594      MOVE WS-AR-INVOICE-DATA (WS-INDEX + 1) TO
00595          WS-AR-INVOICE-DATA (WS-INDEX)
00596
00597
00598      170-IMPRIME-CABECERA-INFORME.
00599
00600          ADD 1 TO WS-PAGE-NUMBER
00601          MOVE WS-PAGE-NUMBER TO WS-ERROR-PAGE-NUMBER
00602          WRITE ERROR-REPORT-LINE
00603              FROM WS-ERROR-PAGE-TITLE
00604                  AFTER ADVANCING PAGE
00605          WRITE ERROR-REPORT-LINE
00606              FROM WS-ERROR-HEADING
00607                  AFTER ADVANCING 2 LINES
00608

```

Fig. 11-3 (cont.)

### 11.3 ACTUALIZACION DE FICHEROS EN DISCO

Cuando un fichero maestro organizado secuencialmente está almacenado en disco, es posible actualizar registros lógicos sin necesidad de crear un nuevo fichero maestro: esto se consigue mediante la lectura de los registros y su traslado a la memoria principal, después se procede a realizar las modificaciones oportunas y se vuelven a escribir en el lugar exacto del disco donde originalmente fueron encontrados. Vea el diagrama de flujo del sistema en la Figura 11-4. Este proceso está sujeto a las siguientes restricciones: (i) no podrán insertarse registros en el fichero; (ii) las supresiones habrán de realizarse lógicamente (señalizando los registros con un código de supresión); (iii) la longitud de un registro lógico no podrá alterarse; (iv) el fichero deberá ser abierto en modo entrada-salida (véase el Capítulo 6).

En COBOL, el verbo REWRITE (reescribir) se utiliza para escribir un registro lógico en su correspondiente lugar en el disco de donde acaba de ser leído (READ). Su formato es

REWRITE nombre-registro [FROM identificador]

#### EJEMPLO 11.4

```

FD  FICHERO-MAESTRO
01  REGISTRO-MAESTRO...

```

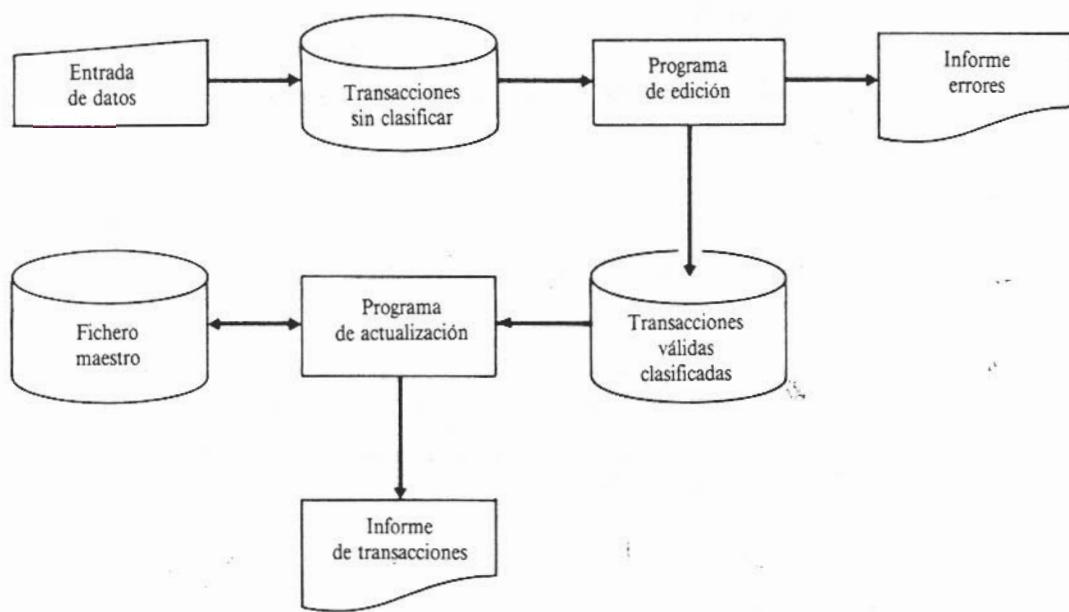


Fig. 11-4

```

OPEN I-O FICHERO-MAESTRO
READ FICHERO-MAESTRO AT END...
PERFORM CAMBIOS-REGISTRO-MAESTRO
REWRITE REGISTRO-MAESTRO
  
```

Tenga en cuenta que para realizar una actualización in situ, el fichero deberá estar abierto en modo entrada-salida, lo que permite el funcionamiento normal de las instrucciones READ y REWRITE para ser utilizadas en la lectura y escritura de registros lógicos en sus correspondientes emplazamientos físicos en el disco.

### Preguntas de repaso

- 11.1 ¿En qué consisten los siguientes procesos: (a) creación de ficheros, (b) acceso a un fichero, (c) mantenimiento de ficheros (actualización)?
- 11.2 Exponga las tres operaciones que deberán realizarse a lo largo del mantenimiento de un fichero.
- 11.3 ¿Cuál es el propósito de un área FILE STATUS? ¿Cómo se define y cómo se utiliza?
- 11.4 Defina: (a) fichero maestro, (b) fichero de transacciones, (c) clave (campo).
- 11.5 Indique el orden de pasos a seguir en una típica actualización de fichero secuencial.
- 11.6 Explique los términos “fichero maestro antiguo” y “fichero maestro nuevo”.
- 11.7 Exponga el método de backup “abuelo, padre, hijo”.
- 11.8 ¿Qué es un registro de transacciones?
- 11.9 Dibuje un diagrama de flujo para una típica actualización de fichero secuencial.
- 11.10 Explique el algoritmo “MAESTRO-TRANSACCIONES”, utilizando pseudocódigo.
- 11.11 ¿Qué se entiende por “actualización in situ”?

- 11.12 Contraste la apertura (OPENing) de un fichero en los modos siguientes: (a) INPUT, (b) OUTPUT, (c) I-O.
- 11.13 ¿Qué limitaciones habrán de establecerse para la actualización in situ de un fichero secuencial?
- 11.14 Explique el uso del verbo REWRITE.
- 11.15 Dibuje un diagrama de flujo para una actualización in situ de un fichero secuencial.
- 11.16 Exponga una ventaja y una desventaja de la actualización in situ frente al método clásico (maestro-antiguo/maestro-nuevo). (*Pista:* considere registros invariables y el tema del backup.)

### Problemas resueltos

- 11.17 ¿Cuál es la diferencia entre *actualización de ficheros* y *actualización de registros*?

La “actualización de ficheros” hace referencia al mantenimiento del fichero, en el cual los registros lógicos se añaden a un fichero, se suprime de un fichero y/o ven sus contenidos modificados. “Actualización de registros”, sin embargo, se refiere al proceso de realización de cambios en los registros lógicos durante la actualización de ficheros.

- 11.18 ¿Qué está mal en los datos de la siguiente prueba de actualización de fichero secuencial?

FICHERO-TRANSACCIONES: 4, 23, 15, 12, 30, 50  
FICHERO-MAESTRO: 10, 20, 30, 40, 50, 60, 70

El fichero de transacciones deberá estar clasificado por orden de la misma clave que el fichero maestro. Los registros de transacciones deberán contener la clave del registro del fichero maestro al cual pretenden actualizar.

- 11.19 En relación al Problema 11.12, dé una cuarta opción OPEN.

OPEN EXTEND, donde los nuevos registros podrán añadirse al final del fichero, utilizando el verbo WRITE.

- 11.20 ¿Cuántas generaciones de transacciones y de ficheros maestros antiguos deberán conservarse y durante cuánto tiempo, con la utilización del método de backup “abuelo, padre, hijo”?

No hay respuesta única: la importancia de la aplicación y de los datos para el negocio, las necesidades legales impuestas por el gobierno, los requisitos contables, etc., todo ello deberá ser soportado.

- 11.21 ¿Cómo se maneja el backup cuando los ficheros son actualizados in situ?

Una copia del fichero podrá hacerse antes que se incorporen las transacciones. Esta copia podrá conservarse así como las transacciones y ser utilizadas para reconstruir el fichero maestro en caso de infortunio.

- 11.22 Los Problemas 8.30 y 8.31 configuran lo que es una secuencia de actualización de fichero secuencial, siendo el Problema 8.30 el programa editor y el Problema 8.31 el que realiza la actualización del fichero. Revise el Problema 8.30 para utilizar la tabla de número de días en cada mes para validar al campo fecha en las entradas de tarjetas de tiempo. (Ignore tanto los casos de cambio de siglo como los de años bisiestos.)

La Figura 11-5 proporciona un programa COBOL parcial. Las líneas 110-126 definen la tabla de fechas y las líneas 269-290 y 299-312 realizan la comprobación de la entrada.

(LA TABLA SIGUIENTE ESTA EN EL WORKING-STORAGE) . . .

```

00110      01 WS-TABLA-FECHAS-VALIDAS.
00111          05 VALORES-DIAS-POR-MES.
00112              10 FILLER          PIC 99      VALUE 31.
00113              10 FILLER          PIC 99      VALUE 28.
00114              10 FILLER          PIC 99      VALUE 31.
00115              10 FILLER          PIC 99      VALUE 30.
00116              10 FILLER          PIC 99      VALUE 31.
00117              10 FILLER          PIC 99      VALUE 30.
00118              10 FILLER          PIC 99      VALUE 31.
00119              10 FILLER          PIC 99      VALUE 31.
00120              10 FILLER          PIC 99      VALUE 30.
00121              10 FILLER          PIC 99      VALUE 31.
00122              10 FILLER          PIC 99      VALUE 30.
00123              10 FILLER          PIC 99      VALUE 31.
00124          05 DIAS-POR-MES      REDEFINES VALORES-DIAS-POR-MES
00125                  PIC 99
00126                  OCCURS 12 TIMES.
.
.
```

(LO SIGUIENTE SON PARRAFOS DE LA DIVISION DE PROCEDIMIENTOS) . . .

```

00269      240-VALIDA-CAMPOS.
00270
00271          MOVE "NO"    TO WS-INTER-ERROR-DETECTADO
00272          MOVE SPACES TO WS-LINEA-ABAJO
00273          IF ID-TIEMPO-TARJETA NOT NUMERIC
00274              MOVE ALL "-" TO WS-ID-ABAJO
00275              MOVE "SI"     TO WS-INTER-ERROR-DETECTADO
00276
00277          IF HORAS-TIEMPO-TARJETA NOT NUMERIC
00278              MOVE ALL "-" TO WS-HORAS-ABAJO
00279              MOVE "SI"     TO WS-INTER-ERROR-DETECTADO
00280
00281          IF FECHA-CIERRE-TIEMPO-TARJETA NOT NUMERIC
00282              MOVE ALL "-" TO WS-FECHA-ABAJO
00283              MOVE "SI"     TO WS-INTER-ERROR-DETECTADO
00284          ELSE
00285              PERFORM 260-VALIDA-FECHA
00286
00287          IF DEPARTAMENTO-TIEMPO-TARJETA NOT NUMERIC
00288              MOVE ALL "-" TO WS-DEPARTAMENTO-ABAJO
00289              MOVE "SI"     TO WS-INTER-ERROR-DETECTADO
00290
00291
00292      260-VALIDA-FECHA.
00293
00294          IF AÑO-TIEMPO-TARJETA GREATER THAN SYSTEM-YY
00295              MOVE ALL "-" TO WS-AÑO-ABAJO
00296              MOVE "SI"     TO WS-INTER-ERROR-DETECTADO
00297
00298          IF MES-TIEMPO-TARJETA LESS THAN 1 OR GREATER THAN 12
00299              MOVE ALL "-" TO WS-MES-ABAJO
00300              MOVE "SI"     TO WS-INTER-ERROR-DETECTADO
00301          ELSE IF DIA-TIEMPO-TARJETA LESS THAN 1 OR
00302              GREATER THAN DIAS-POR-MES (MES-TIEMPO-TARJETA)
00303              MOVE ALL "-" TO WS-DIA-ABAJO
00304              MOVE "SI"     TO WS-INTER-ERROR-DETECTADO
00305
00306
00307
00308
00309
00310
00311
00312
.
```

Fig. 11-5

- 11.23** Diseñe un programa en COBOL para el Problema 8.31. Utilice un listado de tipos impositivos para calcular la imposición de la nómina, y actualizar el fichero maestro de empleados (*in situ*) añadiendo los campos salario bruto, importe del impuesto y salario neto a los campos del registro del maestro de empleados.

Vea la Figura 11-6. El cómputo impositivo se halla en las líneas 112-137 y 331-343; la actualización, en las líneas 234-256 y 309-315.

```

00001 IDENTIFICATION DIVISION.
00002
00003 PROGRAM-ID. PAYROLL.
00004
00005 AUTHOR. LARRY NEWCOMER.
00006 INSTALLATION. PENN STATE UNIVERSITY -- YORK CAMPUS.
00007
00008 ENVIRONMENT DIVISION.
00009
00010 CONFIGURATION SECTION.
00011 SOURCE-COMPUTER. IBM-3081.
00012 OBJECT-COMPUTER. IBM-3081.
00013
00014 INPUT-OUTPUT SECTION.
00015 FILE-CONTROL.
00016
00017   SELECT FICHERO-TIEMPO-VALIDO
00018     ASSIGN TO DISKTIME
00019     ORGANIZATION IS SEQUENTIAL
00020     ACCESS IS SEQUENTIAL
00021
00022   SELECT FICHERO-MAESTRO-EMPLEADOS
00023     ASSIGN TO EMPMAST
00024     ORGANIZATION IS SEQUENTIAL
00025     ACCESS IS SEQUENTIAL
00026
00027   SELECT FICHERO-CHEQUES-NOMINA
00028     ASSIGN TO PAYCHECK
00029     ORGANIZATION IS SEQUENTIAL
00030     ACCESS IS SEQUENTIAL
00031
00032   SELECT FICHERO-INFORME-NOMINA
00033     ASSIGN TO REGISTER
00034     ORGANIZATION IS SEQUENTIAL
00035     ACCESS IS SEQUENTIAL
00036
00037 DATA DIVISION.
00038
00039 FILE SECTION.
00040
00041   FD FICHERO-TIEMPO-VALIDO
00042     BLOCK CONTAINS 0 RECORDS
00043     RECORD CONTAINS 80 CHARACTERS
00044     LABEL RECORDS ARE STANDARD
00045
00046
00047   01 VALID-TIME-RECORD.
00048     05 VALID-TIME-ID          PIC X(5).
00049     05 VALID-TIME-HOURS      PIC S9(2)V9.
00050     05 VALID-TIME-CLOSING-DATE PIC X(6).
00051     05 VALID-TIME-DEPARTMENT PIC X(4).
00052     05 VALID-TIME-EDIT-DATE  PIC X(6).
00053     05 FILLER                PIC X(56).
00054
00055

```

Fig. 11-6

```

00056      FD  FICHERO-MAESTRO-EMPLEADOS
00057          BLOCK CONTAINS 0 RECORDS
00058          RECORD CONTAINS 80 CHARACTERS
00059          LABEL RECORDS ARE STANDARD
00060
00061
00062      01  EMPLOYEE-MASTER-RECORD.
00063          05  EMPLOYEE-MASTER-ID      PIC X(5).
00064          05  EMPLOYEE-MASTER-NAME    PIC X(20).
00065          05  EMPLOYEE-MASTER-RATE   PIC S9(3)V99.
00066          05  EMPLOYEE-MASTER-YTD-GROSS
00067                      PIC S9(7)V99.
00068          05  EMPLOYEE-MASTER-YTD-TAX  PIC 9(6)V99.
00069          05  EMPLOYEE-MASTER-YTD-NET  PIC 9(7)V99.
00070          05  FILLER             PIC X(24).
00071
00072      FD  FICHERO-CHEQUES-NOMINA
00073          BLOCK CONTAINS 0 RECORDS
00074          RECORD CONTAINS 66 CHARACTERS
00075          LABEL RECORDS ARE STANDARD
00076
00077
00078      01  PAYROLL-CHECK-RECORD.
00079          05  PAYROLL-ID           PIC X(5).
00080          05  PAYROLL-HOURS        PIC S9(2)V9.
00081          05  PAYROLL-DATE         PIC X(6).
00082          05  PAYROLL-DEPARTMENT   PIC X(4).
00083          05  PAYROLL-RATE         PIC S9(3)V99.
00084          05  PAYROLL-NAME         PIC X(20).
00085          05  PAYROLL-GROSS        PIC S9(6)V99.
00086          05  PAYROLL-TAX          PIC S9(5)V99.
00087          05  PAYROLL-NET          PIC S9(6)V99.
00088
00089      FD  FICHERO-INFORME-NOMINA
00090          RECORD CONTAINS 132 CHARACTERS
00091          LABEL RECORDS ARE OMITTED
00092
00093
00094      01  ERROR-REPORT-LINE      PIC X(132).
00095
00096      WORKING-STORAGE SECTION.
00097
00098      01  PROGRAM-SWITCHES.
00099          05  WS-END-TIME-FILE-SW  PIC X(3).
00100          88  NO-MORE-TIMES       VALUE "YES".
00101          88  TIME-AVAILABLE     VALUE "NO".
00102          05  WS-END-MASTER-FILE-SW PIC X(3).
00103          88  NO-MORE-MASTERS    VALUE "YES".
00104          88  MASTER-AVAILABLE    VALUE "NO".
00105
00106      01  PROGRAM-COUNTERS.
00107          05  WS-PAGE-NUMBER     PIC S9(3)      COMP-3.
00108          05  WS-NUMBER-EMPLOYEES PIC S9(5)      COMP-3.
00109          05  WS-NUMBER-TO-SKIP   PIC S9         COMP SYNC.
00110          05  WS-LINES-ON-PAGE    PIC S9(2)     COMP SYNC.
00111
00112      *
00113      *
00114      * LA TABLA DE IMPUESTOS SE USA PARA EXAMINARLA. COMO
00115      * EL ULTIMO ELEMENTO = ZERO, LA BUSQUEDA TENDRA EXITO
00116      * SIEMPRE. AUNQUE SE UTILIZAN VALUE Y REDEFINES
00117      * PARA INICIALIZAR ESTA TABLA, TAMBIEN PODRIA
00118      * HABERSE HECHO CON ENTRADA DESDE UN FICHERO
00119      *
00120      *

```

Fig. 11-6 (cont.)

```

00121
00122      01 TAX-DATA.
00123          05 TAX-TABLE-VALUES.
00124              10 FILLER      PIC S9(3)V99  COMP-3 VALUE +800.00.
00125              10 FILLER      PIC SV999   COMP-3 VALUE +.25.
00126              10 FILLER      PIC S9(3)V99  COMP-3 VALUE +500.00.
00127              10 FILLER      PIC SV999   COMP-3 VALUE +.15.
00128              10 FILLER      PIC S9(3)V99  COMP-3 VALUE +300.00.
00129              10 FILLER      PIC SV999   COMP-3 VALUE +.08.
00130              10 FILLER      PIC S9(3)V99  COMP-3 VALUE ZERO.
00131              10 FILLER      PIC SV999   COMP-3 VALUE +.05.
00132          05 TAX-TABLE      REDEFINES TAX-TABLE-VALUES
00133                      OCCURS 4 TIMES
00134                      INDEXED BY TAX-INDEX.
00135              10 LOWER-LIMIT  PIC S9(3)V99  COMP-3.
00136              10 TAX-RATE     PIC SV999   COMP-3.
00137
00138      01 PROGRAM-TOTAL-AREAS.
00139          05 WS-TOTAL-HOURS    PIC S9(7)V9   COMP-3.
00140
00141      01 PROGRAM-COMPUTATION-AREAS.
00142          05 WS-GROSS-PAY      PIC S9(7)V99  COMP-3.
00143          05 WS-NET-PAY       PIC S9(7)V99  COMP-3.
00144          05 WS-TAX          PIC S9(5)V99  COMP-3.
00145
00146      01 PROGRAM-CONSTANTS.
00147          05 WS-PAGE-SIZE     PIC S9(2)    COMP SYNC
00148                      VALUE +50.
00149          05 WS-SKIP-BEFORE-HEADING  PIC S9(2)    COMP SYNC
00150                      VALUE +2.
00151          05 WS-SKIP-BEFORE-FOOTING  PIC S9(2)    COMP SYNC
00152                      VALUE +4.
00153          05 WS-SKIP-BEFORE-DETAIL  PIC S9(2)    COMP SYNC
00154                      VALUE +2.
00155          05 WS-UNMATCHED-TIME-MESSAGE
00156                      PIC X(30)
00157                      VALUE "EMPLOYEE NOT ON FILE".
00158          05 WS-UNMATCHED-MASTER-MESSAGE
00159                      PIC X(30)
00160                      VALUE "INACTIVE THIS PAY PERIOD".
00161          05 WS-OVERTIME-POINT    PIC S99V9   COMP-3
00162                      VALUE +40.0.
00163          05 WS-OVERTIME-FACTOR   PIC S9V9   COMP-3
00164                      VALUE +1.5.
00165
00166      01 WS-SYSTEM-DATE.
00167          05 SYSTEM-YY        PIC 99.
00168          05 SYSTEM-MM        PIC 99.
00169          05 SYSTEM-DD        PIC 99.
00170
00171      01 WS-HEADING-LINE-1.
00172          05 FILLER          PIC X(2)    VALUE SPACES.
00173          05 FILLER          PIC X(18)
00174                      VALUE "PAYROLL REGISTER".
00175          05 WS-HEADING-MM    PIC Z9.
00176          05 FILLER          PIC X      VALUE "/".
00177          05 WS-HEADING-DD    PIC 99.
00178          05 FILLER          PIC X      VALUE "/".
00179          05 WS-HEADING-YY    PIC 99.
00180          05 FILLER          PIC X(3)    VALUE SPACES.
00181          05 FILLER          PIC X(5)    VALUE "PAGE ".
00182          05 WS-HEADING-PAGE  PIC ZZ9.
00183          05 FILLER          PIC X(93)   VALUE SPACES.
00184
00185      01 WS-HEADING-LINE-2.
00186          05 FILLER          PIC X(7)    VALUE " ID".

```

Fig. 11-6 (cont.)

```

00187      05 FILLER          PIC X(5)      VALUE "HRS".
00188      05 FILLER          PIC X(8)      VALUE " DATE".
00189      05 FILLER          PIC X(6)      VALUE "DEPT".
00190      05 FILLER          PIC X(8)      VALUE " RATE".
00191      05 FILLER          PIC X(98)     VALUE "NET PAY".
00192
00193      01 WS-DETAIL-LINE.
00194          05 WS-DETAIL-ID    PIC X(5).
00195          05 FILLER         PIC X(1)      VALUE SPACES.
00196          05 WS-DETAIL-HOURS PIC ZZ.9.
00197          05 FILLER         PIC X(2)      VALUE SPACES.
00198          05 WS-DETAIL-DATE  PIC X(6).
00199          05 FILLER         PIC X(2)      VALUE SPACES.
00200          05 WS-DETAIL-DEPARTMENT PIC X(4).
00201          05 FILLER         PIC X(2)      VALUE SPACES.
00202          05 WS-DETAIL-RATE   PIC ZZZ.99.
00203          05 FILLER         PIC X(2)      VALUE SPACES.
00204          05 WS-DETAIL-REMARKS.
00205              10 WS-DETAIL-PAY  PIC ZZZ,ZZZ.99.
00206              10 FILLER        PIC X(88).
00207
00208      01 WS-FOOTING-LINE.
00209          05 FILLER         PIC X(12)
00210                  VALUE "# EMPLOYEES=".
00211          05 WS-FOOTING-COUNT PIC ZZ,ZZ9.
00212          05 FILLER         PIC X(3)      VALUE SPACES.
00213          05 FILLER         PIC X(12)
00214                  VALUE "TOTAL HOURS=".
00215          05 WS-FOOTING-TOTAL PIC Z,ZZZ,ZZZ.9.
00216          05 FILLER         PIC X(88)     VALUE SPACES.
00217
00218      01 WS-OUTPUT-AREA    PIC X(132).
00219
00220      PROCEDURE DIVISION.
00221
00222      CREA-FICHERO-CHEQUES-NOMINA.
00223
00224          PERFORM 100-INICIALIZACION
00225          PERFORM 200-PROCESA-TARJETAS
00226              UNTIL NO-MORE-TIMES AND NO-MORE-MASTERS
00227          PERFORM 300-TERMINO
00228          STOP RUN
00229
00230
00231      100-INICIALIZACION.
00232
00233          OPEN    INPUT   FICHERO-TIEMPO-VALIDO
00234          I-O      FICHERO-MAESTRO-EMPLEADOS
00235          OUTPUT  FICHERO-INFORME-NOMINA
00236                  FICHERO-CHEQUES-NOMINA
00237          MOVE "NO" TO      WS-END-TIME-FILE-SW
00238                  MOVE ZERO TO      WS-END-MASTER-FILE-SW
00239
00240          MOVE ZERO TO      WS-PAGE-NUMBER
00241                  MOVE WS-PAGE-SIZE TO  WS-NUMBER-EMPLOYEES
00242                  ACCEPT WS-SYSTEM-DATE FROM DATE
00243          MOVE SYSTEM-YY TO  WS-HEADING-YY
00244          MOVE SYSTEM-MM TO  WS-HEADING-MM
00245          MOVE SYSTEM-DD TO  WS-HEADING-DD
00246
00247
00248          PERFORM 270-CONSIGUE-REG-TIEMPO
00249          PERFORM 280-CONSIGUE-REG-MAESTRO
00250
00251      200-PROCESA-TARJETAS.
00252

```

Fig. 11-6 (cont.)

```

00253      MOVE SPACES TO WS-DETAIL-LINE
00254      IF VALID-TIME-ID EQUAL EMPLOYEE-MASTER-ID
00255          PERFORM 210-CREA-REG-NOMINA
00256          PERFORM 215-ACTUALIZA-FICHERO-MAESTRO
00257          PERFORM 270-CONSIGUE-REG-TIEMPO
00258          PERFORM 280-CONSIGUE-REG-MAESTRO
00259      ELSE IF VALID-TIME-ID LESS THAN EMPLOYEE-MASTER-ID
00260          MOVE WS-UNMATCHED-TIME-MESSAGE
00261              TO WS-DETAIL-REMARKS
00262          MOVE VALID-TIME-ID      TO WS-DETAIL-ID
00263          MOVE VALID-TIME-HOURS   TO WS-DETAIL-HOURS
00264          MOVE VALID-TIME-CLOSING-DATE
00265              TO WS-DETAIL-DATE
00266          MOVE VALID-TIME-DEPARTMENT
00267              TO WS-DETAIL-DEPARTMENT
00268          MOVE WS-SKIP-BEFORE-DETAIL
00269              TO WS-NUMBER-TO-SKIP
00270          MOVE WS-DETAIL-LINE      TO WS-OUTPUT-AREA
00271          PERFORM 270-CONSIGUE-REG-TIEMPO
00272      ELSE
00273          MOVE WS-UNMATCHED-MASTER-MESSAGE
00274              TO WS-DETAIL-REMARKS
00275          MOVE EMPLOYEE-MASTER-ID    TO WS-DETAIL-ID
00276          MOVE EMPLOYEE-MASTER-RATE  TO WS-DETAIL-RATE
00277          MOVE WS-SKIP-BEFORE-DETAIL TO WS-NUMBER-TO-SKIP
00278          MOVE WS-DETAIL-LINE      TO WS-OUTPUT-AREA
00279          PERFORM 280-CONSIGUE-REG-MAESTRO
00280
00281      PERFORM 260-IMPRIME-REGISTRO
00282
00283
00284      210-CREA-REG-NOMINA.
00285
00286          MOVE VALID-TIME-ID      TO WS-DETAIL-ID
00287          MOVE VALID-TIME-HOURS   TO WS-DETAIL-HOURS
00288          MOVE VALID-TIME-CLOSING-DATE TO WS-DETAIL-DATE
00289          MOVE VALID-TIME-DEPARTMENT TO WS-DETAIL-DEPARTMENT
00290          MOVE EMPLOYEE-MASTER-RATE  TO WS-DETAIL-RATE
00291          MOVE WS-SKIP-BEFORE-DETAIL TO WS-NUMBER-TO-SKIP
00292
00293          PERFORM 220-CALCULA-PAGO
00294          MOVE WS-NET-PAY        TO WS-DETAIL-PAY
00295          PERFORM 250-ESCRIBE-REG-NOMINA
00296          MOVE WS-DETAIL-LINE TO WS-OUTPUT-AREA
00297          ADD1                  TO WS-NUMBER-EMPLOYEES
00298          ADD VALID-TIME-HOURS   TO WS-TOTAL-HOURS
00299
00300
00301      220-CALCULA-PAGO.
00302
00303          PERFORM 230-CALCULA-SALARIO-BRUTO
00304          PERFORM 240-CALCULA-IMPUESTOS
00305          SUBTRACT WS-TAX FROM WS-GROSS-PAY
00306              GIVING WS-NET-PAY
00307
00308
00309      215-ACTUALIZA-FICHERO-MAESTRO.
00310
00311          ADD WS-TAX            TO EMPLOYEE-MASTER-YTD-TAX
00312          ADD WS-GROSS-PAY      TO EMPLOYEE-MASTER-YTD-GROSS
00313          ADD WS-NET-PAY        TO EMPLOYEE-MASTER-YTD-NET
00314          REWRITE EMPLOYEE-MASTER-RECORD
00315
00316
00317      230-CALCULA-SALARIO-BRUTO.
00318

```

Fig. 11-6 (cont.)

```

00319 IF VALID-TIME-HOURS GREATER THAN WS-OVERTIME-POINT
00320   COMPUTE WS-GROSS-PAY =
00321     (WS-OVERTIME-POINT * EMPLOYEE-MASTER-RATE)
00322     +
00323       ((VALID-TIME-HOURS - WS-OVERTIME-POINT)
00324         * EMPLOYEE-MASTER-RATE * WS-OVERTIME-FACTOR)
00325 ELSE
00326   COMPUTE WS-GROSS-PAY =
00327     VALID-TIME-HOURS * EMPLOYEE-MASTER-RATE
00328
00329   240-CALCULA-IMPUESTOS
00330
00331   *
00332   *
00333   * USA BUSQUEDA SECUENCIAL PARA LA TABLA DE IMPUESTOS
00334   * (POR EL DISEÑO NO PUEDE OCURRIR AT END...)
00335   *
00336   *
00337
00338   SET TAX-INDEX TO 1
00339   SEARCH TAX-TABLE
00340     WHEN WS-GROSS-PAY GREATER THAN LOWER-LIMIT (TAX-INDEX)
00341       COMPUTE WS-TAX =
00342         WS-GROSS-PAY * TAX-RATE (TAX-INDEX)
00343
00344
00345   250-ESCRIBE-REG-NOMINA.
00346
00347   MOVE VALID-TIME-ID      TO PAYROLL-ID
00348   MOVE VALID-TIME-HOURS  TO PAYROLL-HOURS
00349   MOVE VALID-TIME-CLOSING-DATE TO PAYROLL-DATE
00350   MOVE VALID-TIME-DEPARTMENT TO PAYROLL-DEPARTMENT
00351   MOVE EMPLOYEE-MASTER-RATE TO PAYROLL-RATE
00352   MOVE EMPLOYEE-MASTER-NAME TO PAYROLL-NAME
00353   MOVE WS-GROSS-PAY        TO PAYROLL-GROSS
00354   MOVE WS-TAX              TO PAYROLL-TAX
00355   MOVE WS-NET-PAY          TO PAYROLL-NET
00356   WRITE PAYROLL-CHECK-RECORD
00357
00358
00359   260-IMPRIME-REGISTRO
00360
00361   IF WS-LINES-ON-PAGE + WS-NUMBER-TO-SKIP
00362     GREATER THAN WS-PAGE-SIZE
00363       ADD 1 TO WS-PAGE-NUMBER
00364       MOVE WS-PAGE-NUMBER TO WS-HEADING-PAGE
00365       WRITE ERROR-REPORT-LINE
00366         FROM WS-HEADING-LINE-1
00367         AFTER ADVANCING PAGE
00368       WRITE ERROR-REPORT-LINE
00369         FROM WS-HEADING-LINE-2
00370         AFTER ADVANCING WS-SKIP-BEFORE-HEADING LINES
00371       MOVE WS-SKIP-BEFORE-HEADING TO WS-LINES-ON-PAGE
00372
00373       WRITE ERROR-REPORT-LINE
00374         FROM WS-OUTPUT-AREA
00375           AFTER ADVANCING WS-NUMBER-TO-SKIP LINES
00376           ADD WS-NUMBER-TO-SKIP TO WS-LINES-ON-PAGE
00377
00378
00379   270-CONSIGUE-REG-TIEMPO
00380
00381   READ FICHERO-TIEMPO-VALIDO
00382     AT END
00383       MOVE "YES" TO WS-END-TIME-FILE-SW

```

Fig. 11-6 (cont.)

```
00384      MOVE HIGH-VALUES TO VALID-TIME-ID
00385
00386
00387      280-CONSIGUE-REG-MAESTRO.
00388
00389      READ FICHERO-MAESTRO-EMPLEADOS
00390          AT END
00391              MOVE "YES" TO WS-END-MASTER-FILE-SW
00392              MOVE HIGH-VALUES TO EMPLOYEE-MASTER-ID
00393
00394
00395      300-TERMINO.
00396
00397          MOVE WS-NUMBER-EMPLOYEES      TO WS-FOOTING-COUNT
00398          MOVE WS-TOTAL-HOURS        TO WS-FOOTING-TOTAL
00399          MOVE WS-FOOTING-LINE       TO WS-OUTPUT-AREA
00400          MOVE WS-SKIP-BEFORE-FOOTING TO WS-NUMBER-TO-SKIP
00401          MOVE WS-FOOTING-LINE       TO WS-OUTPUT-AREA
00402          PERFORM 260-IMPRIME-REGISTRO
00403
00404          CLOSE    FICHERO-TIEMPO-VALIDO
00405              FICHERO-MAESTRO-EMPLEADOS
00406              FICHERO-CHEQUES-NOMINA
00407              FICHERO-INFORME-NOMINA
00408
```

Fig. 11-6 (cont.)

# Capítulo 12

## Ordenación y fusión de ficheros

La ordenación de ficheros consiste en la reclasificación de los *registros lógicos de un fichero* que reside en un dispositivo de almacenamiento auxiliar. Esto contrasta con la ordenación de tablas o relaciones, en donde partes de un único registro lógico (la tabla) son reorganizados. La clasificación de ficheros es fácilmente programada utilizando el verbo en COBOL “SORT”.

### 12.1 EL VOCABULARIO DE ORDENACION DE FICHEROS

Los campos contenidos en un registro lógico por los que el fichero se ordena o clasifica se denominan *campos de control* o *claves de ordenación*. Los registros de un fichero pueden ordenarse en secuencias *ascendentes* o *descendentes* sobre una clave dada. El COBOL del sistema IBM OS/VS permite la ordenación simultánea de un fichero en hasta 12 diferentes claves, y en cualquier combinación secuencial tanto ascendente como descendente. Cuando la ordenación se realiza en claves múltiples, el campo clave más importante es denominado *clave mayor* y el menos importante se denomina *clave menor*; las restantes *claves intermedias* —incluida la clave menor, también— realizan ordenamientos dentro de cada valor de la clave mayor. Consecuentemente, si centramos la atención únicamente en una clave intermedia o menor, el fichero en su conjunto no aparecerá ordenado.

**EJEMPLO 12.1** La Tabla 12-1 muestra un fichero de ventas ordenado por tres claves. La ordenación mayor es ascendente; la intermedia, dentro de la ordenación mayor, es descendente; la ordenación menor, dentro de la ordenación intermedia, es ascendente.

Tabla 12-1

Departamento # (Clave mayor)	Vendedor # (Clave intermedia)	Cliente # (Clave menor)	Importe ventas (No es una clave)
101	7823	10925	1,000.98
101	7823	27832	2,000.00
101	7823	59838	1,500.00
101	6345	20782	1,200.32
101	6345	48793	5,000.00
101	3287	10925	3,003.43
206	8291	27832	1,305.50
206	8291	40058	2,783.45
206	5094	39840	5,672.34
206	5094	60291	3,476.52
206	4925	27832	4,000.00
206	4925	30618	2,987.50
206	3920	10071	4,526.98

## 12.2 UTILIZACION DE LA INSTRUCCION SORT

La instrucción SORT, cuya sintaxis viene dada en la Figura 12-1, se utiliza en la división de procedimientos. Junto a ella habrá que definir un fichero de ordenación ficticio que deberá tener una instrucción SELECT en la división del entorno (o "SD"), semejante a la descripción del fichero ("FD"), en la división de datos. La descripción de registros para el fichero de ordenación define los campos claves sobre los que se realizará la ordenación. El área del registro lógico del fichero de ordenación se utiliza para pasar registros hacia adelante y hacia atrás entre el programa COBOL y un *programa de utilidades preescrito de ordenación/fusión*, el cual realiza la ordenación propiamente dicha. Los programas de utilidades vienen con el sistema operativo.

```

SORT nombre-fichero-1
  ON { ASCENDING } KEY nombre-dato-1 [nombre-dato-2] ...
  [ ON { ASCENDING } KEY nombre-dato-3 [nombre-dato-4] ... ]
  [COLLATING SEQUENCE IS nombre-alfabeto]
  { USING nombre-fichero-2 [nombre-fichero-3] ...
    { INPUT PROCEDURE IS nombre-sección-1 { { THROUGH } nombre-sección-2 } }
    { GIVING nombre-fichero-4
      { OUTPUT PROCEDURE IS nombre-sección-3 { { THROUGH } nombre-sección-4 } }
    }
  }
}

```

Fig. 12-1

### EJEMPLO 12.2

```

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT FICHERO-PAGOS-HIPOTECAS
    ASSIGN TO MORTRECV
    ORGANIZATION IS SEQUENTIAL
    ACCESS IS SEQUENTIAL

  SELECT FICHERO-ORDEN-PAGOS
    ASSIGN TO SORTWK

DATA DIVISION.
FILE SECTION.
FD FICHERO-PAGOS-HIPOTECAS
  BLOCK CONTAINS 0 CHARACTERS
  RECORD CONTAINS 23 CHARACTERS
  LABEL RECORDS ARE STANDARD

01 REGISTRO-PAGOS-HIPOTECA.
  05 ID-CUENTA-HIPOTECA          PIC X(6).
  05 VENCIMIENTO-HIPOTECA       PIC X(6).
  05 PAGO-HIPOTECA              PIC X(6).
  05 IMPORTE-HIPOTECA           PIC S9(3)V99.

SD FICHERO-ORDEN-PAGOS
  RECORD CONTAINS 23 CHARACTERS

```

01 REGISTRO-ORDEN.	
05 ID-CUENTA-ORDEN	PIC X(6).
05 VENCIMIENTO-ORDEN	PIC X(6).
05 PAGO-ORDEN	PIC X(6).
05 IMPORTE-ORDEN	PIC S9(3)V99.

PROCEDURE DIVISION.

```
SORT FICHERO-ORDEN-PAGOS
  ON ASCENDING KEY ID-CUENTA-ORDEN
  ON ASCENDING KEY PAGO-ORDEN
  INPUT PROCEDURE IS REG-PAGO-PANTALLA
  OUTPUT PROCEDURE IS PRODUCE-INFO-TRIMESTRAL
```

REG-PAGO-PANTALLA SECTION.

PRODUCE-INFO-TRIMESTRAL SECTION.

Aquí FICHERO-PAGOS-HIPOTECAS es el fichero real en disco que tiene que ordenarse en forma ascendente por fecha de pago (clave menor) dentro de cada identificador de cuenta (clave mayor). FICHERO-ORDEN-PAGOS es el fichero de ordenación. Observe:

- (1) Para un fichero de ordenación, la instrucción SELECT tiene la forma simplificada

```
SELECT nombre-fichero-ordenación
ASSIGN TO nombre-asignación
```

Puesto que el fichero de ordenación no es realmente un fichero, el nombre que sigue a ASSIGN TO se procesa como si fuera un comentario. No se necesitan instrucciones adicionales en el lenguaje de control de trabajos para definir el fichero de ordenación.

- (2) Para que la instrucción SORT invoque apropiadamente a la utilidad de fusión/ordenación se necesitan por lo general instrucciones complementarias en el lenguaje de control de trabajos. El lector debe conocer lo que sea necesario en su instalación.
- (3) FICHERO-ORDEN-PAGOS se define en la división de datos por medio de SD:

```
SD nombre-fichero-ordenación
  [RECORD CONTAINS [entero-1 TO] entero-2 CHARACTERS]
    [DATA {RECORD IS | RECORDS ARE} nombre-dato]
```

- (4) REGISTRO-ORDEN (nivel 01) provee la información descriptiva de los registros a ordenar y sirve además como buffer para que los registros vayan hacia adelante o hacia atrás.
- (5) La instrucción SORT contiene el nombre del fichero de ordenación y no del fichero real. La cláusula ASCENDING DESCENDING KEY define los campos clave para la ordenación *desde el mayor al menor*; estos campos deben figurar en la descripción del registro del fichero de ordenación.
- (6) La INPUT PROCEDURE debe ser SECTION. REG-PAGO-PANTALLA contiene un bucle que lee cada registro lógico del FICHERO-PAGOS-HIPOTECAS, lo copia en REGISTRO-ORDEN y lo devuelve al programa de utilidad ejecutando la instrucción RELEASE. Cuando se ejecuta SORT, se lleva a cabo una vez la INPUT PROCEDURE.
- (7) Al finalizar la ejecución de la INPUT PROCEDURE, el programa de utilidad ordena automáticamente los registros lógicos que le han sido proporcionados por medio de RELEASE. Al concluir la ordenación, se ejecuta una vez la OUTPUT PROCEDURE (en este caso PRODUCE-INFO-TRIMESTRAL).
- (8) La OUTPUT PROCEDURE también debe ser una SECTION de COBOL. La utilidad de ordenación tiene que "devolver" a REGISTRO-ORDEN el siguiente registro lógico en orden. Esto se materializa ejecutando una instrucción RETURN dentro de la OUTPUT PROCEDURE. La OUTPUT PROCEDURE puede a continuación procesar el registro (en este caso imprimiendo una línea del informe). Al

igual que la INPUT PROCEDURE, PRODUCE-INFO-TRIMESTRAL contendrá un bucle por el cual todo el fichero retomará del programa de utilidad, un registro cada vez.

- (9) Cuando termina la ejecución de la OUTPUT PROCEDURE, se prosigue con la instrucción inmediatamente siguiente a la instrucción SORT.

Como la INPUT PROCEDURE RELEASE envía los registros lógicos al programa de utilidad de uno en uno, se puede utilizar para *visualizar o preprocesar los registros antes de que sean ordenados*. De hecho, si no es preciso visualizar o preprocesar los registros, se puede eliminar la INPUT PROCEDURE; la utilidad de ordenación puede conseguir los registros directamente del fichero. Esta opción se indica con la cláusula USING de la instrucción SORT.

**EJEMPLO 12.3** Suponga que se desea imprimir etiquetas de correo de un fichero maestro de clientes el cual está ordenado por el número del cliente; deseamos enviar cartas solamente a aquellos clientes que no hayan realizado una compra en los últimos 8 meses. A fin de reducir los gastos de correo, las etiquetas serán ordenadas por el código postal. Como seguramente la mayoría de los clientes habrán estado activos a lo largo de los últimos 8 meses, un INPUT PROCEDURE podría ser utilizado para expulsar (RELEASE) a fines clasificatorios *únicamente* aquellos registros de cliente que interesan.

Además, los registros del fichero de ordenación podrían ser definidos para contener *únicamente* los campos nombre y dirección de los registros maestros de clientes, y la INPUT PROCEDURE podrá escribirse de forma que copie únicamente estos campos en el área de registros del fichero de ordenación antes de expulsar (RELEASING) el registro de ordenación. La ordenación necesitará, pues, menos tiempo de ordenador, ya que son más pequeños los registros a ordenar.

**EJEMPLO 12.4** Suponga que el fichero maestro de clientes incluye las compras totales del pasado año y las del presente año. Se desea imprimir un listado de clientes en secuencia descendente según el cambio porcentual de las compras del año pasado a éste. Hasta aquí, podría describirse un campo de cambio porcentual como una parte de los registros del fichero de ordenación. Podría utilizarse una INPUT PROCEDURE para introducir (INPUT) un registro del maestro de clientes; *calcular* el valor del cambio porcentual y luego copiar este valor, junto con cualquier otra información necesaria del maestro de clientes, al registro de ordenación para ser expulsado (RELEASE) a la utilidad de ordenación.

Hacemos hincapié en que cuando se utiliza la INPUT PROCEDURE los registros a clasificar *no necesariamente han de coincidir con los registros del fichero real* (es decir, no existía ningún campo de cambio porcentual en los registros maestros). Por otro lado, cuando se elige USING..., el SD reflejará exactamente los PICTUREs, USAGEs y estructuras de los registros del fichero de USING.

Así como la INPUT PROCEDURE preprocesa registros no ordenados, la OUTPUT PROCEDURE procesa a posteriori los registros ordenados. En caso de que no se requiera el proceso a posteriori, la opción GIVING puede utilizarse para que la utilidad de ordenación coloque los registros ordenados directamente sobre el fichero GIVING.

**EJEMPLO 12.5** Si el FICHERO-PAGOS-HIPOTECAS del Ejemplo 12.2 fuese utilizado para actualizar el fichero maestro de hipotecas, el programa editor que validase y ordenase los pagos hipotecarios podría contener:

```
SORT FICHERO-ORDEN-PAGOS
  ON ASCENDING KEY ID-CUENTA-ORDEN
  INPUT PROCEDURE IS VALIDA-REG-DATOS
  GIVING FICHERO-TRANS-VALIDAS
```

donde FICHERO-TRANS-VALIDAS es un fichero, con una sentencia SELECT regular y FD.

Debería mencionarse que un programa COBOL podrá contener cualquier número de instrucciones SORT (o MERGE), en el supuesto de que no exista *anidamiento* entre las instrucciones.

Una vez observado el funcionamiento completo de la instrucción SORT, dedicaremos el resto de esta sección a ver de cerca sus cláusulas por separado así como sus diversas opciones.

#### Cláusula ASCENDING/DESCENDING KEY

El objeto de estas cláusulas es identificar las posiciones de las claves dentro de los registros a ordenar. En el caso de registros de longitud variable, se requerirá que cada campo clave se encuentre exactamente en la misma posición en todos los registros.

**EJEMPLO 12.6** Considere el fichero

```

SD FICHERO-ORDEN-ARTICULOS
RECORD CONTAINS 12 TO 22 CHARACTERS.

01 REGISTRO-ARTICULO-FABRICADO.
  05 ID-ARTICULO-FABRICADO      PIC X(6).
  05 EXISTENCIAS-FABRICADO    PIC S9(5) COMP-3.
  05 CODIGO-ALMACEN-FABRICADO PIC X(3).

01 REGISTRO-ARTICULO-COMPRADO.
  05 ID-ARTICULO-COMPRADO      PIC X(6).
  05 EXISTENCIAS-COMPRADO     PIC S9(5) COMP-3.
  05 CODIGO-ALMACEN-COMPRADO PIC X(3).
  05 ID-PROVEEDOR            PIC X(7).
  05 PRECIO-PROVEEDOR        PIC S9(3)V99 COMP-3.

```

La longitud del REGISTRO-ARTICULO-FABRICADO es 12 y la del REGISTRO-ARTICULO-COMPRADO es 22; sin embargo, como se requiere, ID-ARTICULO ocupa la misma posición en ambos registros (bytes 1-6), al igual que CODIGO-ALMACEN (bytes 10-12). Por ello, la instrucción SORT para el fichero podría ser cualquiera de las dos siguientes:

```

SORT FICHERO-ORDEN-ARTICULOS
  ASCENDING KEY ID-ARTICULO-FABRICADO
  DESCENDING KEY CODIGO-ALMACEN-FABRICADO
  INPUT PROCEDURE IS VISUALIZA-ARTICULOS
  OUTPUT PROCEDURE IS IMPRIME-INFORME

```

o

```

SORT FICHERO-ORDEN-ARTICULOS
  ASCENDING KEY ID-ARTICULO-COMPRADO
  DESCENDING KEY CODIGO-ALMACEN-COMPRADO
  INPUT PROCEDURE IS VISUALIZA-ARTICULOS
  OUTPUT PROCEDURE IS IMPRIME-INFORME

```

**Cláusula COLLATING SEQUENCE**

Esta cláusula es análoga a la cláusula PROGRAM COLLATING SEQUENCE que se estudió en la Sección 4.3; sin embargo, el orden de los caracteres especificado en una instrucción SORT no tiene porque coincidir con el especificado para la comparación de elementos alfanuméricos en la división de procedimientos. "Nombre-alfabeto" (Fig. 12-1) debe definirse en el párrafo SPECIAL-NAMES (Sección 4.4). Generalmente en la instrucción SORT se omite la cláusula COLLATING SEQUENCE, dejando que funcione el orden por defecto.

**INPUT PROCEDURE: RELEASE...****OUTPUT PROCEDURE: RETURN...**

Cuando se utiliza una INPUT PROCEDURE, ésta debe contener al menos una instrucción RELEASE:

RELEASE nombre-registro [FROM nombre-dato]

La instrucción RELEASE es similar a WRITE; de hecho, puede imaginarse que su función es la escritura de un registro del fichero de ordenación en el programa de ordenación, quien toma estos registros para ordenarlos al acabar la ejecución de la INPUT PROCEDURE. La función de la opción FROM es similar a la de WRITE...FROM...

**EJEMPLO 12.7** En el Ejemplo 12.6, la rutina VISUALIZA-ARTICULOS podría contener:

```

IF ARTICULO-FABRICADO
  RELEASE REGISTRO-ARTICULO-FABRICADO
  FROM REGISTRO-ENTRADA

```

ELSE

```
MOVE ID-ENTRADA TO ID-ARTICULO-COMPRADO
MOVE CANTIDAD-ENTRADA TO EXISTENCIAS-COMPRADO
MOVE CODIGO-ENTRADA TO CODIGO-ALMACEN-COMPRADO
MOVE PROVEEDOR-ENTRADA TO ID-PROVEEDOR
MOVE PRECIO-ENTRADA TO PRECIO-PROVEEDOR
RELEASE REGISTRO-ARTICULO-COMPRADO
```

Cada OUTPUT PROCEDURE debe contener al menos una instrucción RETURN, que indique a la utilidad de ordenación que ponga el siguiente registro lógico ordenado en el área del registro del fichero de ordenación:

RETURN nombre-fichero-clasificación RECORD [INTO nombre-dato]  
AT END instrucción imperativa

La instrucción RETURN es similar a READ y RETURN... INTO... funciona igual que READ... INTO... Más aún, la cláusula AT END funciona idénticamente a la cláusula de idéntico nombre en la instrucción READ.

#### SORT...USING...GIVING...

Cuando no se precisan procesar los registros lógicos ni antes ni después, se puede emplear SORT...USING...GIVING. Sin embargo, es más eficiente *ejecutar el programa de utilidad por separado* (sin invocarlo desde COBOL). En un SORT de COBOL, los ficheros de USING y GIVING tienen que estar cerrados cuando se ejecuta la instrucción SORT.

#### SORT...INPUT PROCEDURE...GIVING...

Cuando se utiliza una INPUT (OUTPUT) PROCEDURE, la división de procedimientos debe estructurarse como en el Ejemplo 12.8.

#### EJEMPLO 12.8

```
PROCEDURE DIVISION.
  SORT...
    ON...
    INPUT PROCEDURE IS VISUALIZA-ENTRADA
    OUTPUT PROCEDURE IS IMPRIME-INFOMRE
  STOP RUN
```

VISUALIZA-ENTRADA SECTION.  
.....

IMPRIME-INFOMRE SECTION.  
.....

PROCED-FIN-CLASIF SECTION.  
PARRAFO-A.  
.....

PARRAFO-B.  
.....

Observe que el verbo SORT se coloca en el módulo ejecutivo, inmediatamente a continuación de STOP RUN. Una instrucción STOP RUN durante la INPUT OUTPUT PROCEDURE produciría resultados imprevisibles; en el caso del COBOL del sistema IBM OS/VS se abortaría el programa.

Observe también la "PROCED-FIN-CLASIF SECTION"; sin esta cabecera, los PARRAFO-A y PARRAFO-B pertenecerían a la IMPRIME-INFORME SECTION. Si el programa fuera tal que bien VISUALIZA-ENTRADA o bien IMPRIME-INFORME ejecutan cualquiera de estos párrafos (legal en el COBOL del sistema IBM OS/VS), sería mejor una cabecera "PARRAFOS-EJECUTADOS SECTION". Véase el Problema 12.32 para más información sobre el uso de PERFORM en las INPUT OUTPUT PROCEDURE.

Una aplicación típica de SORT...INPUT PROCEDURE...GIVING... sería un programa de edición.

**EJEMPLO 12.9** Vuelva a hacer el programa de edición de los Problemas 11.22 y 8.30 incluyendo una instrucción SORT que utilice una INPUT PROCEDURE para visualizar los registros inválidos y emplee la opción GIVING para colocar los registros válidos (transacciones) en un fichero en disco.

Véase la Figura 12-2, prestando especial atención a la estructura de la división de procedimientos. Toda la anterior división de procedimientos (de la que sólo una parte se da en la Figura 11-5) pasa a ser la INPUT PROCEDURE de SORT. La instrucción WRITE original (no incluida en la Figura 11-5, pero sí en la línea 209 de la Figura 8-14) que transfiere una transacción validada al disco se ha reemplazado por una instrucción RELEASE que entrega la transacción validada a la utilidad de ordenación. La utilidad de ordenación creará el FICHERO-TIEMPO-VALIDO como respuesta a la opción GIVING. Observe que el fichero de GIVING *debe estar cerrado* al ejecutar SORT, puesto que esta instrucción abre automáticamente los ficheros de GIVING y de USING.

```

00014      INPUT-OUTPUT SECTION.
00015      FILE-CONTROL.
00016
00017          SELECT FICHERO-TIEMPO-TARJETAS
00018              ASSIGN TO TIMECARD
00019                  ORGANIZATION IS SEQUENTIAL
00020                  ACCESS IS SEQUENTIAL
00021
00022      *
00023      *
00024      *
00025      * PARA REALIZAR UNA CLASIFICACION, SE DEBE DEFINIR
00026      * CON UNA INSTRUCCION SELECT EL "FICHERO DE ORDENACION",
00027      * QUE NO ES PROPIAMENTE UN FICHERO, PERO QUE
00028      * REPRESENTA EL AREA EN LA QUE LA UTILIDAD SORT/MERGE
00029      * MANIPULA LOS REGISTROS A CLASIFICAR.
00030      *
00031      *
00032      *
00033          SELECT FICHERO-TARJETAS-CLASIFICADO
00034              ASSIGN TO SORTWK
00035
00036          SELECT FICHERO-TIEMPO-VALIDO
00037              ASSIGN TO DISKTIME
00038                  ORGANIZATION IS SEQUENTIAL
00039                  ACCESS IS SEQUENTIAL
00040
00041          SELECT FICHERO-LISTADO-ERRORES
00042              ASSIGN TO ERRORLOG
00043                  ORGANIZATION IS SEQUENTIAL
00044                  ACCESS IS SEQUENTIAL
00045
00046
00047      DATA DIVISION.
00048
00049      FILE SECTION.
00050
00051          FD  FICHERO-TIEMPO-TARJETAS
00052              RECORD CONTAINS 80 CHARACTERS
00053                  LABEL RECORDS ARE OMITTED
00054
00055

```

Fig. 12-2

```

00056      01 TIME-CARD-RECORD.
00057          05 TIME-CARD-ID           PIC X(5).
00058          05 TIME-CARD-HOURS      PIC 9(2)V9.
00059          05 TIME-CARD-X-HOURS    REDEFINES TIME-CARD-HOURS
00060                  PIC X(3).
00061          05 TIME-CARD-CLOSING-DATE.
00062              10 TIME-CARD-CLOSING-MO      PIC 99.
00063              10 TIME-CARD-CLOSING-DAY     PIC 99.
00064              10 TIME-CARD-CLOSING-YR      PIC 99.
00065          05 TIME-CARD-DEPARTMENT   PIC X(4).
00066          05 FILLER                 PIC X(62).
00067
00068      *
00069      *
00070      * EL "FICHERO DE ORDENACION" DEBE DEFINIRSE CON "SD"
00071      * EN VEZ DE CON "FD". "SD" DEFINE LA COMPOSICION
00072      * DEL REGISTRO A CLASIFICAR. REPRESENTA EL AREA
00073      * DE MEMORIA DE LA QUE LA UTILIDAD SORT/MERGE
00074      * TOMA Y DESPUES CEDE LOS REGISTROS (UNO CADA
00075      * VEZ) DESPUES DE PROCESARLOS.
00076      *
00077      *
00078      *
00079          SD FICHERO-TARJETAS-CLASIFICADO
00080          RECORD CONTAINS 23 CHARACTERS
00081
00082
00083          01 VALID-TIME-RECORD.
00084              05 VALID-TIME-ID          PIC X(5).
00085              05 VALID-TIME-HOURS      PIC S9(2)V9  COMP-3.
00086              05 VALID-TIME-CLOSING-DATE PIC X(6).
00087              05 VALID-TIME-DEPARTMENT  PIC X(4).
00088              05 VALID-TIME-EDIT-DATE   PIC X(6).
00089
00090          FD FICHERO-TIEMPO-VALIDO
00091          BLOCK CONTAINS 0 RECORDS
00092          RECORD CONTAINS 23 CHARACTERS
00093          LABEL RECORDS ARE STANDARD
00094
00095
00096          01 SORTED-VALID-RECORD      PIC X(23).
00097
00098
00099          FD FICHERO-LISTADO-ERRORES
00100          RECORD CONTAINS 132 CHARACTERS
00101          LABEL RECORDS ARE OMITTED
00102
00103
00104          01 ERROR-REPORT-LINE        PIC X(132).
00105
00106          WORKING-STORAGE SECTION.

00215          PROCEDURE DIVISION.
00216
00217      *
00218      *
00219      * CON EL VERBO "SORT" SE DEBE NOMBRAR AL FICHERO DE
00220      * ORDENACION. TAMBIEN SE INDICA LA(S) CLAVE(S) POR
00221      * LAS QUE SE VA A CLASIFICAR (ESTAS CLAVES DEBEN
00222      * FORMAR PARTE DE LA DESCRIPCION DEL REGISTRO),
00223      * Y SI VA A SER ASCENDENTE O DESCENDENTE. EN
00224      * ESTE CASO EL PROGRAMA DE EDICION ES UNA "INPUT
00225      * PROCEDURE" PARA "SORT". LOS REGISTROS CLASIFICADOS
00226      * SE COLOCAN EN EL FICHERO DE "GIVING".
00227      *
00228      *
00229      *

```

Fig. 12-2 (cont.)

```

00230      SORT FICHERO-TARJETAS-CLASIFICADO
00231          ON ASCENDING KEY VALID-TIME-ID
00232          INPUT PROCEDURE IS 000-EDITA-TARJETAS
00233              GIVING VALID-TIME-FILE
00234          STOP RUN
00235
00236
00237      000-EDITA-TARJETAS SECTION.
00238
00239          PERFORM 100-INICIALIZACION
00240          PERFORM 200-PRODUCE-LISTADO
00241              UNTIL NO-MORE-RECORDS
00242          PERFORM 300-TERMINO
00243
00244
00245      PARRAFOS-A-EJECUTAR SECTION.
00246
00247      100-INICIALIZACION.
00248
00249          OPEN      INPUT      FICHERO-TIEMPO-TARJETAS
00250                  OUTPUT     FICHERO-LISTADO-ERRORES
00251          MOVE "NO" TO           WS-END-TIME-FILE-SW
00252          MOVE ZERO TO          WS-PAGE-NUMBER
00253
00254          MOVE WS-PAGE-SIZE TO    WS-NUMBER-EMPLOYEES
00255          ACCEPT WS-SYSTEM-DATE FROM DATE
00256          MOVE SYSTEM-YY TO      WS-TOTAL-HOURS
00257          MOVE SYSTEM-MM TO      WS-LINES-ON-PAGE
00258          MOVE SYSTEM-DD TO      WS-HEADING-YY
00259
00260
00261      200-PRODUCE-LISTADO.
00262
00263          PERFORM 250-CONSIGUE-TARJETA-TIEMPO
00264          IF RECORD-AVAILABLE
00265              PERFORM 240-VALIDA-CAMPOS
00266              IF VALID-RECORD
00267                  ADD 1           TO WS-NUMBER-EMPLOYEES
00268                  ADD TIME-CARD-HOURS   TO WS-TOTAL-HOURS
00269                  PERFORM 210-LIBERA-PARA-CLASIFICAR
00270
00271          ELSE
00272              PERFORM 220-PRODUCE-LISTADO-ERRORES
00273
00274
00275      210-LIBERA-PARA-CLASIFICAR.
00276
00277      *
00278      *
00279      * LIBERA UN REGISTRO PARA QUE LA UTILIDAD SORT/MERGE
00280      * LO CLASIFIQUE. PARA ELLO SE COPIA EL REGISTRO
00281      * EN EL AREA DEL BUFFER DEL FICHERO DE ORDENACION.
00282      *
00283      *
00284
00285          MOVE TIME-CARD-ID          TO VALID-TIME-ID
00286          MOVE TIME-CARD-HOURS       TO VALID-TIME-HOURS
00287          MOVE TIME-CARD-CLOSING-DATE TO VALID-TIME-CLOSING-DATE
00288          MOVE TIME-CARD-DEPARTMENT   TO VALID-TIME-DEPARTMENT
00289          MOVE WS-SYSTEM-DATE        TO VALID-TIME-EDIT-DATE
00290          RELEASE VALID-TIME-RECORD
00291
00292
00293      220-PRODUCE-LISTADO-ERRORES.
00294
00295          MOVE TIME-CARD-ID          TO WS-DETAIL-ID

```

Fig. 12-2 (cont.)

```

00296      MOVE TIME-CARD-X-HOURS          TO WS-DETAIL-HOURS
00297      MOVE TIME-CARD-CLOSING-DATE    TO WS-DETAIL-DATE
00298      MOVE TIME-CARD-DEPARTMENT      TO WS-DETAIL-DEPARTMENT
00299
00300      MOVE WS-DETAIL-LINE           TO WS-OUTPUT-AREA
00301      MOVE WS-SKIP-BEFORE-DETAIL    TO WS-NUMBER-TO-SKIP
00302      PERFORM 230-IMPRIME-AREA-SALIDA
00303
00304      MOVE WS-UNDER-LINE           TO WS-OUTPUT-AREA
00305      MOVE WS-SKIP-BEFORE-UNDER     TO WS-NUMBER-TO-SKIP
00306      PERFORM 230-IMPRIME-AREA-SALIDA
00307
00308
00309      230-IMPRIME-AREA-SALIDA.
00310
00311      IF WS-LINES-ON-PAGE + WS-NUMBER-TO-SKIP
00312          GREATER THAN WS-PAGE-SIZE
00313              ADD 1 TO WS-PAGE-NUMBER
00314              MOVE WS-PAGE-NUMBER TO WS-HEADING-PAGE
00315              WRITE ERROR-REPORT-LINE
00316                  FROM WS-HEADING-LINE-1
00317                  AFTER ADVANCING PAGE
00318              WRITE ERROR-REPORT-LINE
00319                  FROM WS-HEADING-LINE-2
00320                  AFTER ADVANCING WS-SKIP-BEFORE-HEADING LINES
00321              MOVE WS-SKIP-BEFORE-HEADING TO WS-LINES-ON-PAGE
00322
00323              WRITE ERROR-REPORT-LINE
00324                  FROM WS-OUTPUT-AREA
00325                  AFTER ADVANCING WS-NUMBER-TO-SKIP LINES
00326              ADD WS-NUMBER-TO-SKIP TO WS-LINES-ON-PAGE
00327
00328
00329      240-VALIDA-CAMPOS.
00330
00331      MOVE "NO"   TO WS-ERROR-DETECTED-SW;
00332      MOVE SPACES TO WS-UNDER-LINE
00333      IF TIME-CARD-ID NOT NUMERIC
00334          MOVE ALL "-" TO WS-UNDER-ID
00335          MOVE "YES"  TO WS-ERROR-DETECTED-SW
00336
00337      IF TIME-CARD-HOURS NOT NUMERIC
00338          MOVE ALL "-" TO WS-UNDER-HOURS
00339          MOVE "YES"  TO WS-ERROR-DETECTED-SW
00340
00341      IF TIME-CARD-CLOSING-DATE NOT NUMERIC
00342          MOVE ALL "-" TO WS-UNDER-DATE
00343          MOVE "YES"  TO WS-ERROR-DETECTED-SW
00344      ELSE
00345          PERFORM 260-VALIDA-FECHA
00346
00347      IF TIME-CARD-DEPARTMENT NOT NUMERIC
00348          MOVE ALL "-" TO WS-UNDER-DEPARTMENT
00349          MOVE "YES"  TO WS-ERROR DETECTED-SW
00350
00351
00352      250-CONSIGUE-TARJETA-TIEMPO.
00353
00354      READ FICHERO-TIEMPO-TARJETAS
00355          AT END
00356              MOVE "YES" TO WS-END-TIME-FILE-SW
00357
00358
00359      260-VALIDA-FECHA.
00360

```

Fig. 12-2 (cont.)

```

00361      IF TIME-CARD-CLOSING-YR GREATER THAN SYSTEM-YY
00362          MOVE ALL "-" TO WS-UNDER-YR
00363          MOVE "YES" TO WS-ERROR-DETECTED-SW
00364
00365          IF TIME-CARD-CLOSING-MO LESS THAN 1 OR GREATER THAN 12
00366              MOVE ALL "-" TO WS-UNDER-MO
00367              MOVE "YES" TO WS-ERROR-DETECTED-SW
00368          ELSE IF TIME-CARD-CLOSING-DAY LESS THAN 1 OR
00369              GREATER THAN DAYS-PER-MONTH (TIME-CARD-CLOSING-MO)
00370              MOVE ALL "-" TO WS-UNDER-DAY
00371              MOVE "YES" TO WS-ERROR-DETECTED-SW
00372
00373
00374      300-TERMINO.
00375
00376          MOVE WS-NUMBER-EMPLOYEES      TO WS-FOOTING-COUNT
00377          MOVE WS-TOTAL-HOURS        TO WS-FOOTING-TOTAL
00378          MOVE WS-FOOTING-LINE       TO WS-OUTPUT-AREA
00379          MOVE WS-SKIP-BEFORE-FOOTING TO WS-NUMBER-TO-SKIP
00380          MOVE WS-FOOTING-LINE       TO WS-OUTPUT-AREA
00381          PERFORM 230-IMPRIME-AREA-SALIDA
00382
00383          CLOSE   FICHERO-TIEMPO-TARJETAS
00384          FICHERO-LISTADO-ERRORES
00385

```

Fig. 12-2 (cont.)

**SORT...USING...OUTPUT PROCEDURE...**

Esta versión puede utilizarse para ordenar y después procesar un *fichero completo*, si no es preciso preprocessar los registros lógicos (por ejemplo, en el caso de las etiquetas de correos).

**EJEMPLO 12.10** El Problema 8.32 relativo a la impresión de cheques de una aplicación de nóminas cuyo programa de edición se trató en el Problema 11.22 y el Ejemplo 12.9. Revise el Problema 8.32 para que imprima los cheques por orden alfabetico del nombre del empleado (en lugar de por el número de empleado que es el orden en el FICHERO-DISCO-NOMINAS).

Véase la Figura 12-3; los comentarios de las líneas 156-160 se refieren al “programa original” al que corresponde el organigrama estructurado de la Figura 8-17 pero que no se mostró en el Problema 8.32.

```

00022      SELECT FICHERO-DISCO-NOMINA
00023          ASSIGN TO PAYDISK
00024          ORGANIZATION IS SEQUENTIAL
00025          ACCESS IS SEQUENTIAL
00026
00027      SELECT FICHERO-ORD-CHEQUES
00028          ASSIGN TO SORTUTIL
00029
00030      SELECT FICHERO-CHEQUES
00031          ASSIGN TO CHECKS
00032          ORGANIZATION IS SEQUENTIAL
00033          ACCESS IS SEQUENTIAL
00034
00035
00036      DATA DIVISION.
00037
00038      FILE SECTION.
00039
00040      FD  FICHERO-DISCO-NOMINA
00041          BLOCK CONTAINS 0 RECORDS

```

Fig. 12-3

```

00042      RECORD CONTAINS 80 CHARACTERS
00043      LABEL RECORDS ARE STANDARD
00044
00045
00046      *
00047      * OBSERVE QUE PAYROLL-CHECK-RECORD NO NECESITA
00048      * SER TOTALMENTE DESCRITO CON SORT...USING...
00049      *
00050
00051      01 PAYROLL-CHECK-RECORD      PIC X(80).
00052
00053      SD FICHERO-ORD-CHEQUES
00054      RECORD CONTAINS 80 CHARACTERS
00055
00056
00057      01 SORT-CHECK-RECORD.
00058          05 PAYROLL-ID      PIC X(5).
00059          05 PAYROLL-HOURS    PIC S9(2)V9.
00060          05 PAYROLL-DATE     PIC X(6).
00061          05 PAYROLL-DEPARTMENT PIC X(4).
00062          05 PAYROLL-RATE     PIC S9(3)V99.
00063          05 PAYROLL-NAME     PIC X(4).
00064              10 PAYROLL-INITIALS PIC X(4).
00065              10 PAYROLL-LAST-NAME PIC X(16).
00066          05 PAYROLL-GROSS    PIC S9(6)V99.
00067          05 PAYROLL-TAX     PIC S9(5)V99.
00068          05 PAYROLL-NET     PIC S9(6)V99.
00069          05 FILLER        PIC X(14).
00070
00071      FD FICHERO-CHEQUES
00072      RECORD CONTAINS 132 CHARACTERS
00073      LABEL RECORDS ARE OMITTED
00074
00075
00076      01 PAYCHECK-LINE      PIC X(132).
00077
00078      WORKING-STORAGE SECTION.

```

```

00152      PROCEDURE DIVISION.
00153
00154
00155      *
00156      * LA INSTRUCCION SORT SE COLOCA EN EL MODULO DE
00157      * NIVEL MAS ALTO. EL RESTO DEL PROGRAMA ES LA
00158      * RUTINA DE SALIDA CON LOS SIGUIENTES CAMBIOS
00159      * UNICAMENTE: 1) NO DEBE ABRIRSE FICHERO-DISCO-
00160      * NOMINA Y 2) "READ FICHERO-DISCO-NOMINA" SE
00161      * CONVIERTA EN "RETURN FICHERO-ORD-CHEQUES".
00162
00163      SORT FICHERO-ORD-CHEQUES
00164          ON ASCENDING KEY PAYROLL-LAST-NAME
00165          USING PAYROLL-DISK-FILE
00166          OUTPUT PROCEDURE IS IMPRIME-CHEQUES
00167          STOP RUN
00168
00169
00170      IMPRIME-CHEQUES SECTION.
00171
00172          PERFORM 100-INICIALIZACION
00173          PERFORM 200-PRODUCE-CHEQUES
00174              UNTIL NO-MORE-RECORDS
00175          PERFORM 300-TERMINO
00176

```

Fig. 12-3 (cont.)

```

00177      *-----  

00178          PARRAFOS-A-EJECUTAR SECTION.  

00179      *-----  

00180          100-INICIALIZACION.  

00181  

00182          OPEN    OUTPUT  FICHERO-CHEQUES  

00183          MOVE "NO" TO           WS-END-DISK-FILE-SW  

00184          MOVE ZERO TO        WS-NUMBER-EMPLOYEES  

00185                      WS-TOTAL-HOURS  

00186          ACCEPT WS-TODAYS-DATE FROM DATE  

00187          MOVE SYSTEM-YY        TO WS-YY  

00188          MOVE SYSTEM-MM        TO WS-MM  

00189          MOVE SYSTEM-DD        TO WS-DD  

00190          ACCEPT WS-CHECK-NUMBER  

00191              FROM STARTING-CHECK-NUMBER-DEVICE  

00192  

00193          PERFORM 110-PRODUCE-ASTERISCOS  

00194          PERFORM 210-CONSIGUE-SIGUIENTE-EMP  

00195  

00196  

00197          110-PRODUCE-ASTERISCOS.  

00198  

00199          MOVE ALL "*" TO WS-STUB-ID  

00200                      WS-STUB-NAME  

00201                      WS-STUB-DATE  

00202                      WS-CHECK-NAME  

00203          PERFORM 230-IMPRIME-CHEQUE  

00204                      WS-NUMBER-OF-TEMPLATES TIMES  

00205  

00206  

00207          200-PRODUCE-CHEQUES.  

00208  

00209          PERFORM 220-FORMATEA-CHEQUE  

00210          PERFORM 230-IMPRIME-CHEQUE  

00211          ADD PAYROLL-HOURS           TO WS-TOTAL-HOURS  

00212          ADD 1                     TO WS-NUMBER-EMPLOYEES  

00213                      WS-CHECK-NUMBER  

00214          PERFORM 210-CONSIGUE-SIGUIENTE-EMP  

00215  

00216  

00217          210-CONSIGUE-SIGUIENTE-EMP.  

00218  

00219          RETURN FICHERO-ORD-CHEQUES  

00220          AT END  

00221          MOVE "YES" TO WS-END-DISK-FILE-SW  

00222  

00223  

00224          220-FORMATEA-CHEQUE.  

00225  

00226          MOVE PAYROLL-ID           TO WS-STUB-ID  

00227          MOVE PAYROLL-NAME         TO WS-STUB-NAME  

00228                      WS-CHECK-NAME  

00229          MOVE PAYROLL-DEPARTMENT     TO WS-STUB-DEPARTMENT  

00230          MOVE PAYROLL-DATE          TO WS-STUB-DATE  

00231          MOVE PAYROLL-HOURS         TO WS-STUB-HOURS  

00232          MOVE PAYROLL-RATE          TO WS-STUB-RATE  

00233          MOVE PAYROLL-GROSS         TO WS-STUB-GROSS  

00234          MOVE PAYROLL-TAX          TO WS-STUB-TAX  

00235          MOVE PAYROLL-NET          TO WS-STUB-NET  

00236                      WS-CHECK-AMOUNT  

00237  

00238  

00239          230-IMPRIME-CHEQUE.  

00240  

00241          WRITE PAYCHECK-LINE  

00242              FROM WS-STUB-LINE-1

```

Fig. 12-3 (cont.)

```

00243          AFTER ADVANCING WS-SKIP-BEFORE-STUB-1 LINES
00244      WRITE PAYCHECK-LINE
00245          FROM WS-STUB-LINE-2
00246          AFTER ADVANCING WS-SKIP-BEFORE-STUB-2 LINES
00247      WRITE PAYCHECK-LINE
00248          FROM WS-CHECK-LINE-1
00249          AFTER ADVANCING WS-SKIP-BEFORE-CHECK-1 LINES
00250      WRITE PAYCHECK-LINE
00251          FROM WS-CHECK-LINE-2
00252          AFTER ADVANCING WS-SKIP-BEFORE-CHECK-2 LINES
00253
00254
00255      300-TERMINO.
00256
00257      CLOSE    FICHERO-CHEQUES
00258

```

Fig. 12-3 (cont.)

**SORT...INPUT PROCEDURE...OUTPUT PROCEDURE...**

**EJEMPLO 12.11** En la Figura 12-4 se ilustran las INPUT OUTPUT PROCEDURES con un programa que crea un fichero indexado; dichos ficheros son muy populares para las aplicaciones en las que se requiere procesamiento al azar. El modo más eficiente de crear ficheros indexados es construirlos secuencialmente, colocando los registros en el fichero ordenados por una clave. Se utiliza una INPUT PROCEDURE para detectar los registros inválidos y evitar que se ordenen; después una OUTPUT PROCEDURE crea realmente el fichero indexado. Observe que INPUT PROCEDURE y OUTPUT PROCEDURE parecen programas separados: comienzan abriendo ficheros e inicializando interruptores, después llevan a cabo un párrafo con el procesamiento principal hasta el final de los datos y por último cierran los ficheros. Esto no constituye una sorpresa, porque las dos "rutinas" ejecutan independientemente bajo el control del verbo SORT.

```

00015      INPUT-OUTPUT SECTION.
00016
00017      FILE-CONTROL.
00018
00019          SELECT FICHERO-INFO-CREACION
00020              ASSIGN TO CREATE
00021              ORGANIZATION IS SEQUENTIAL
00022              ACCESS IS SEQUENTIAL
00023
00024          SELECT FICHERO-ORD-TRANSACCIONES
00025              ASSIGN TO SORTWK
00026
00027          SELECT FICHERO-MAESTRO-PRECIOS
00028              ASSIGN TO PRICES
00029              ORGANIZATION IS INDEXED
00030              ACCESS IS SEQUENTIAL
00031              RECORD KEY IS PRICING-ITEM-ID
00032              FILE STATUS IS MASTER-STATUS-CODE
00033
00034          SELECT ERRORES-LOGICOS
00035              ASSIGN TO ERRORS
00036              ORGANIZATION IS SEQUENTIAL
00037              ACCESS IS SEQUENTIAL
00038
00039
00040      DATA DIVISION.
00041
00042      FILE SECTION.
00043
00044          FD  FICHERO-INFO-CREACION
00045              RECORD CONTAINS 80 CHARACTERS

```

Fig. 12-4

```

00046      LABEL RECORDS ARE OMITTED
00047
00048
00049      01 CREATION-RECORD.
00050          05 CREATION-ITEM-ID      PIC X(5).
00051          05 CREATION-ITEM-DESCRIPTION
00052              PIC X(25).
00053          05 CREATION-PRICE      PIC 9(5)V99.
00054          05 CREATION-PRICE-X     REDEFINES CREATION-PRICE
00055              PIC X(7).
00056          05 CREATION-QUANTITY    PIC 9(5).
00057          05 CREATION-QUANTITY-X   REDEFINES CREATION-QUANTITY
00058              PIC X(5).
00059          05 FILLER           PIC X(38).

00060      SD FICHERO-ORD-TRANSACCIONES
00061          RECORD CONTAINS 80 CHARACTERS
00062
00063      01 TRANS-RECORD.
00064          05 TRANSACT-ITEM-ID     PIC X(5).
00065          05 TRANSACT-ITEM-DESCRIPTION
00066              PIC X(25).
00067          05 TRANSACT-PRICE      PIC S9(5)V99.
00068          05 TRANSACT-PRICE-X     REDEFINES TRANSACT-PRICE
00069              PIC X(7).
00070          05 TRANSACT-QUANTITY    PIC S9(5).
00071          05 TRANSACT-QUANTITY-X   REDEFINES TRANSACT-QUANTITY
00072              PIC X(5).
00073          05 FILLER           PIC X(38).

00074
00075      FD FICHERO-MAESTRO-PRECIOS
00076          RECORD CONTAINS 37 CHARACTERS
00077          LABEL RECORDS ARE STANDARD
00078
00079      01 PRICING-RECORD.
00080          05 PRICING-ITEM-ID      PIC X(5).
00081          05 PRICING-ITEM-DESCRIPTION
00082              PIC X(25).
00083          05 PRICING-QUANTITY-ONHAND PIC S9(5)      COMP-3.
00084          05 PRICING-PRICE        PIC S9(5)V99  COMP-3.
00085
00086      FD ERRORES-LOGICOS
00087          RECORD CONTAINS 132 CHARACTERS
00088          LABEL RECORDS ARE OMITTED
00089          LINAGE IS 60 LINES
00090              WITH FOOTING AT 59
00091              LINES AT TOP 3
00092              LINES AT BOTTOM 3
00093
00094
00095      01 ERROR-LINE           PIC X(132).
00096
00097      WORKING-STORAGE SECTION.
00098
00099      01 PROGRAM-SWITCHES.
00100          05 WS-END-CREATION-SW    PIC X(3).
00101              88 NO-MORE-RECORDS   VALUE "YES".
00102              88 MORE-RECORDS     VALUE "NO".
00103          05 WS-VALID-TRANS-SW   PIC X(3).
00104              88 VALID-TRANS     VALUE "YES".
00105              88 INVALID-TRANS   VALUE "NO".
00106          05 MASTER-STATUS-CODE  PIC XX.

00107
00108      01 PROGRAM-COUNTERS.
00109          05 WS-PAGE-NUMBER      PIC S9(3)      COMP-3.
00110
00111      01 SYSTEM-DATE-AREA.

```

Fig. 12-4 (cont.)

```

00112      05 SYSTEM-YY          PIC 99.
00113      05 SYSTEM-MM          PIC 99.
00114      05 SYSTEM-DD          PIC 99.
00115
00116      01 PROGRAM-CONSTANTS.
00117      05 WS-MESSAGES.
00118          10 WS-INVALID-FIELDS-MSG      PIC X(30)
00119                      VALUE "INVALID FIELDS".
00120          10 WS-FILE-FULL-MSG      PIC X(30)
00121                      VALUE "FILE FULL--NO ADD".
00122          10 WS-DUPLICATE-MESSAGE      PIC X(30)
00123                      VALUE "DUPLICATE KEY--NO ADD".
00124
00125
00126      01 WS-HEADING-LINE-1.
00127          05 FILLER            PIC X(3)      VALUE SPACES.
00128          05 FILLER            PIC X(32)
00129                      VALUE "PRICING FILE CREATION ERROR LOG".
00130          05 WS-MM              PIC Z9.
00131          05 FILLER            PIC X          VALUE "/".
00132          05 WS-DD              PIC 99.
00133          05 FILLER            PIC X          VALUE "/".
00134          05 WS-YY              PIC 99.
00135          05 FILLER            PIC X(2)      VALUE SPACES.
00136          05 FILLER            PIC X(5)      VALUE "PAGE ".
00137          05 WS-HEADING-PAGE      PIC ZZ9.
00138          05 FILLER            PIC X(79)     VALUE SPACES.
00139
00140      01 WS-HEADING-LINE-2.
00141          05 FILLER            PIC X(10)     VALUE "ITEM-ID".
00142          05 FILLER            PIC X(27)
00143                      VALUE "DESCRIPTION".
00144          05 FILLER            PIC X(9)      VALUE "PRICE".
00145          05 FILLER            PIC X(86)     VALUE "QUANTITY".
00146
00147      01 WS-DETAIL-LINE.
00148          05 WS-DETAIL-ITEM-ID      PIC X(5).
00149          05 FILLER            PIC X(5)      VALUE SPACES.
00150          05 WS-DETAIL-DESCRIPTION      PIC X(25).
00151          05 FILLER            PIC X(2)      VALUE SPACES.
00152          05 WS-DETAIL-PRICE      PIC X(7).
00153          05 FILLER            PIC X(2)      VALUE SPACES.
00154          05 WS-DETAIL-QUANTITY      PIC X(5).
00155          05 FILLER            PIC X(2)      VALUE SPACES.
00156          05 WS-DETAIL-MESSAGE      PIC X(79).
00157
00158      PROCEDURE DIVISION.
00159
00160          SORT FICHERO-ORD-TRANSACCIONES
00161              ON ASCENDING KEY TRANSACT-ITEM-ID
00162              INPUT PROCEDURE IS 000-SCREEN-TRANSACTIONS
00163              OUTPUT PROCEDURE IS 001-CREATE-PRICING-MASTER
00164              STOP RUN
00165
00166
00167      000-TRANSAACCIONES-PANTALLA SECTION.
00168
00169          OPEN OUTPUT ERRORES-LOGICOS
00170          OPEN INPUT FICHERO-INFO-CREACION
00171          MOVE "NO" TO WS-END-CREATION-SW
00172
00173          ACCEPT SYSTEM-DATE-AREA FROM DATE
00174          MOVE SYSTEM-MM TO WS-MM
00175          MOVE SYSTEM-DD TO WS-DD
00176          MOVE SYSTEM-YY TO WS-YY
00177

```

Fig. 12-4 (cont.)

```

00178      MOVE ZERO      TO WS-PAGE-NUMBER
00179
00180      PERFORM 170-IMPRIME-CABECERAS
00181      PERFORM 180-CONSIGUE-SIGUIENTE-TRAN
00182
00183      PERFORM 100-GENERA-REG-PRECIO
00184          UNTIL NO-MORE-RECORDS
00185
00186      CLOSE   FICHERO-INFO-CREACION
00187
00188
00189      001-CREA-MAESTRO-PRECIOS SECTION.
00190
00191      OPEN OUTPUT FICHERO-MAESTRO-PRECIOS
00192      MOVE "NO" TO WS-END-CREATION-SW
00193      PERFORM 140-DEVUELVE-SIGUIENTE-TRAN
00194      PERFORM 120-PRODUCE-REG-MAESTRO
00195          UNTIL NO-MORE-RECORDS
00196      CLOSE   FICHERO-MAESTRO-PRECIOS
00197          ERRORES-LOGICOS
00198
00199
00200      PARRAFOS-A-EJECUTAR SECTION.
00201
00202      100-GENERA-REG-PRECIO.
00203
00204      PERFORM 110-VALIDA-TRANSACCION
00205      IF VALID-TRANS
00206          PERFORM 125-LIBERA-PARA-CLASIFICAR
00207      ELSE
00208          MOVE WS-INVALID-FIELDS-MSG  TO WS-DETAIL-MESSAGE
00209          PERFORM 130-PRODUCE-ERROR-LOGICO
00210
00211      PERFORM 180-CONSIGUE-SIGUIENTE-TRAN
00212
00213
00214      110-VALIDA-TRANSACCION
00215
00216      IF      CREATION-PRICE NOT NUMERIC
00217          OR CREATION-QUANTITY NOT NUMERIC
00218          MOVE "NO" TO WS-VALID-TRANS-SW
00219      ELSE
00220          MOVE "YES" TO WS-VALID-TRANS-SW
00221
00222
00223      120-PRODUCE-REG-MAESTRO.
00224
00225      MOVE TRANSACT-ITEM-ID  TO PRICING-ITEM-ID
00226      MOVE TRANSACT-ITEM-DESCRIPTION
00227          TO PRICING-ITEM-DESCRIPTION
00228      MOVE TRANSACT-PRICE    TO PRICING-PRICE
00229      MOVE TRANSACT-QUANTITY  TO PRICING-QUANTITY-ONHAND
00230      PERFORM 150-ESCRIBE-REG-MAESTRO
00231      IF MASTER-STATUS-CODE-NOT-EQUAL "00"
00232          MOVE WS-DUPLICATE-MESSAGE  TO WS-DETAIL-MESSAGE
00233          PERFORM 130-PRODUCE-ERROR-LOGICO
00234
00235      PERFORM 140-DEVUELVE-SIGUIENTE-TRAN
00236
00237
00238      125-LIBERA-PARA-CLASIFICAR.
00239
00240      RELEASE TRANS-RECORD
00241          FROM CREATION-RECORD
00242
00243

```

Fig. 12-4 (cont.)

```

00244      130-PRODUCE-ERROR-LOGICO.
00245
00246      MOVE TRANSACT-ITEM-ID      TO WS-DETAIL-ITEM-ID
00247      MOVE TRANSACT-ITEM-DESCRIPTION
00248                      TO WS-DETAIL-DESCRIPTION
00249      MOVE TRANSACT-PRICE-X     TO WS-DETAIL-PRICE "
00250      MOVE TRANSACT-QUANTITY-X   TO WS-DETAIL-QUANTITY "
00251      PERFORM 160-IMPRIME-LINEA-DETALLE
00252
00253
00254      140-DEVUELVE-SIGUIENTE-TRAN.
00255
00256      RETURN FICHERO-ORD-TRANSACCIONES
00257          AT END
00258          MOVE "YES" TO WS-END-CREATION-SW
00259
00260
00261
00262      150-ESCRIBE-REG-MAESTRO.
00263
00264          WRITE PRICING-RECORD
00265
00266
00267      160-IMPRIME-LINEA-DETALLE.
00268
00269          WRITE ERROR-LINE
00270              FROM WS-DETAIL-LINE
00271              AFTER ADVANCING 2 LINES
00272              AT END-OF-PAGE
00273                  PERFORM 170-PRINT-HEADINGS
00274
00275
00276      170-IMPRIME-CABECERAS.
00277
00278          ADD 1 TO WS-PAGE-NUMBER
00279          MOVE WS-PAGE-NUMBER      TO WS-HEADING-PAGE
00280          WRITE ERROR-LINE :
00281              FROM WS-HEADING-LINE-1
00282              AFTER ADVANCING PAGE
00283          WRITE ERROR-LINE
00284              FROM WS-HEADING-LINE-2
00285              AFTER ADVANCING 2 LINES
00286
00287
00288      180-CONSIGUE-SIGUIENTE-TRAN.
00289
00290          READ FICHERO-INFO-CREACION
00291          AT END
00292          MOVE "YES" TO WS-END-CREATION-SW
00293
00294

```

Fig. 12-4 (cont.)

### Ordenación por un elemento compuesto

La eficiencia de una ordenación por varias claves puede incrementarse agrupando los campos clave en un elemento compuesto y ordenando después por el mismo. Esta técnica se puede aplicar cuando las claves que van a agruparse son (i) *mutuamente adyacentes* en el registro; (ii) todas alfanuméricas o numéricas sin signo en formato USAGE DISPLAY; (iii) todas crecientes o decrecientes; (iv) van desde la clave mayor (primera del grupo) a la clave menor (última del grupo).

#### EJEMPLO 12.12

```

SD FICHERO-ORDENACION...
01 REGISTRO-ORDENACION.
05 CLAVES-ORDENACION.

```

```

10 ID-REGION          PIC X(4).
10 ID-DEPARTAMENTO   PIC X(7).
10 ID-VENDEDOR        PIC X(3).
05 NOMBRE-REGION      PIC X(10).
05 NOMBRE-DEPARTAMENTO PIC X(20).
05 NOMBRE-VENDEDOR    PIC X(15).

```

.....

```

SORT FICHERO-ORDENACION
ON ASCENDING KEY CLAVES-ORDENACION

```

.....

### 12.3 FUSION DE FICHEROS CON EL VERBO MERGE

*Fusión* es la combinación de dos o más ficheros *ordenados* en uno sólo. Se habla de una “*n*-función” si se han combinado *n* ficheros; en el sistema IBM OS/VS necesariamente  $n \leq 8$ . Por lo general, las mismas utilidades del sistema que realizan ordenaciones de ficheros también tienen que ver con la fusión de ficheros. Dado que tanto la ordenación como la fusión de ficheros tienen que ver con la clasificación de los registros de un fichero, se aplican a ambas funciones los conceptos de orden ascendente o descendente y claves mayor, intermedia y menor.

La sintaxis de la instrucción MERGE (Fig. 12-5) es muy similar a la de la instrucción SORT. Para llevar a cabo tanto una ordenación como una fusión, el programador tiene que definir un fichero especial de ordenación o fusión con una instrucción SELECT en la división del entorno y con una instrucción SD en la división de datos. Aparte de las propias instrucciones SORT/MERGE, los requerimientos de COBOL para la ordenación y fusión son *idénticos*.

La instrucción MERGE debe definir los campos clave por los que los ficheros a fusionar *están ya ordenados* y también porque otras claves deben ordenar el fichero resultante. Las claves se escriben de la mayor a la menor al igual que en la instrucción SORT.

```

MERGE nombre-fichero-1
  ON {ASCENDING
       DESCENDING} KEY nombre-dato-1 [nombre-dato-2]...
  [ON {ASCENDING
       DESCENDING} KEY nombre-dato-3 [nombre-dato-3]...]
  [COLLATING SEQUENCE IS nombre-alfabeto]
  [USING nombre-fichero-2 nombre-fichero-3 [nombre-fichero-4]...
  GIVING nombre-fichero-5
  {OUTPUT PROCEDURE IS nombre-sección-1 [THROUGH {THRU} nombre-sección-2]}]

```

Fig. 12-5

Observe que no se permite INPUT PROCEDURE con MERGE. Los ficheros de USING (los que se van a fusionar de acuerdo con las especificaciones ASCENDING/DESCENDING KEY) deben estar cerrados en el instante en que se ejecuta MERGE, puesto que esta instrucción los abre automáticamente. Al finalizar la fusión, MERGE los vuelve a cerrar.

La opción GIVING especifica que los registros fusionados deben escribirse directamente en el fichero indicado. El fichero de GIVING debe estar cerrado al ejecutar MERGE puesto que se abre automáticamente, después se escriben los registros y al final se vuelve a cerrar.

Las funciones de la OUTPUT PROCEDURE son exactamente iguales a las que tenía en la instrucción SORT.

<i>Fichero A</i>	<i>Fichero B</i>	<i>Fichero C</i>	<i>Fichero fusionado</i>
20	10	5	5
40	30	25	10
60	50	35	20
80	70		25
	90		30
			35
			40
			50
			60
			70
			80
			90

Fig. 12-6

**EJEMPLO 12.13** Para fusionar los ficheros que se muestran en la Figura 12-6, se podría hacer con el programa siguiente:

ENVIRONMENT DIVISION.

```
.....  
SELECT FICHERO-A ASSIGN TO...  
SELECT FICHERO-B ASSIGN TO...  
SELECT FICHERO-C ASSIGN TO...  
SELECT FICHERO-FUSIONADO  
ASSIGN TO SORTWK.  
.....
```

DATA DIVISION.

FILE SECTION.

```
FD FICHERO-A...  
01 REGISTRO-A...
```

```
FD FICHERO-B...  
01 REGISTRO-B...
```

```
FD FICHERO-C...  
01 REGISTRO-C...
```

```
SD FICHERO-FUSIONADO  
RECORD CONTAINS...  
01 REGISTRO-FUSIONADO.  
05 CLAVE-FUSION-1...  
05 CAMPO-DATOS-1...  
05 CLAVE-FUSION-2...  
05 CAMPO-DATOS-2...
```

PROCEDURE DIVISION.

```
MERGE FICHERO-FUSIONADO  
ON ASCENDING KEY CLAVE-FUSION-1  
ON DESCENDING KEY CLAVE-FUSION-2  
USING FICHERO-A FICHERO-B FICHERO-C  
OUTPUT PROCEDURE IS PROCESA-REG-FUSIONADOS  
STOP RUN
```

PROCESA-REG-FUSIONADOS SECTION.

PARRAFOS-A-EJECUTAR SECTION.  
CAPTURA-REG-FUSIONADO  
RETURN FICHERO-FUSIONADO  
AT END...

El programa seguiría siendo válido si en lugar de MERGE se pone SORT; de hecho esta sustitución es *esencial* si los ficheros no están preordenados. Por supuesto, si los ficheros están ordenados, MERGE es mucho más eficiente que SORT.

Observe que cuando se especifica más de un fichero en la cláusula USING de SORT, se combinan y ordenan todos los registros de todos los ficheros. Así se tiene un medio de fusionar ficheros *sin clasificar*.

### Preguntas de repaso

- 12.1 Diferencia entre ordenación de ficheros y de tablas.
- 12.2 Explique: campo clave, ordenación creciente, ordenación decreciente, clave mayor, clave intermedia, clave menor.
- 12.3 Explique la relación entre la utilidad del sistema para ordenación/fusión y las instrucciones de COBOL SORT/MERGE.
- 12.4 ¿Qué es un “fichero especial de ordenación”?
- 12.5 Explique la función de las instrucciones RELEASE y RETURN.
- 12.6 Explique las restricciones de las instrucciones SELECT y SD para definir ficheros especiales de ordenación/fusión.
- 12.7 Dé algunos ejemplos de cuándo podría ser útil la INPUT PROCEDURE.
- 12.8 Dé algunos ejemplos de cuándo podría ser útil la OUTPUT PROCEDURE.
- 12.9 Dé un ejemplo en el que la INPUT PROCEDURE construya los registros y los ceda después a la utilidad de ordenación/fusión.
- 12.10 Explique cualquier restricción que exista en las claves de ordenación cuando se manejan registros de longitud variable.
- 12.11 Explique el propósito de la cláusula COLLATING SEQUENCE en las instrucciones SORT y MERGE.
- 12.12 ¿Por qué no son comunes las ordenaciones con USING...GIVING... en los programas en COBOL?
- 12.13 Comente la estructura de la división de procedimientos al utilizar SORT o MERGE.
- 12.14 Comente lo que sucede al ejecutar una instrucción SORT.
- 12.15 ¿Qué sucede si se ejecuta STOP RUN dentro de la INPUT u OUTPUT PROCEDURE?
- 12.16 Comente las relaciones entre INPUT y OUTPUT PROCEDURE bajo el control de la instrucción SORT.
- 12.17 Explique de qué modo agrupando las claves se puede mejorar la eficiencia en la ordenación. ¿Qué restricciones se aplican a esta técnica?
- 12.18 Distinga entre ordenación y fusión de ficheros.

- 12.19 Dé algunos ejemplos de cuándo puede ser útil la fusión de ficheros.
- 12.20 ¿Qué se entiende por una "3-fusión"?
- 12.21 ¿Por qué los ficheros de USING y GIVING deben estar cerrados al ejecutar SORT o MERGE?
- 12.22 ¿Cómo se debe programar una fusión si los ficheros no están preordenados?

### Problemas resueltos

- 12.23 (a) Identifique las claves mayor, intermedia y menor en los registros de abajo. (b) ¿Qué claves son ascendentes y cuáles descendentes?

<i>Nombre</i>	<i>Región</i>	<i>Número de cuenta</i>
PERELMAN	10	5
SNYDER	10	5
ABEL	10	3
CONVERSE	10	3
FOLKERS	20	8
GETZ	20	8
BAKER	20	4

- (a) Clave mayor, región; clave intermedia, número de cuenta; clave menor, nombre.  
 (b) Región y nombre son ascendentes; número de cuenta es descendente.

- 12.24 Además de la utilidad de ordenación/fusión, qué otras utilidades suelen estar disponibles en un sistema de computadoras.

Programas para copiar ficheros, para imprimir el contenido de un fichero, para listar todos los ficheros de un disco o cinta, para recuperar ficheros de un disco o cinta dañados, etc.

- 12.25 ¿Cuál es la diferencia entre las definiciones COBOL de un fichero de ordenación y otro de fusión?

Ninguna. Ambos requieren las instrucciones SELECT y SD. No hay forma de saber si uno de estos ficheros se va a utilizar en una instrucción SORT o en una instrucción MERGE excepto mirando la división de procedimientos.

- 12.26 ¿Cuál es el formato de ASSIGN TO para un fichero de ordenación/clasificación?

Sirve cualquier nombre legal, ya que ASSIGN TO se trata como si fuera un comentario.

- 12.27 Indique cuándo se usaría INPUT PROCEDURE, GIVING, OUTPUT PROCEDURE en lo siguiente: (a) un programa que imprima etiquetas de correo para todos los registros de una cinta; (b) un programa que calcule e imprima las notas medias de todos los estudiantes de un curso de una universidad; (c) un programa que realice una copia de seguridad de un fichero de cheques para clientes a pagar, que debe estar clasificado por el número de cuenta en el libro mayor en lugar de por el número del cheque; (d) un programa de edición que produzca un fichero en disco de transacciones validadas.

- (a) USING, OUTPUT PROCEDURE (para formatear el informe).  
 (b) INPUT PROCEDURE (para elegir los estudiantes) y OUTPUT PROCEDURE (para calcular la nota media).

- (c) USING y GIVING (en la vida real se ejecutaría directamente la utilidad de ordenación/fusión en lugar de hacerlo a través de un programa en COBOL).
- (d) INPUT PROCEDURE (para filtrar las transacciones válidas) y GIVING.

**12.28** ¿Qué está equivocado en lo siguiente?

```
SD UN-FICHERO-ORDENACION
BLOCK CONTAINS 10 RECORDS
RECORD CONTAINS 200 CHARACTERS
LABEL RECORDS ARE STANDARD
```

Sólo se permiten las cláusulas RECORD CONTAINS... y DATA RECORDS ARE... en la definición de un fichero de ordenación/fusión (SD): BLOCK CONTAINS... y LABEL RECORDS... son cláusulas inválidas.

**12.29** Escriba un programa que clasifique registros de ventas por las ventas medias por cliente calculadas en una INPUT PROCEDURE.

```
SELECT FICHERO-VENTAS ASSIGN TO...
SELECT FICHERO-ORDENACION ASSIGN TO ANYTHING.
```

```
.....
```

FD	FICHERO-VENTAS...	
01	REGISTRO-VENTAS.	
05	ID-VENDEDOR	PIC X(7).
05	TOTAL-VENTAS	PIC S9(5)V99 COMP.
05	NUMERO-CLIENTES	PIC S9(3) COMP.
SD	FICHERO-ORDENACION	
	RECORD-CONTAINS...	
01	REGISTRO-ORDENACION.	
05	ID-VENTAS-ORD	PIC X(7).
05	TOTAL-VENTAS-ORD	PIC S9(5)V99 COMP.
05	NUM-CLIENTES-ORD	PIC S9(3) COMP.
05	VENTAS-MEDIAS-ORD	PIC S9(5)V99 COMP.

```
.....
```

```
SORT FICHERO-ORDENACION
ON DESCENDING KEY VENTAS-MEDIAS-ORD
INPUT PROCEDURE IS CALCULO-MEDIA
OUTPUT PROCEDURE IS IMPRIME-INFORME
STOP RUN
```

```
CALCULO-MEDIA SECTION.
OPEN INPUT FICHERO-VENTAS
MOVE "NO" TO INTER-FIN-FICHERO
PERFORM CONSIGUE-REG-VENTAS
PERFORM LIBERA-UN-REGISTRO
    UNTIL INTER-FIN-FICHERO EQUAL "SI"
CLOSE FICHERO-VENTAS
```

```
IMPRIME-INFORME SECTION.
```

```
.....
```

PARRAFOS A EJECUTAR SECTION.

CONSIGUE-REG-VENTAS.  
 READ FICHERO-VENTAS  
 AT END MOVE "SI" TO INTER-FIN-FICHERO

LIBERA-UN-REGISTRO.  
 MOVE ID-VENDEDOR TO ID-VENTAS-ORD

```

MOVE TOTAL-VENTAS      TO TOTAL-VENTAS-ORD
MOVE NUMERO-CLIENTES   TO NUM-CLIENTES-ORD
DIVIDE TOTAL-VENTAS BY NUMERO-CLIENTES
      GIVING VENTAS-MEDIAS-ORD
RELEASE REGISTRO-ORDENACION
PERFORM CONSIGUE-REG-VENTAS

```

- 12.30 La secuencia EBCDIC coloca los dígitos 0-9 después de las letras A-Z. Realice una ordenación que coloque los dígitos 0-9 antes que las letras.

```

SPECIAL-NAMES.
  ORDEN-PEDIDOS-CLIENTES IS "0" THRU "9"
      "A" THRU "Z"

```

```

SORT FICHERO-ORDENACION
  ON ASCENDING KEY CUALQUIERA
  COLLATING SEQUENCE IS ORDEN-PEDIDOS-CLIENTES
  INPUT PROCEDURE IS ENTRADA
  OUTPUT PROCEDURE IS PROCESO

```

- 12.31 Comente cómo se pueden ordenar campos fecha.

Las fechas definidas por PIC X(6) o PIC 9(6) se pueden clasificar si los seis caracteres están en la forma aammdd (año, mes, día) en vez de en la forma mmddaa (mes, día, año). La forma "aammdd" preserva el orden apropiado. Otra forma consiste en utilizar la fecha juliana aaddd (Sección 6.6). Ninguno de estos procedimientos es válido cuando se cambia de siglo.

- 12.32 Ciertos compiladores COBOL no permiten que una INPUT/OUTPUT PROCEDURE ejecute con PERFORM un párrafo localizado fuera de la INPUT/OUTPUT SECTION. Asumiendo que su compilador es de este tipo, vuelva a escribir la INPUT PROCEDURE del Problema 12.29.

```

CALCULO-MEDIA SECTION.
  OPEN INPUT FICHERO-VENTAS
  MOVE "NO" TO INTER-FIN-FICHERO
  PERFORM CONSIGUE-REG-VENTAS
  PERFORM LIBERA-UN-REGISTRO
    UNTIL INTER-FIN-FICHEROS "SI"
  CLOSE FICHERO-VENTAS
  GO TO SALIDA-CALCULO-MEDIA

```

CONSIGUE-REG-VENTAS.

LIBERA-UN-REGISTRO.

SALIDA-CALCULO-MEDIA.

EXIT.

IMPRIME-INFORME SECTION.

GO TO..., instrucción ampliamente usada en el COBOL no estructurado, es necesaria en el programa de arriba para impedir que se ejecuten CAPTURA-REGISTRO y DEVUELVE-REGISTRO una vez más después de haber cargado FICHERO-VENTAS. "EXIT" es una instrucción para no hacer nada pero que sirve para que CALCULO-MEDIA-SALIDA sea el párrafo final de la SECTION.

- 12.33 Redefina el siguiente registro de ordenación para que la tarea sea más eficiente:

```
SD FICHERO-ORDENACION...
01 REG-ORDENACION.
  05 ID-EMPLEADO          PIC X(4).
  05 NOMBRE                PIC X(20).
  05 DEPARTAMENTO          PIC XX.
  05 NUM-FAMILIARES        PIC S9      COMP-3.
  05 SALARIO-HORA           PIC S9(2)V99 COMP.
```

```
.....
```

```
SORT FICHERO-ORDENACION
  ON DESCENDING KEY SALARIO-HORA
  ON ASCENDING KEY DEPARTAMENTO
  ON ASCENDING KEY ID-EMPLEADO
```

```
SD FICHERO-ORDENACION...
01 REG-ORDENACION.
  05 CLAVES-CLASIFICACION.
    10 DEPARTAMENTO        PIC XX.
    10 ID-EMPLEADO          PIC X(4).
    05 NOMBRE                PIC X(20).
    05 NUM-FAMILIARES        PIC S9      COMP-3.
    05 SALARIO-HORA           PIC S9(2)V99 COMP.
```

```
.....
```

```
SORT FICHERO-ORDENACION
  ON DESCENDING KEY SALARIO-HORA
  ON ASCENDING KEY CLAVES-CLASIFICACION
```

- 12.34 ¿Cómo se pueden fusionar los dos ficheros siguientes?

Fichero A: 10, 5, 18, 17, 23

Fichero B: 20, 30, 40, 50

Como el fichero A no está ordenado, deben fusionarse por medio de la instrucción SORT:

```
SORT FICHERO-ORDENACION
  ON...
  USING FICHERO-A FICHERO-B
  OUTPUT PROCEDURE IS...
```

- 12.35 ¿Qué está equivocado en lo siguiente?

```
PROCEDURE DIVISION.
  OPEN INPUT FICHERO-A
  FICHERO-B
  MERGE FICHERO-FUSION-TRABAJO
  ON ASCENDING KEY CLAVE-FUSION
  USING FICHERO-A
  FICHERO-B
  OUTPUT PROCEDURE IS PROCESA-REGISTROS
```

La opción USING requiere que los ficheros estén *cerrados* al ejecutar SORT o MERGE (puesto que la utilidad de ordenación/fusión los abre automáticamente).

## Ejercicios de programación

- 12.36 Escriba un programa para una agencia de cobro con los siguientes datos:

**Entrada.** Tarjetas que contienen el número de cuenta (columnas 1-6); nombre (columnas 7-30); importe (columnas 31-37, 2 decimales).

**Salida.** Informe impreso: comenzando en una página nueva con título y cabeceras apropiadas. A lo largo de la página deben imprimirse número de cuenta, nombre e importe. Diseñe el formato como le guste. Ordene las cuentas por orden decreciente del importe debido. Al final, imprima el número total de cuentas y el importe total.

- 12.37 Escriba un programa para ayudar a una universidad a controlar los pagos de los estudiantes.

**Entrada.** Tarjetas con ID estudiante [X(9)], nombre [X(20)] e importe debido [9(2)V9(2)].

**Salida.** Imprima un informe con título y cabeceras. A lo largo de la página se imprimirá la información de los estudiantes. El informe debe comenzar con el estudiante que deba la mayor cantidad y continuar hasta el que deba la menor cantidad. Al final del informe, imprima el número total de estudiantes que deben dinero y la cantidad media debida.

- 12.38 Escriba un programa que produzca un informe de errores ordenados y un fichero maestro ordenado.

**Entrada.**

Nombre campo	Características	Posición
Número cliente	9(6)	1-6
Nombre cliente	X(20)	11-30
Número artículo	X(6)	31-36
Número pedido	X(5)	38-42
Fecha pedido	9(6)	44-49
Cantidad pedido	9(3)	51-53
Precio	999V99	55-59

Las tarjetas no están ordenadas. Defina número de cliente, número pedido y número de artículo como alfanumérico.

**Salida.** (1) Un informe que contenga información de aquellas tarjetas que contengan errores. Valide comprobando qué número de cliente, número de artículo, fecha, cantidad y precio son numéricos y que el precio es mayor que cero. (2) Un fichero maestro en disco ordenado por número de cliente y para cada cliente por número de artículo. Habrá un registro de este fichero por cada cliente. El formato del fichero será:

Nombre campo	Características	Posición
Número cliente	X(6)	1-6
Nombre cliente	X(20)	7-26
Número de pedidos	9(COMP-3)	27

seguido de los datos de cada pedido:

Número de artículo	X(6)
Número de pedido	X(5)
Fecha	9(6)
Cantidad	9(3)
Precio	999V99

Habrá como máximo 4 pedidos por cliente y la longitud total será de 127 bytes.

**Procesamiento.** Valide los registros de entrada, imprima un informe de tarjetas erróneas. Clasifique las tarjetas válidas por número de artículo dentro de cada número de cliente. Utilice los registros clasificados para crear el fichero maestro en disco. Si hubiera más de 4 pedidos para un cliente, sólo se incluirán los cuatro primeros en el fichero maestro. Imprima mensajes de error para los pedidos extra e ignore dichas tarjetas.

- 12.39 Actualice con los pedidos de hoy el fichero maestro creado en el Problema 12.38. Puede utilizar el algoritmo maestro-transacciones comentado en el Capítulo 11.

**Entrada.** (1) Un fichero maestro con el formato del Problema 12.38, pero con registros de longitud variable. Un registro será lo largo que sea necesario para contener el número de pedidos que sea. Los registros que contengan un pedido serán de 52 caracteres; aquellos que tengan 4 pedidos tendrán 127 caracteres. El fichero reside en disco. (2) Un fichero de transacciones con idéntico formato al del Problema 12.38. Será un fichero de tarjetas.

**Salida.** (1) Fichero maestro actualizado. (2) Un registro de transacciones, es decir, un informe impreso que muestre los cambios realizados en el maestro y que visualice mensajes de error para aquellos pedidos que no han actualizado el fichero.

**Procesamiento.** El fichero maestro estará ordenado por número de cliente; clasifique el fichero de pedidos en el mismo orden (todas las tarjetas han sido validadas anteriormente). En la rutina de actualización:

- si hay un pedido para un cliente que no está en el maestro, imprima un mensaje de error e ignore el pedido.
- si el número de cliente de un pedido coincide con el de algún registro del maestro, entonces:
  - si el artículo pedido ya está en el maestro, actualice sumando la cantidad a la que figure en el maestro y cambie la fecha de pedido haciendo figurar la de hoy.
  - si el artículo pedido no está en el maestro, imprima un mensaje de error en el caso de que no hubiera espacio para insertarlo en el maestro. Si hubiera espacio, insértelo en el maestro de forma que se mantenga el orden por número de artículo. **Observación:** Asegúrese de escribir todos los registros del maestro que se lean, aunque no haya pedidos para ellos.

## Apéndice A

### Palabras reservadas de COBOL

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
ACCEPT	x	-	x
ACCESS	x	-	x
ACTUAL	-	-	x
ADD	x	-	x
ADVANCING	x	-	x
AFTER	x	-	x
ALL	x	-	x
ALPHABET	-	x	-
ALPHABETIC	x	-	x
ALPHANUMERIC	-	x	-
ALPHANUMERIC-EDITED	-	x	-
ALSO	x	-	x
ALTER	x	-	x
ALTERNATE	x	-	x
AND	x	-	x
ANY	-	x	-
APPLY	-	-	x
ARE	x	-	x
AREA	x	-	x
AREAS	x	-	x
ASCENDING	x	-	x
ASSIGN	x	-	x
AT	x	-	x
AUTHOR	x	-	x
BASIS	-	x	-
BEFORE	x	-	x
BEGINNING	-	-	x
BINARY	-	x	-
BIT	-	x	-
BITS	-	x	-
BLANK	x	-	x
BLOCK	x	-	x
BOOLEAN	-	x	-
BOTTOM	x	-	x
BY	x	-	x
CALL	x	-	x
CANCEL	x	-	x
CBL	-	-	x
CD	x	-	x
CF	x	-	x
CH	x	-	x
CHANGED	-	-	x
CHARACTER	x	-	x
CHARACTERS	x	-	x

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
CLOCK-UNITS	x	-	-
CLOSE	x	-	x
COBOL	x	-	-
CODE	x	-	x
CODE-SET	x	-	x
COLLATING	x	-	x
COLUMN	x	-	x
COMMA	x	-	x
COMMIT	-	x	-
COMMON	-	x	-
COMMUNICATION	x	-	x
COMP	x	-	x
COMP-1	-	-	x
COMP-2	-	-	x
COMP-3	-	-	x
COMP-4	-	-	x
COMPUTATIONAL	x	-	x
COMPUTATIONAL-1	-	-	x
COMPUTATIONAL-2	-	-	x
COMPUTATIONAL-3	-	-	x
COMPUTATIONAL-4	-	-	x
COMPUTE	x	-	x
CONFIGURATION	x	-	x
CONNECT	-	x	-
CONSOLE	-	-	x
CONTAINS	x	-	x
CONTENT	-	x	-
CONTINUE	-	x	-
CONTROL	x	-	x
CONTROLS	x	-	x
CONVERTING	-	x	-
COPY	x	-	x
CORE-INDEX	-	-	x
CORR	x	-	x
CORRESPONDING	x	-	x
COUNT	x	-	x
CSP	-	-	x
CURRENCY	x	-	x
CURRENT	-	x	-
CURRENT-DATE	-	-	x
C01	-	-	x
C02	-	-	x
C03	-	-	x
C04	-	-	x
C05	-	-	x
C06	-	-	x
C07	-	-	x
C08	-	-	x
C09	-	-	x
C10	-	-	x
C11	-	-	x
C12	-	-	x
DATA	x	-	x
DATE	x	-	x
DATE-COMPILED	x	-	x
DATE-WRITTEN	x	-	x
DAY	x	-	x

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
DAY-OFF-WEEK	-	x	-
DB	-	x	-
DB-ACCESS-CONTROL-KEY	-	x	-
DB-DATA-NAME	-	x	-
DB-EXCEPTION	-	x	-
DB-RECORD-NAME	-	x	-
DB-SET-NAME	-	x	-
DB-STATUS	-	x	-
DE	x	-	x
DEBUG	-	-	x
DEBUG-CONTENTS	x	-	x
DEBUG-ITEM	x	-	x
DEBUG-LINE	x	-	x
DEBUG-NAME	x	-	x
DEBUG-SUB-1	x	-	x
DEBUG-SUB-2	x	-	x
DEBUG-SUB-3	x	-	x
DEBUGGING	x	-	x
DECIMAL-POINT	x	-	x
DECLARATIVES	x	-	x
DELETE	x	-	x
DELIMITED	x	-	x
DELIMITER	x	-	x
DEPENDING	x	-	x
DESCENDING	x	-	x
DESTINATION	x	-	x
DETAIL	x	-	x
DISABLE	x	-	x
DISCONNECT	-	x	-
DIS	-	-	x
DISPLAY	x	-	x
DISPLAY-n	-	x	-
DISPLAY-ST	-	-	x
DIVIDE	x	-	x
DIVISION	x	-	x
DOWN	x	-	x
DUPLICATE	-	x	-
DUPLICATES	x	-	x
DYNAMIC	x	-	x
EGI	x	-	x
EJECT	-	-	x
ELSE	x	-	x
EMI	x	-	x
EMPTY	-	x	-
ENABLE	x	-	x
END	x	-	x
END-ADD	-	x	-
END-CALL	-	x	-
END-COMPUTE	-	x	-
END-DELETE	-	x	-
END-DIVIDE	-	x	-
END-EVALUATE	-	x	-
END-IF	-	x	-
END-MULTIPLY	-	x	-
END-OF-PAGE	x	-	x
END-PERFORM	-	x	-
END-READ	-	x	-

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
END-RECEIVE	-	x	-
END-RETURN	-	x	-
END-REWRITE	-	x	-
END-SEARCH	-	x	-
END-START	-	x	-
END-STRING	-	x	-
END-SUBTRACT	-	x	-
END-UNSTRING	-	x	-
END-WRITE	-	x	-
ENDING	-	-	x
ENTER	x	-	x
ENTRY	-	-	x
ENVIRONMENT	x	-	x
EOP	x	-	x
EQUAL	x	-	x
EQUALS	-	x	-
ERASE	-	x	-
ERROR	x	-	x
ESI	x	-	x
EVALUATE	-	x	-
EVERY	x	-	x
EXAMINE	-	-	x
EXCEEDS	-	x	-
EXCEPTION	x	-	x
EXCLUSIVE	-	x	-
EXHIBIT	-	-	x
EXIT	x	-	x
EXOR	-	x	-
EXTEND	x	-	x
EXTERNAL	-	x	-
FALSE	-	x	-
FD	x	-	x
FILE	x	-	x
FILE-CONTROL	x	-	x
FILE-LIMIT	-	-	x
FILE-LIMITS	-	-	x
FILLER	x	-	x
FINAL	x	-	x
FIND	-	x	-
FINISH	-	x	-
FIRST	x	-	x
FOOTING	x	-	x
FOR	x	-	x
FREE	-	x	-
FROM	x	-	x
FUNCTION	-	x	-
GENERATE	x	-	x
GET	-	x	-
GIVING	x	-	x
GLOBAL	-	x	-
GO	x	-	x
GREATER	x	-	x
GROUP	x	-	x
HEADING	x	-	x
HIGH-VALUE	x	-	x
HIGH-VALUES	x	-	x

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
I-O	x	-	x
I-O-CONTROL	x	-	x
ID	-	-	x
IDENTIFICATION	x	-	x
IF	x	-	x
IN	x	-	x
INDEX	x	-	x
INDEX-n	-	x	-
INDEXED	x	-	x
INDICATE	x	-	x
INITIAL	x	-	x
INITIALIZE	-	x	x
INITIATE	x	-	x
INPUT	x	-	x
INPUT-OUTPUT	x	-	x
INSERT	-	-	x
INSPECT	x	-	x
INSTALLATION	x	-	x
INTO	x	-	x
INVALID	x	-	x
IS	x	-	x
JUST	x	-	x
JUSTIFIED	x	-	x
KEEP	-	x	-
KEY	x	-	x
LABEL	x	-	x
LAST	x	-	x
LD	-	x	-
LEADING	x	-	x
LEAVE	-	-	x
LEFT	x	-	x
LENGTH	x	-	x
LESS	x	-	x
LIMIT	x	-	x
LIMITS	x	-	x
LINAGE	x	-	x
LINAGE-COUNTER	x	-	x
LINE	x	-	x
LINE-COUNTER	x	-	x
LINES	x	-	x
LINKAGE	x	-	x
LOCALLY	-	x	-
LOCK	x	-	x
LOW-VALUE	x	-	x
LOW-VALUES	x	-	x
MEMBER	-	x	-
MEMORY	x	-	x
MERGE	x	-	x
MESSAGE	x	-	x
MODE	x	-	x
MODIFY	-	x	-
MODULES	x	-	x
MORE-LABELS	-	-	x
MOVE	x	-	x

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
MULTIPLE	x	-	x
MULTIPLY	x	-	x
NAMED	-	-	x
NATIVE	x	-	x
NEGATIVE	x	-	x
NEXT	x	-	x
NO	x	-	x
NOMINAL	-	x	x
NOT	x	-	x
NOTE	-	-	x
NULL	-	x	-
NUMBER	x	-	x
NUMERIC	x	-	x
NUMERIC-EDITED	-	x	-
OBJECT-COMPUTER	x	-	x
OCCURS	x	-	x
OF	x	-	x
OFF	x	-	x
OMITTED	x	-	x
ON	x	-	x
OPEN	x	-	x
OPTIONAL	x	-	x
OR	x	-	x
ORDER	-	x	-
ORGANIZATION	x	-	x
OTHER	-	x	-
OTHERWISE	-	-	x
OUTPUT	x	-	x
OVERFLOW	x	-	x
OWNER	-	x	-
PACKED-DECIMAL	-	x	-
PADDING	-	x	-
PAGE	x	-	x
PAGE-COUNTER	x	-	x
PASSWORD	-	-	x
PERFORM	x	-	x
PF	x	-	x
PH	x	-	x
PIC	x	-	x
PICTURE	x	-	x
PLUS	x	-	x
POINTER	x	-	x
POSITION	x	-	x
POSITIONING	-	-	x
POSITIVE	x	-	x
PRINTING	x	-	-
PRIOR	-	x	-
PROCEDURE	x	-	x
PROCEDURES	x	-	x
PROCEED	x	-	x
PROCESSING	-	-	x
PROGRAM	x	-	x
PROGRAM-ID	x	-	x
PROTECTED	-	x	-
PURGE	-	x	-

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
QUEUE	x	-	x
QUOTE	x	-	x
QUOTES	x	-	x
RANDOM	x	-	x
RD	x	-	x
READ	x	-	x
READY	-	-	x
REALM	-	x	-
REALMS	-	x	-
RECEIVE	x	-	x
RECONNECT	-	x	-
RECORD	x	-	x
RECORD-NAME	-	x	-
RECORD-OVERFLOW	-	-	x
RECORDS	x	-	x
REDEFINES	x	-	x
REEL	x	-	x
REFERENCE	-	x	-
REFERENCES	x	-	x
RELATIVE	x	-	x
RELEASE	x	-	x
RELOAD	-	-	x
REMAINDER	x	-	x
REMARKS	-	-	x
REMOVAL	x	-	x
RENAMES	x	-	x
REORG-CRITERIA	-	-	x
REPEATED	-	x	-
REPLACE	-	x	-
REPLACING	x	-	x
REPORT	x	-	x
REPORTING	x	-	x
REPORTS	x	-	x
REREAD	-	-	x
RERUN	x	-	x
RESERVE	x	-	x
RESET	x	-	x
RETAINING	-	x	-
RETRIEVAL	-	x	-
RETURN	x	-	x
RETURN-CODE	-	-	x
REVERSED	x	-	x
REWIND	x	-	x
REWRITE	x	-	x
RF	x	-	x
RH	x	-	x
RIGHT	x	-	x
ROLLBACK	-	x	-
ROUNDED	-	-	x
RUN	x	-	x
SAME	x	-	x
SD	x	-	x
SEARCH	x	-	x
SECTION	x	-	x
SECURITY	x	-	x
SEEK	-	-	x

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
SEGMENT	x	-	x
SEGMENT-LIMIT	x	-	x
SELECT	x	-	x
SELECTIVE	-	-	x
SEND	x	-	x
SENTENCE	x	-	x
SEPARATE	x	-	x
SEQUENCE	x	-	x
SEQUENTIAL	x	-	x
SET	x	-	x
SETS	-	x	-
SIGN	x	-	x
SIZE	x	-	x
SKIP-1	-	-	x
SKIP-2	-	-	x
SKIP-3	-	-	x
SORT	x	-	x
SORT-CORE-SIZE	-	-	x
SORT-FILE-SIZE	-	-	x
SORT-MERGE	x	-	x
SORT-MESSAGE	-	-	x
SORT-MODE-SIZE	-	-	x
SORT-RETURN	-	-	x
SOURCE	x	-	x
SOURCE-COMPUTER	x	-	x
SPACE	x	-	x
SPACES	x	-	x
SPECIAL-NAMES	x	-	x
STANDARD	x	-	x
STANDARD-1	x	-	x
STANDARD-2	-	x	-
START	x	-	x
STATUS	x	-	x
STOP	x	-	x
STORE	-	-	x
STRING	x	-	x
SUB-QUEUE-1	x	-	x
SUB-QUEUE-2	x	-	x
SUB-QUEUE-3	x	-	x
SUB-SCHEMA	-	x	-
SUBTRACT	x	-	x
SUM	x	-	x
SUPPRESS	x	-	x
SYMBOLIC	x	-	x
SYNC	x	-	x
SYNCHRONIZED	x	-	x
SYSIN	-	-	x
SYSOUT	-	-	x
SYSPUNCH	-	-	x
S01	-	-	x
S02	-	-	x
TABLE	x	-	x
TALLY	-	-	x
TALLYING	x	-	x
TAPE	x	-	x
TENANT	-	x	-
TERMINAL	x	-	x

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
TERMINATE	×	-	×
TEST	-	×	-
TEXT	×	-	×
THAN	×	-	×
THEN	-	-	×
THROUGH	×	-	×
THRU	×	-	×
TIME	×	-	×
TIME-OF-DAY	-	-	×
TIMES	×	-	×
TO	×	-	×
TOP	×	-	×
TOTALED	-	-	×
TOTALING	-	-	×
TRACE	-	-	×
TRACK-AREA	-	-	×
TRACK-LIMIT	-	-	×
TRACKS	-	-	×
TRAILING	×	-	×
TRANSFORM	-	-	×
TRUE	-	×	-
TYPE	×	-	×
UNEQUAL	-	×	-
UNIT	×	-	×
UNSTRING	×	-	×
UNTIL	×	-	×
UP	×	-	×
UPDATE	-	×	-
UPON	×	-	×
UPSI-0	-	-	×
UPSI-1	-	-	×
UPSI-2	-	-	×
UPSI-3	-	-	×
UPSI-4	-	-	×
UPSI-5	-	-	×
UPSI-6	-	-	×
UPSI-7	-	-	×
USAGE	×	-	×
USAGE-MODE	-	×	-
USE	×	-	×
USING	×	-	×
VALUE	×	-	×
VALUES	×	-	×
VARYING	×	-	×
WHEN	×	-	×
WHEN-COMPILED	-	-	×
WITH	×	-	×
WITHIN	-	×	-
WORDS	×	-	×
WORKING-STORAGE	×	-	×
WRITE	×	-	×
WRITE-ONLY	-	-	×
ZERO	×	-	×
ZEROES	×	-	×
ZEROS	×	-	×

<i>Palabra reservada</i>	<i>COBOL de 1974 norma ANS</i>	<i>COBOL de 1980 norma CODASYL</i>	<i>COBOL de IBM OS/VS</i>
+	x	-	x
-	x	-	x
*	x	-	x
/	x	-	x
**	x	-	x
<	x	-	x
>	x	-	x
=	x	-	x

## Apéndice B

### Secuencias de orden de caracteres

Secuencia EBCDIC ascendente

Representación decimal	Representación binaria	Símbolo	Significado
0	00000000		
74	01001010	,	signo céntimo
75	01001011	.	punto, punto decimal
76	01001100	<	signo menor que
77	01001101	(	paréntesis izquierdo
78	01001110	+	signo más
79	01001111		barra vertical, OR lógico
80	01010000	&	ampersand
90	01011010	!	exclamación
91	01011011	\$	dólar
92	01011100	*	asterisco
93	01011101	)	paréntesis derecho
94	01011110	;	punto y coma
95	01011111	¬	NOT lógico
96	01100000	-	menos, guión
97	01100001	/	barra
107	01101011	,	coma
108	01101100	%	porcentaje
109	01101101	_	subrayado
110	01101110	>	signo mayor que
111	01101111	?	interrogación
122	01111010	:	dos puntos
123	01111011	#	número, almohadilla
124	01111100	@	signo @

Secuencia EBCDIC ascendente (*cont.*)

<i>Representación decimal</i>	<i>Representación binaria</i>	<i>Símbolo</i>	<i>Significado</i>
125	01111101	'	apóstrofo
126	01111110	=	signo igual
127	01111111	"	dobles comillas
129	10000001	a	
130	10000010	b	
131	10000011	c	
132	10000100	d	
133	10000101	e	
134	10000110	f	
135	10000111	g	
136	10001000	h	
137	10001001	i	
145	10010001	j	
146	10010010	k	
147	10010011	l	
148	10010100	m	
149	10010101	n	
150	10010110	o	
151	10010111	p	
152	10011000	q	
153	10011001	r	
162	10100010	s	
163	10100011	t	
164	10100100	u	
165	10100101	v	
166	10100110	w	
167	10100111	x	
168	10101000	y	
169	10101001	z	
193	11000001	A	

Secuencia EBCDIC ascendente (*cont.*)

<i>Representación decimal</i>	<i>Representación binaria</i>	<i>Símbolo</i>	<i>Significado</i>
194	11000010	B	
195	11000011	C	
196	11000100	D	
197	11000101	E	
198	11000110	F	
199	11000111	G	
200	11001000	H	
201	11001001	I	
209	11010001	J	
210	11010010	K	
211	11010011	L	
212	11010100	M	
213	11010101	N	
214	11010110	O	
215	11010111	P	
216	11011000	Q	
217	11011001	R	
226	11100010	S	
227	11100011	T	
228	11100100	U	
229	11100101	V	
230	11100110	W	
231	11100111	X	
232	11101000	Y	
233	11101001	Z	
240	11110000	0	
241	11110001	1	
242	11110010	2	
243	11110011	3	
244	11110100	4	

## Secuencia EBCDIC ascendente (cont.)

<i>Representación decimal</i>	<i>Representación binaria</i>	<i>Símbolo</i>	<i>Significado</i>
245	11110101	5	
246	11110110	6	
247	11110111	7	
248	11111000	8	
249	11111001	9	

## Secuencia ASCII ascendente

<i>Representación decimal</i>	<i>Representación binaria</i>	<i>Símbolo</i>	<i>Significado</i>
0	00000000		nulo
32	00100000	SP	espacio
33	00100001		OR lógico
34	00100010	"	dobles comillas
35	00100011	#	número, almohadilla
36	00100100	\$	dólar
37	00100101	%	porcentaje
38	00100110	&	ampersand
39	00100111	'	apóstrofo
40	00101000	(	abre paréntesis
41	00101001	)	cierra paréntesis
42	00101010	*	asterisco
43	00101011	+	signo más
44	00101100	,	coma
45	00101101	-	menos, guión
46	00101110	.	punto, punto decimal
47	00101111	/	barra
48	00110000	0	
49	00110001	1	
50	00110010	2	
51	00110011	3	

Secuencia ASCII ascendente (*cont.*)

<i>Representación decimal</i>	<i>Representación binaria</i>	<i>Símbolo</i>	<i>Significado</i>
52	00110100	4	
53	00110101	5	
54	00110110	6	
55	00110111	7	
56	00111000	8	
57	00111001	9	
58	00111010	:	dos puntos
59	00111011	;	punto y coma
60	00111100	<	menor que
61	00111101	=	igual
62	00111110	>	mayor que
63	00111111	?	interrogación
64	01000000	@	AT comercial
65	01000001	A	
66	01000010	B	
67	01000011	C	
68	01000100	D	
69	01000101	E	
70	01000110	F	
71	01000111	G	
72	01001000	H	
73	01001001	I	
74	01001010	J	
75	01001011	K	
76	01001100	L	
77	01001101	M	
78	01001110	N	
79	01001111	O	
80	01010000	P	
81	01010001	Q	

Secuencia ASCII ascendente (*cont.*)

<i>Representación decimal</i>	<i>Representación binaria</i>	<i>Símbolo</i>	<i>Significado</i>
82	01010010	R	
83	01010011	S	
84	01010100	T	
85	01010101	U	
86	01010110	V	
87	01010111	W	
88	01011000	X	
89	01011001	Y	
90	01011010	Z	
91	01011011	[	abre corchete
92	01011100	\	barra inversa
93	01011101	]	cierra corchete
94	01011110	^	circunflejo, NOT lógico
95	01011111	_	subrayado
96	01100000	`	acentro grave
97	01100001	a	
98	01100010	b	
99	01100011	c	
100	01100100	d	
101	01100101	e	
102	01100110	f	
103	01100111	g	
104	01101000	h	
105	01101001	i	
106	01101010	j	
107	01101011	k	
108	01101100	l	
109	01101101	m	
110	01101110	n	
111	01101111	o	